

```

import { Ionicons } from '@expo/vector-icons';
import { Tabs } from 'expo-router/js-tabs';
import type { BottomTabBarProps } from 'expo-router/js-tabs';
import { useFocusEffect } from 'expo-router';
import { useCallback, useEffect, useRef } from 'react';
import { Animated, Easing, Pressable, StyleSheet, View } from 'react-native';
import { colors } from '../../src/theme';
import { useRole } from '../../src/context/RoleContext';
/** 企业端默认落地「岗位」Tab（与求职端首屏一致） */
export const unstable_settings = {
  initialRouteName: 'job-management',
};
/** 企业端底部 Tab 与求职端共用品牌主色 primary; 未选中 -outline, 选中填充实心 */
const ICON_MAP = {
  'enterprise-overview': { outline: 'person-outline', filled: 'person' },
  'job-management': { outline: 'folder-outline', filled: 'folder' },
  'talent-recommendation': { outline: 'people-outline', filled: 'people' },
  'enterprise-chat': { outline: 'chatbubble-outline', filled: 'chatbubble' },
} as const;
type TabRoute = keyof typeof ICON_MAP;
function TabItem({
  focused,
  route,
  label,
  onPress,
  onLongPress,
}: {
  focused: boolean;
  route: TabRoute;
  label: string;
  onPress: () => void;
  onLongPress: () => void;
}) {
  const { outline, filled } = ICON_MAP[route];
  const translateY = useRef(new Animated.Value(focused ? -2 : 0)).current;
  const progress = useRef(new Animated.Value(focused ? 1 : 0)).current;
  useEffect(() => {
    const translateAnim = Animated.timing(translateY, {
      toValue: focused ? -2 : 0,
      duration: 150,
      easing: Easing.out(Easing.ease),
      useNativeDriver: true,
    });
    const progressAnim = Animated.timing(progress, {
      toValue: focused ? 1 : 0,
      duration: 200,
      easing: Easing.inOut(Easing.ease),
      useNativeDriver: true,
    });
    translateAnim.start();
    progressAnim.start();
    return () => {
      translateAnim.stop();
      progressAnim.stop();
    };
  }, [focused, translateY, progress]);
  const inactiveOpacity = progress.interpolate({
    inputRange: [0, 1],
    outputRange: [1, 0],
  });
  return (
    <Pressable
      style={({ pressed }) => [styles.tabItem, pressed && styles.tabItemPressed]}
      onPress={onPress}
      onLongPress={onLongPress}
      accessibilityRole="button"
      accessibilityState={{ selected: focused }}
    >
      <Animated.View style={({ transform: [{ translateY }] })>
        <View style={styles.iconBox}>
          <Animated.View
            style={[StyleSheet.absoluteFill, styles.center, { opacity: inactiveOpacity }]}

```

```

    >
    <Icons name={outline} size={22} color={colors.textHint} />
  </Animated.View>
  <Animated.View style={[StyleSheet.absoluteFill, styles.center, { opacity: progress }]}>
    <Icons name={filled} size={22} color={colors.primary} />
  </Animated.View>
</View>
</Animated.View>
<View style={styles.labelBox}>
  <Animated.Text style={[styles.tabLabel, styles.labelLayer, { opacity: inactiveOpacity }]}>
    {label}
  </Animated.Text>
  <Animated.Text
    style={[styles.tabLabel, styles.tabLabelActive, styles.labelLayer, { opacity: progress }]}
  >
    {label}
  </Animated.Text>
</View>
</Pressable>
);
}
function TabBar({ state, descriptors, navigation, insets }: BottomTabBarProps) {
  return (
    <View style={[styles.tabBar, { paddingBottom: insets.bottom + 4 }]}>
      {state.routes.map((route, index) => {
        const { options } = descriptors[route.key];
        const focused = state.index === index;
        const label =
          typeof options.tabBarLabel === 'string' ? options.tabBarLabel : options.title ?? '';
        const onPress = () => {
          const event = navigation.emit({
            type: 'tabPress',
            target: route.key,
            canPreventDefault: true,
          });
          if (!focused && !event.defaultPrevented) {
            navigation.navigate(route.name);
          }
        };
        const onLongPress = () => navigation.emit({ type: 'tabLongPress', target: route.key });
        return (
          <TabItem
            key={route.key}
            route={route.name as TabRoute}
            focused={focused}
            label={label}
            onPress={onPress}
            onLongPress={onLongPress}
          />
        );
      })}
    </View>
  );
}
/**
 * 企业端底部 Tab 导航：岗位 / 人才 / 沟通 / 我的
 * 绿色 success 强调，与求职端蓝色 primary 形成视觉区分。
 */
export default function EnterpriseTabLayout() {
  const { setRole } = useRole();
  // 用 useFocusEffect：进入/返回该端时随焦点写入角色，
  // 避免「切换到企业端」后返回时 layout 未重新挂载导致 role 残留 candidate。
  useFocusEffect(
    useCallback(() => {
      setRole('enterprise');
    }, [setRole]),
  );
  return (
    <Tabs
      screenOptions={{ headerShown: false }}
      tabBar={(props) => <TabBar {...props} />}
    >

```

```

    <Tabs.Screen name="job-management" options={{ title: '岗位' }} />
    <Tabs.Screen name="talent-recommendation" options={{ title: '人才' }} />
    <Tabs.Screen name="enterprise-chat" options={{ title: '沟通' }} />
    <Tabs.Screen name="enterprise-overview" options={{ title: '我的' }} />
  </Tabs>
);
}
const styles = StyleSheet.create({
  tabBar: {
    flexDirection: 'row',
    alignItems: 'center',
    backgroundColor: colors.white,
    paddingTop: 4,
  },
  tabItem: {
    flex: 1,
    alignItems: 'center',
    justifyContent: 'center',
    paddingVertical: 2,
  },
  tabItemPressed: {
    transform: [{ scale: 0.92 }],
  },
  iconBox: {
    width: 22,
    height: 22,
  },
  center: {
    alignItems: 'center',
    justifyContent: 'center',
  },
  labelBox: {
    height: 14,
    marginTop: 1,
    alignSelf: 'stretch',
  },
  labelLayer: {
    position: 'absolute',
    top: 0,
    left: 0,
    right: 0,
  },
  tabLabel: {
    fontSize: 11,
    lineHeight: 14,
    fontWeight: '400',
    textAlign: 'center',
    color: colors.textHint,
  },
  tabLabelActive: {
    fontWeight: '600',
    color: colors.primary,
  },
});
import { Ionicons } from '@expo/vector-icons';
import { useRouter } from 'expo-router';
import {
  ActivityIndicator,
  Pressable,
  ScrollView,
  StyleSheet,
  Text,
  TextInput,
  View,
} from 'react-native';
import { SafeAreaView } from 'react-native-safe-area-context';
import { useEnterpriseCommunication } from '../src/hooks';
import { ENTERPRISE_DISPLAY_NAME } from '../src/constants';
import { colors } from '../src/theme';
import type { EnterpriseChatItem, EnterpriseCommunicationTab, PendingTask } from '../src/types';
/* ===== 子组件: 顶部导航 ===== */
function Header({ companyName, onBack }: { companyName: string; onBack: () => void }) {

```

```

const router = useRouter();
return (
  <View style={styles.header}>
    <Pressable style={styles.headerBtn} onPress={onBack} hitSlop={8}>
      <Icons name="chevron-back" size={24} color={colors.textPrimary} />
    </Pressable>
    <View style={styles.headerCenter}>
      <Text style={styles.headerCompany}>{companyName}</Text>
      <Text style={styles.headerTitle}>沟通</Text>
    </View>
    <Pressable
      style={styles.startBtn}
      onPress={() => router.navigate('/talent-recommendation')}
      hitSlop={6}
    >
      <Icons name="add" size={16} color={colors.white} />
      <Text style={styles.startBtnText}>发起沟通</Text>
    </Pressable>
  </View>
);
}
/* ===== 子组件: Tab 栏 ===== */
function TabBar({
  tabs,
  active,
  onChange,
}): {
  tabs: EnterpriseCommunicationTab[];
  active: string;
  onChange: (key: string) => void;
} {
  return (
    <View style={styles.tabBar}>
      {tabs.map((tab) => {
        const isActive = tab.key === active;
        return (
          <Pressable key={tab.key} style={styles.tabItem} onPress={() => onChange(tab.key)}>
            <Text style={[styles.tabLabel, isActive && styles.tabLabelActive]}>
              {tab.label}
            <Text style={[styles.tabCount, isActive && styles.tabCountActive]}>{tab.count}</Text>
          </Text>
          <View style={[styles.tabIndicator, isActive && styles.tabIndicatorActive]} />
        </Pressable>
      )};
    </View>
  );
}
/* ===== 子组件: 待处理任务 ===== */
function PendingTasks({
  tasks,
  onPressTask,
}): {
  tasks: PendingTask[];
  onPressTask: (task: PendingTask) => void;
} {
  return (
    <View style={styles.pendingCard}>
      <Text style={styles.pendingTitle}>待处理</Text>
      <View style={styles.pendingGrid}>
        {tasks.map((task) => {
          const token =
            task.color === 'action'
              ? colors.action
              : task.color === 'compareOrange'
                ? colors.compareOrange
                : colors.primary;
          return (
            <Pressable
              key={task.id}
              style={styles.pendingItem}
              onPress={() => onPressTask(task)}
            >

```

```

    >
    <View style={[styles.pendingIconWrap, { backgroundColor: token + '15' }]}>
      <Icons name={task.icon as keyof typeof Icons.glyphMap} size={22} color={token} />
    </View>
    <View style={styles.pendingBody}>
      <View style={styles.pendingLabelRow}>
        <Text style={styles.pendingLabel}>{task.label}</Text>
        <View style={styles.pendingBadge}>
          <Text style={styles.pendingBadgeText}>{task.count}人</Text>
        </View>
      </View>
      {task.hint ? <Text style={styles.pendingHint}>{task.hint}</Text> : null}
    </View>
    <Icons name="chevron-forward" size={16} color={colors.textHint} />
  </Pressable>
);
  }}
</View>
</View>
);
}
/* ===== 子组件: 聊天列表项 ===== */
function ChatRow({ chat }: { chat: EnterpriseChatItem }) {
  const router = useRouter();
  return (
    <Pressable style={styles.chatRow} onPress={() => router.push('/chat/' + chat.id)}>
      <View style={styles.chatAvatar}>
        <Text style={styles.chatAvatarText}>{chat.avatar}</Text>
      </View>
      <View style={styles.chatBody}>
        <View style={styles.chatHeader}>
          <View style={styles.chatNameWrap}>
            <Text style={styles.chatName}>{chat.name}</Text>
            {chat.labels.length > 0 ? (
              <View style={styles.chatLabelTag}>
                <Text style={styles.chatLabelTagText}>{chat.labels[0]}</Text>
              </View>
            ) : null}
          </View>
          <Text style={styles.chatTime}>{chat.time}</Text>
        </View>
        <Text style={styles.chatTitle}>{chat.title}</Text>
        <View style={styles.chatFooter}>
          <Text style={styles.chatLastMessage} numberOfLines={1}>
            {chat.lastMessage}
          </Text>
          {chat.unreadCount > 0 ? (
            <View style={styles.unreadBadge}>
              <Text style={styles.unreadBadgeText}>{chat.unreadCount}</Text>
            </View>
          ) : null}
        </View>
      </View>
    </Pressable>
  );
}
/* ===== 子组件: 搜索栏 ===== */
function SearchBar({
  value,
  onChange,
  onClear,
}: {
  value: string;
  onChange: (text: string) => void;
  onClear: () => void;
}) {
  return (
    <View style={styles.searchBar}>
      <Icons name="search-outline" size={18} color={colors.textHint} />
      <TextInput
        style={styles.searchInput}
        value={value}

```

```

        onChangeText={onChange}
        placeholder="搜索候选人姓名 / 职位 / 消息"
        placeholderTextColor={colors.textHint}
        returnKeyType="search"
    />
    {value !== '' && (
        <Pressable onPress={onClear} hitSlop={8}>
            <Ionicons name="close-circle" size={16} color={colors.textHint} />
        </Pressable>
    )}
</View>
);
}
/* ===== 子组件: 快捷操作栏 ===== */
/** 单个快捷操作。`route` 必填 — 每个入口都必须有落点, 没有目标页的宁可不放 */
type QuickActionItem = { icon: string; label: string; route: string };
function QuickActions({
    actions,
    onPressAction,
}): {
    actions: QuickActionItem[];
    onPressAction: (action: QuickActionItem) => void;
}) {
    return (
        <View style={styles.quickActionsCard}>
            {actions.map((action) => (
                <Pressable key={action.label} style={styles.quickAction} onPress={() => onPressAction(action)}>
                    <Ionicons name={action.icon as keyof typeof Ionicons.glyphMap} size={20}
color={colors.textSecondary}
/>
                    <Text style={styles.quickActionText}>{action.label}</Text>
                </Pressable>
            ))}
        </View>
    );
}
/* ===== 子组件: 空状态 ===== */
function EmptyState({
    dataReference,
    onDiscover,
}): {
    dataReference: { label: string; value: string }[];
    onDiscover: () => void;
}) {
    return (
        <View style={styles.emptyCard}>
            <View style={styles.emptyIconWrap}>
                <Ionicons name="chatbubbles-outline" size={48} color={colors.primary} />
            </View>
            <Text style={styles.emptyTitle}>还没有沟通记录</Text>
            <Text style={styles.emptySubtitle}>从人才推荐中发现合适的候选人, 主动发起沟通吧</Text>
            <Pressable style={styles.emptyBtn} onPress={onDiscover} hitSlop={6}>
                <Text style={styles.emptyBtnText}>去发现人才</Text>
            </Pressable>
            <View style={styles.dataCard}>
                <Text style={styles.dataTitle}>数据参考</Text>
                {dataReference.map((item, index) => (
                    <View key={index} style={styles.dataRow}>
                        <Ionicons name="trending-up-outline" size={16} color={colors.success} />
                        <Text style={styles.dataText}>
                            {item.label}
                        <Text style={styles.dataValue}> {item.value}</Text>
                    </Text>
                </View>
                ))}
            </View>
        </View>
    );
}
/* ===== 页面 ===== */
/*
* 「批量发 Offer」「群发通知」原先在这里, 但两者都没有对应页面, 点了只能弹「功能开发中」。

```

```

* 按「不留假交互」的口径摘掉（与 talent-recommendation.tsx 的 POOL_ACTIONS 同一处理）。
*/
const QUICK_ACTIONS: QuickActionItem[] = [
  { icon: 'mail-outline', label: '批量发邀请', route: '/talent-recommendation' },
  { icon: 'calendar-outline', label: '面试评分', route: '/interview-score' },
];
export default function EnterpriseChatScreen() {
  const router = useRouter();
  const {
    communication,
    activeTab,
    setActiveTab,
    keyword,
    setKeyword,
    chats,
    isEmptySearch,
    loading,
    error,
    refresh,
  } = useEnterpriseCommunication();
  /** 快捷操作：每个入口都由数据保证有落点，页面不再需要「没页面就弹提示」的分支 */
  const handleQuickAction = (action: QuickActionItem) => {
    router.push(action.route);
  };
  // 待办卡片：跳页面或切 Tab，目标由接口层给，页面不写死映射。
  // PendingTask 是联合类型，两者必居其一，故不需要「都没有」的兜底分支
  const handlePressTask = (task: PendingTask) => {
    if (task.tabKey) {
      setActiveTab(task.tabKey);
      setKeyword('');
      return;
    }
    router.push(task.route as never);
  };
  if (loading) {
    return (
      <SafeAreaView style={styles.container} edges={['top']}>
        <Header companyName={ENTERPRISE_DISPLAY_NAME} onBack={() => router.back()} />
        <View style={styles.center}>
          <ActivityIndicator size="large" color={colors.primary} />
        </View>
      </SafeAreaView>
    );
  }
  if (error || !communication) {
    return (
      <SafeAreaView style={styles.container} edges={['top']}>
        <Header companyName={ENTERPRISE_DISPLAY_NAME} onBack={() => router.back()} />
        <View style={styles.center}>
          <Ionicons name="alert-circle-outline" size={40} color={colors.danger} />
          <Text style={styles.tip}>加载失败</Text>
          <Pressable style={styles.retryBtn} onPress={refresh}>
            <Text style={styles.retryText}>重试</Text>
          </Pressable>
        </View>
      </SafeAreaView>
    );
  }
  return (
    <SafeAreaView style={styles.container} edges={['top', 'bottom']}>
      <Header companyName={communication.companyName} onBack={() => router.back()} />
      <TabBar tabs={communication.tabs} active={activeTab} onChange={setActiveTab} />
      <ScrollView contentContainerStyle={styles.body} showsVerticalScrollIndicator={false}>
        {communication.hasData ? (
          <View>
            <PendingTasks tasks={communication.pendingTasks} onPressTask={handlePressTask} />
            <SearchBar value={keyword} onChange={setKeyword} onClear={() => setKeyword('')} />
            {chats.length === 0 ? (
              <View style={styles.emptyList}>
                <Ionicons name="chatbubbles-outline" size={32} color={colors.textHint} />
                <Text style={styles.emptyListText}>
                  {isEmptySearch ? `没有匹配「${keyword.trim()}」的会话` : `这个分类下暂时没有会话`}
                </Text>
              </View>
            ) : (
              <List>
                {chats.map(chat => (
                  <ListItem>
                    <Text>{chat.sender}</Text>
                    <Text>{chat.message}</Text>
                    <Text>{chat.time}</Text>
                  </ListItem>
                ))}
              </List>
            )}
          </View>
        ) : (
          <View>
            <Text>暂无数据</Text>
          </View>
        )}
      </ScrollView>
    </SafeAreaView>
  );
}

```

```

        </Text>
      </View>
    ) : (
      <View style={styles.chatListCard}>
        {chats.map((chat, index) => {
          const isLast = index === chats.length - 1;
          return (
            <View key={chat.id} style={isLast ? undefined : styles.chatBorder}>
              <ChatRow chat={chat} />
            </View>
          );
        })}
      </View>
    )}
    <QuickActions actions={QUICK_ACTIONS} onPressAction={handleQuickAction} />
  </View>
) : (
  <EmptyState
    dataReference={communication.dataReference}
    onDiscover={() => router.navigate('/talent-recommendation')}
  />
)
</ScrollView>
</SafeAreaView>
);
}
const styles = StyleSheet.create({
  container: { flex: 1, backgroundColor: colors.pageBg },
  header: {
    flexDirection: 'row',
    alignItems: 'center',
    justifyContent: 'space-between',
    paddingHorizontal: 12,
    paddingVertical: 12,
    backgroundColor: colors.white,
  },
  headerBtn: { width: 40, alignItems: 'center' },
  headerCenter: { alignItems: 'center' },
  headerCompany: { fontSize: 12, color: colors.textSecondary },
  headerTitle: { fontSize: 17, fontWeight: '700', color: colors.textPrimary, marginTop: 2 },
  startBtn: {
    flexDirection: 'row',
    alignItems: 'center',
    backgroundColor: colors.primaryBtn,
    paddingHorizontal: 10,
    paddingVertical: 6,
    borderRadius: 16,
  },
  startBtnText: {
    color: colors.white,
    fontSize: 12,
    fontWeight: '600',
    marginLeft: 2,
  },
  tabBar: {
    flexDirection: 'row',
    backgroundColor: colors.white,
    borderBottomWidth: 1,
    borderBottomColor: colors.borderLight,
  },
  tabItem: {
    flex: 1,
    alignItems: 'center',
    paddingTop: 8,
  },
  tabLabel: {
    fontSize: 13,
    color: colors.textSecondary,
    paddingBottom: 8,
  },
  tabLabelActive: {
    color: colors.textPrimary,
  },

```

```

    fontWeight: '600',
  },
  tabCount: {
    fontSize: 12,
    color: colors.textHint,
    marginLeft: 2,
  },
  tabCountActive: {
    color: colors.primary,
  },
  tabIndicator: {
    height: 2,
    width: 20,
    borderRadius: 1,
    backgroundColor: 'transparent',
    marginBottom: -1,
  },
  tabIndicatorActive: {
    backgroundColor: colors.primary,
  },
  body: {
    padding: 8,
    paddingBottom: 24,
  },
  center: {
    flex: 1,
    alignItems: 'center',
    justifyContent: 'center',
    gap: 12,
  },
  tip: { fontSize: 14, color: colors.textSecondary },
  retryBtn: {
    marginTop: 8,
    paddingHorizontal: 24,
    paddingVertical: 8,
    borderRadius: 18,
    backgroundColor: colors.primaryBtn,
  },
  retryText: { color: colors.white, fontSize: 13, fontWeight: '600' },
  pendingCard: {
    backgroundColor: colors.white,
    borderRadius: 8,
    padding: 8,
    marginBottom: 8,
  },
  pendingTitle: {
    fontSize: 15,
    fontWeight: '700',
    color: colors.textPrimary,
    marginBottom: 12,
  },
  pendingGrid: {
    gap: 8,
  },
  pendingItem: {
    flexDirection: 'row',
    alignItems: 'center',
    backgroundColor: colors.pageBg,
    borderRadius: 8,
    padding: 8,
  },
  pendingIconWrap: {
    width: 40,
    height: 40,
    borderRadius: 10,
    alignItems: 'center',
    justifyContent: 'center',
    marginRight: 12,
  },
  pendingBody: {
    flex: 1,
  },

```

```

pendingLabelRow: {
  flexDirection: 'row',
  alignItems: 'center',
},
pendingLabel: {
  fontSize: 14,
  fontWeight: '600',
  color: colors.textPrimary,
},
pendingBadge: {
  backgroundColor: colors.actionLight,
  borderRadius: 10,
  paddingHorizontal: 8,
  paddingVertical: 2,
  marginLeft: 8,
},
pendingBadgeText: {
  fontSize: 12,
  fontWeight: '700',
  color: colors.actionText,
},
pendingHint: {
  fontSize: 12,
  color: colors.textSecondary,
  marginTop: 2,
},
searchBar: {
  flexDirection: 'row',
  alignItems: 'center',
  backgroundColor: colors.white,
  borderRadius: 10,
  paddingHorizontal: 12,
  paddingVertical: 10,
  marginBottom: 12,
},
searchInput: {
  flex: 1,
  marginLeft: 8,
  fontSize: 14,
  color: colors.textPrimary,
  padding: 0,
},
emptyList: {
  alignItems: 'center',
  gap: 10,
  padding: 8,
  backgroundColor: colors.white,
  borderRadius: 8,
},
emptyListText: { fontSize: 13, color: colors.textSecondary },
chatListCard: {
  backgroundColor: colors.white,
  borderRadius: 8,
  overflow: 'hidden',
  marginBottom: 8,
},
chatBorder: {
  borderBottomWidth: 1,
  borderBottomColor: colors.borderLight,
},
chatRow: {
  flexDirection: 'row',
  alignItems: 'center',
  padding: 14,
},
chatAvatar: {
  width: 48,
  height: 48,
  borderRadius: 24,
  backgroundColor: colors.primaryLight,
  alignItems: 'center',
  justifyContent: 'center',

```

```

    marginRight: 12,
  },
  chatAvatarText: {
    fontSize: 18,
    fontWeight: '700',
    color: colors.primary,
  },
  chatBody: {
    flex: 1,
  },
  chatHeader: {
    flexDirection: 'row',
    alignItems: 'center',
    justifyContent: 'space-between',
  },
  chatNameWrap: {
    flexDirection: 'row',
    alignItems: 'center',
  },
  chatName: {
    fontSize: 15,
    fontWeight: '700',
    color: colors.textPrimary,
  },
  chatLabelTag: {
    backgroundColor: colors.compareOrange + '20',
    borderRadius: 8,
    paddingHorizontal: 6,
    paddingVertical: 2,
    marginLeft: 8,
  },
  chatLabelTagText: {
    fontSize: 11,
    color: colors.compareOrange,
    fontWeight: '600',
  },
  chatTime: {
    fontSize: 12,
    color: colors.textHint,
  },
  chatTitle: {
    fontSize: 13,
    color: colors.textSecondary,
    marginTop: 2,
  },
  chatFooter: {
    flexDirection: 'row',
    alignItems: 'center',
    marginTop: 4,
  },
  chatLastMessage: {
    flex: 1,
    fontSize: 13,
    color: colors.textHint,
  },
  unreadBadge: {
    minWidth: 18,
    height: 18,
    borderRadius: 9,
    backgroundColor: colors.danger,
    alignItems: 'center',
    justifyContent: 'center',
    marginLeft: 8,
  },
  unreadBadgeText: {
    fontSize: 11,
    color: colors.white,
    fontWeight: '700',
  },
  quickActionsCard: {
    flexDirection: 'row',
    backgroundColor: colors.white,

```

```

        borderRadius: 8,
        paddingVertical: 8,
    },
    quickAction: {
        flex: 1,
        alignItems: 'center',
    },
    quickActionText: {
        fontSize: 11,
        color: colors.textSecondary,
        marginTop: 4,
    },
    emptyCard: {
        backgroundColor: colors.white,
        borderRadius: 8,
        padding: 8,
        alignItems: 'center',
    },
    emptyIconWrap: {
        width: 88,
        height: 88,
        borderRadius: 44,
        backgroundColor: colors.primaryLight,
        alignItems: 'center',
        justifyContent: 'center',
        marginBottom: 16,
    },
    emptyTitle: {
        fontSize: 18,
        fontWeight: '700',
        color: colors.textPrimary,
        marginBottom: 8,
    },
    emptySubtitle: {
        fontSize: 13,
        color: colors.textSecondary,
        textAlign: 'center',
        marginBottom: 20,
    },
    emptyBtn: {
        backgroundColor: colors.primaryBtn,
        paddingHorizontal: 32,
        paddingVertical: 12,
        borderRadius: 24,
        marginBottom: 24,
    },
    emptyBtnText: {
        color: colors.white,
        fontSize: 15,
        fontWeight: '600',
    },
    dataCard: {
        width: '100%',
        backgroundColor: colors.primaryLight,
        borderRadius: 8,
        padding: 8,
        alignItems: 'flex-start',
    },
    dataTitle: {
        fontSize: 14,
        fontWeight: '700',
        color: colors.primaryText,
        marginBottom: 12,
    },
    dataRow: {
        flexDirection: 'row',
        alignItems: 'flex-start',
        marginBottom: 10,
    },
    dataText: {
        flex: 1,
        fontSize: 13,
    }

```

```

    color: colors.textPrimary,
    marginLeft: 10,
    lineHeight: 18,
  },
  dataValue: {
    color: colors.success,
    fontWeight: '700',
  },
});
import { Ionicons } from '@expo/vector-icons';
import { LinearGradient } from 'expo-linear-gradient';
import { useRouter } from 'expo-router';
import {
  Pressable,
  ScrollView,
  StyleSheet,
  Text,
  View,
} from 'react-native';
import { SafeAreaView } from 'react-native-safe-area-context';
import { useEnterpriseOverview } from '../src/hooks';
import { LoadingState, ErrorState } from '../src/components/StateView';
import { useRole } from '../src/context/RoleContext';
import { colors, headerGradient, semanticColorTokens } from '../src/theme';
import type { EnterpriseOverview, EnterpriseQuickAction, EnterpriseStat, JobEvaluationSummary, MenuItem }
from '../src/types';
/* ===== 子组件: 顶部导航 ===== */
function Header({ onSettings }: { onSettings: () => void }) {
  return (
    <View style={styles.header}>
      <View style={styles.headerBtn} />
      <Text style={styles.headerTitle}>我的</Text>
      <Pressable style={styles.headerBtn} onPress={onSettings} hitSlop={8}>
        <Ionicons name="settings-outline" size={22} color={colors.textInverse} />
      </Pressable>
    </View>
  );
}
/* ===== 子组件: 企业信息卡 ===== */
function CompanyCard({ overview }: { overview: EnterpriseOverview }) {
  return (
    <View style={styles.companyCard}>
      <View style={styles.companyHeader}>
        <View style={styles.logoBox}>
          <Text style={styles.logoText}>{overview.logo}</Text>
        </View>
        <View style={styles.companyMeta}>
          <Text style={styles.companyName}>{overview.name}</Text>
          <Text style={styles.companyTags}>
            {overview.industry} · {overview.scale} · {overview.stage}
          </Text>
        </View>
      </View>
      <View style={styles.scoreRow}>
        <View style={styles.scoreBox}>
          <Text style={styles.scoreValue}>{overview.overallScore}</Text>
          <Text style={styles.scoreLabel}>综合评分</Text>
        </View>
        <View style={styles.divider} />
        <View style={styles.scoreBox}>
          <Text style={styles.scoreValue}>{overview.reviewCount}</Text>
          <Text style={styles.scoreLabel}>评价人数</Text>
        </View>
        <View style={styles.divider} />
        <View style={styles.scoreBox}>
          <Text style={styles.scoreValue}>{overview.stats[0]?.value ?? 0}</Text>
          <Text style={styles.scoreLabel}>在招岗位</Text>
        </View>
      </View>
    </View>
  );
}

```

```

/* ===== 子组件: 统计四宫格 ===== */
function StatGrid({ stats }: { stats: EnterpriseStat[] }) {
  return (
    <View style={styles.card}>
      <View style={styles.statGrid}>
        {stats.map((stat) => {
          const color = semanticColorTokens[stat.color] ?? colors.primary;
          const up = stat.change >= 0;
          return (
            <View key={stat.id} style={styles.statItem}>
              <View style={[styles.statIcon, { backgroundColor: color + '15' }]}>
                <Ionicons name={stat.icon as keyof typeof Ionicons.glyphMap} size={20} color={color} />
              </View>
              <Text style={styles.statValue}>{stat.value.toLocaleString()}</Text>
              <Text style={styles.statLabel}>{stat.label}</Text>
              <Text style={[styles.statChange, { color: up ? colors.success : colors.danger }]}>
                {up ? '↑' : '↓'} {Math.abs(stat.change)}
              </Text>
            </View>
          );
        })}
      </View>
    </View>
  );
}

/* ===== 子组件: 快捷入口 ===== */
function QuickActions({ actions }: { actions: EnterpriseQuickAction[] }) {
  const router = useRouter();
  return (
    <View style={styles.card}>
      <Text style={styles.cardTitle}>快捷入口</Text>
      <View style={styles.quickGrid}>
        {actions.map((action) => {
          const color = semanticColorTokens[action.color] ?? colors.primary;
          return (
            <Pressable
              key={action.id}
              style={styles.quickItem}
              onPress={() => action.route && router.push(action.route as string)}
            >
              <View style={[styles.quickIcon, { backgroundColor: color + '15' }]}>
                <Ionicons name={action.icon as keyof typeof Ionicons.glyphMap} size={22} color={color} />
              </View>
              <Text style={styles.quickLabel}>{action.label}</Text>
            </Pressable>
          );
        })}
      </View>
    </View>
  );
}

/* ===== 子组件: 岗位评价列表 ===== */
function JobEvaluationList({ jobs }: { jobs: JobEvaluationSummary[] }) {
  const router = useRouter();
  return (
    <View style={styles.card}>
      <View style={styles.cardHeader}>
        <Text style={styles.cardTitle}>岗位评价管理</Text>
        <Pressable onPress={() => router.push('/job-evaluations')}>
          <Text style={styles.cardMore}>查看全部</Text>
        </Pressable>
      </View>
      {jobs.map((job) => {
        const up = job.change >= 0;
        return (
          <View key={job.id} style={styles.jobRow}>
            <View style={styles.jobInfo}>
              <Text style={styles.jobName}>{job.positionName}</Text>
              <Text style={styles.jobDepartment}>{job.department}</Text>
            </View>
            <View style={styles.jobScoreBox}>
              <Text style={styles.jobScore}>{job.score.toFixed(1)}</Text>
            </View>
          </View>
        );
      })}
    </View>
  );
}

```

```

        <Text style={[styles.jobChange, { color: up ? colors.success : colors.danger }]}>
            {up ? '↑' : '↓'} {Math.abs(job.change).toFixed(1)}
        </Text>
    </View>
</View>
);
}}}
</View>
);
}
/* ===== 子组件: 管理后台 ===== */
/**
 * 管理后台入口：运营规则配置 / 数据审计等低频管理操作。
 * 单独成区纵向排列，既不挤占快捷入口的横排位，也避免后台术语出现在求职者端。
 */
function ManageList({ actions }: { actions: EnterpriseQuickAction[] }) {
    const router = useRouter();
    if (actions.length === 0) return null;
    return (
        <View style={styles.card}>
            <Text style={styles.standaloneTitle}>管理后台</Text>
            {actions.map((action) => {
                const color = semanticColorTokens[action.color] ?? colors.primary;
                return (
                    <Pressable
                        key={action.id}
                        style={styles.manageRow}
                        onPress={() => action.route && router.push(action.route as string)}
                    >
                        <View style={[styles.manageIcon, { backgroundColor: color + '15' }]}>
                            <Ionicons name={action.icon as keyof typeof Ionicons.glyphMap} size={20} color={color} />
                        </View>
                        <Text style={styles.manageLabel}>{action.label}</Text>
                        <Ionicons name="chevron-forward" size={18} color={colors.textHint} />
                    </Pressable>
                );
            })}
        </View>
    );
}
/* ===== 子组件: 企业功能菜单 ===== */
/**
 * 企业端功能菜单（更多服务）：发布岗位 / 岗位管理 / 人才推荐等业务功能。
 * 与求职者端菜单严格隔离，由 AppRole='enterprise' 下发，避免角色混淆。
 */
function MenuList({ items }: { items: MenuItem[] }) {
    const router = useRouter();
    if (items.length === 0) return null;
    return (
        <View style={styles.card}>
            <Text style={styles.standaloneTitle}>更多服务</Text>
            {items.map((item) => (
                <Pressable
                    key={item.id}
                    style={styles.manageRow}
                    onPress={() => item.route && router.push(item.route as string)}
                >
                    <View style={[styles.manageIcon, { backgroundColor: colors.primary + '15' }]}>
                        <Ionicons name={item.icon as keyof typeof Ionicons.glyphMap} size={20} color={colors.primary} />
                    </View>
                    <Text style={styles.manageLabel}>{item.title}</Text>
                    <Ionicons name="chevron-forward" size={18} color={colors.textHint} />
                </Pressable>
            ))}
        </View>
    );
}
/* ===== 子组件: 切换到求职端卡片 ===== */
/**
 * 企业端「我的」页回切求职端，与求职端「切换到企业端」卡片对称，
 * 避免经「设置→身份切换」进入企业端后返回栈为空、回不去求职端。
 */

```

```

function CandidateEntryCard() {
  const router = useRouter();
  const { setRole } = useRole();
  return (
    <Pressable
      style={styles.candidateCard}
      onPress={() => {
        setRole('candidate');
        router.replace('/(tabs)/jobs');
      }}
      hitSlop={4}
    >
      <View style={styles.candidateIconWrap}>
        <Ionicons name="person-outline" size={22} color={colors.primary} />
      </View>
      <View style={styles.candidateText}>
        <Text style={styles.candidateTitle}>切换到求职端</Text>
        <Text style={styles.candidateDesc}>返回求职者端，浏览岗位与你的竞争力数据</Text>
      </View>
      <Text style={styles.candidateAction}>去求职端 →</Text>
    </Pressable>
  );
}
/* ===== 页面主组件 ===== */
export default function EnterpriseOverviewScreen() {
  const router = useRouter();
  const { overview, loading, error, refresh } = useEnterpriseOverview();
  if (loading) {
    return (
      <SafeAreaView style={styles.loadingContainer} edges={['top']}>
        <LoadingState text="加载中..." />
      </SafeAreaView>
    );
  }
  if (error || !overview) {
    return (
      <SafeAreaView style={styles.loadingContainer} edges={['top']}>
        <ErrorState onRetry={refresh} />
      </SafeAreaView>
    );
  }
  return (
    <View style={styles.container}>
      <LinearGradient colors={headerGradient} style={styles.gradientHeader}>
        <SafeAreaView edges={['top']}>
          <Header onSettings={() => router.push('/settings')} />
          <CompanyCard overview={overview} />
        </SafeAreaView>
      </LinearGradient>
      <ScrollView style={styles.scroll} contentContainerStyle={styles.scrollContent}>
        <CandidateEntryCard />
        <StatGrid stats={overview.stats} />
        <QuickActions actions={overview.quickActions} />
        <JobEvaluationList jobs={overview.jobEvaluations} />
        <ManageList actions={overview.manageActions} />
        <MenuList items={overview.menuItems} />
      </ScrollView>
    </View>
  );
}
/* ===== 样式 ===== */
const styles = StyleSheet.create({
  container: {
    flex: 1,
    backgroundColor: colors.pageBg,
  },
  loadingContainer: {
    flex: 1,
    justifyContent: 'center',
    alignItems: 'center',
    backgroundColor: colors.pageBg,
  },
});

```

```

gradientHeader: {
  paddingBottom: 16,
},
header: {
  flexDirection: 'row',
  alignItems: 'center',
  justifyContent: 'space-between',
  paddingHorizontal: 16,
  paddingVertical: 12,
},
headerBtn: {
  width: 36,
  height: 36,
  justifyContent: 'center',
  alignItems: 'center',
},
headerTitle: {
  fontSize: 17,
  fontWeight: '600',
  color: colors.textInverse,
},
companyCard: {
  marginHorizontal: 8,
  marginTop: 8,
  padding: 8,
  backgroundColor: colors.white,
  borderRadius: 8,
},
companyHeader: {
  flexDirection: 'row',
  alignItems: 'center',
  marginBottom: 16,
},
logoBox: {
  width: 52,
  height: 52,
  borderRadius: 12,
  backgroundColor: colors.primaryLight,
  justifyContent: 'center',
  alignItems: 'center',
  marginRight: 12,
},
logoText: {
  fontSize: 22,
  fontWeight: '700',
  color: colors.primary,
},
companyMeta: {
  flex: 1,
},
companyName: {
  fontSize: 18,
  fontWeight: '700',
  color: colors.textPrimary,
  marginBottom: 4,
},
companyTags: {
  fontSize: 13,
  color: colors.textSecondary,
},
scoreRow: {
  flexDirection: 'row',
  alignItems: 'center',
  paddingTop: 16,
  borderTopWidth: 1,
  borderTopColor: colors.borderLight,
},
scoreBox: {
  flex: 1,
  alignItems: 'center',
},
scoreValue: {

```

```

    fontSize: 20,
    fontWeight: '700',
    color: colors.primary,
    marginBottom: 2,
  },
  scoreLabel: {
    fontSize: 12,
    color: colors.textSecondary,
  },
  divider: {
    width: 1,
    height: 32,
    backgroundColor: colors.borderLight,
  },
  scroll: {
    flex: 1,
  },
  scrollContent: {
    padding: 8,
    paddingBottom: 32,
  },
  card: {
    backgroundColor: colors.white,
    borderRadius: 8,
    padding: 8,
    marginBottom: 8,
  },
  cardHeader: {
    flexDirection: 'row',
    justifyContent: 'space-between',
    alignItems: 'center',
    marginBottom: 12,
  },
  cardTitle: {
    fontSize: 16,
    fontWeight: '700',
    color: colors.textPrimary,
  },
  cardMore: {
    fontSize: 13,
    color: colors.primary,
  },
  statGrid: {
    flexDirection: 'row',
    flexWrap: 'wrap',
    marginHorizontal: -6,
  },
  statItem: {
    width: '25%',
    paddingHorizontal: 6,
    alignItems: 'center',
  },
  statIcon: {
    width: 44,
    height: 44,
    borderRadius: 12,
    justifyContent: 'center',
    alignItems: 'center',
    marginBottom: 8,
  },
  statValue: {
    fontSize: 17,
    fontWeight: '700',
    color: colors.textPrimary,
    marginBottom: 2,
  },
  statLabel: {
    fontSize: 12,
    color: colors.textSecondary,
    marginBottom: 2,
  },
  statChange: {

```

```

        fontSize: 11,
        fontWeight: '500',
    },
    quickGrid: {
        flexDirection: 'row',
        marginHorizontal: -8,
    },
    quickItem: {
        flex: 1,
        alignItems: 'center',
        paddingVertical: 8,
    },
    quickIcon: {
        width: 48,
        height: 48,
        borderRadius: 14,
        justifyContent: 'center',
        alignItems: 'center',
        marginBottom: 8,
    },
    quickLabel: {
        fontSize: 13,
        color: colors.textPrimary,
    },
    jobRow: {
        flexDirection: 'row',
        alignItems: 'center',
        justifyContent: 'space-between',
        paddingVertical: 12,
        borderBottomWidth: 1,
        borderBottomColor: colors.borderLight,
    },
    jobInfo: {
        flex: 1,
    },
    jobName: {
        fontSize: 15,
        fontWeight: '600',
        color: colors.textPrimary,
        marginBottom: 2,
    },
    jobDepartment: {
        fontSize: 12,
        color: colors.textSecondary,
    },
    jobScoreBox: {
        alignItems: 'flex-end',
    },
    jobScore: {
        fontSize: 18,
        fontWeight: '700',
        color: colors.primary,
    },
    jobChange: {
        fontSize: 11,
        fontWeight: '500',
        marginTop: 2,
    },
    standaloneTitle: {
        fontSize: 16,
        fontWeight: '700',
        color: colors.textPrimary,
        marginBottom: 8,
    },
    manageRow: {
        flexDirection: 'row',
        alignItems: 'center',
        paddingVertical: 12,
        borderBottomWidth: 1,
        borderBottomColor: colors.borderLight,
    },
    manageIcon: {

```

```

        width: 38,
        height: 38,
        borderRadius: 10,
        justifyContent: 'center',
        alignItems: 'center',
        marginRight: 12,
    },
    manageLabel: {
        flex: 1,
        fontSize: 15,
        fontWeight: '500',
        color: colors.textPrimary,
    },
    candidateCard: {
        flexDirection: 'row',
        alignItems: 'center',
        backgroundColor: colors.white,
        borderRadius: 8,
        padding: 8,
        marginBottom: 8,
    },
    candidateIconWrap: {
        width: 44,
        height: 44,
        borderRadius: 12,
        backgroundColor: colors.primaryLight,
        justifyContent: 'center',
        alignItems: 'center',
        marginRight: 12,
    },
    candidateText: {
        flex: 1,
    },
    candidateTitle: {
        fontSize: 15,
        fontWeight: '600',
        color: colors.textPrimary,
        marginBottom: 2,
    },
    candidateDesc: {
        fontSize: 12,
        color: colors.textSecondary,
    },
    candidateAction: {
        fontSize: 13,
        fontWeight: '600',
        color: colors.primary,
        marginLeft: 8,
    },
    },
});
import { Ionicons } from '@expo/vector-icons';
import { useRouter } from 'expo-router';
import { useState } from 'react';
import type { ReactNode } from 'react';
import {
    Alert,
    Modal,
    Pressable,
    ScrollView,
    StyleSheet,
    Text,
    TextInput,
    View,
} from 'react-native';
import { SafeAreaView } from 'react-native-safe-area-context';
import { useJobManagement } from '../../src/hooks';
import { JOB_MANAGEMENT_SORTS, buildJobStatusAction, getJobStatusLabel } from '../../src/api';
import { ErrorState, LoadingState } from '../../src/components/StateView';
import { ENTERPRISE_DISPLAY_NAME } from '../../src/constants';
import { colors } from '../../src/theme';
import { formatCompact, formatDateTime } from '../../src/Utils';
import type { JobManagement, JobManagementJob, JobManagementTab } from '../../src/types';

```

```

/* ===== 子组件: 顶部导航 ===== */
function Header({ companyName, onBack }: { companyName: string; onBack: () => void }) {
  const router = useRouter();
  return (
    <View style={styles.header}>
      <Pressable style={styles.headerBtn} onPress={onBack} hitSlop={8}>
        <Ionicons name="chevron-back" size={24} color={colors.textPrimary} />
      </Pressable>
      <View style={styles.headerCenter}>
        <Text style={styles.headerCompany}>{companyName}</Text>
        <Text style={styles.headerTitle}>岗位管理</Text>
      </View>
      <Pressable
        style={styles.publishBtn}
        onPress={() => router.push('/job-publish')}
        hitSlop={6}
      >
        <Ionicons name="add" size={16} color={colors.white} />
        <Text style={styles.publishBtnText}>发布</Text>
      </Pressable>
    </View>
  );
}

/* ===== 子组件: Tab 栏 ===== */
function TabBar({
  tabs,
  active,
  onChange,
}: {
  tabs: JobManagementTab[];
  active: string;
  onChange: (key: string) => void;
}) {
  return (
    <View style={styles.tabBar}>
      {tabs.map((tab) => {
        const isActive = tab.key === active;
        return (
          <Pressable key={tab.key} style={styles.tabItem} onPress={() => onChange(tab.key)}>
            <Text style={[styles.tabLabel, isActive && styles.tabLabelActive]}>
              {tab.label}
            <Text style={[styles.tabCount, isActive && styles.tabCountActive]}>{tab.count}</Text>
            <View style={[styles.tabIndicator, isActive && styles.tabIndicatorActive]} />
          </Pressable>
        );
      })}
    </View>
  );
}

/* ===== 子组件: 数据概览 ===== */
function StatsOverview({ stats }: { stats: JobManagement['stats'] }) {
  return (
    <View style={styles.statsGrid}>
      {stats.map((stat) => {
        const token = stat.color === 'success'
          ? colors.success
          : stat.color === 'action'
            ? colors.action
            : stat.color === 'info'
              ? colors.info
              : colors.primary;
        const isUp = stat.change.startsWith('+');
        const isDown = stat.change.startsWith('-');
        const changeColor = isUp ? colors.success : isDown ? colors.action : colors.textSecondary;
        return (
          <View key={stat.id} style={styles.statCard}>
            <View style={[styles.statIconWrap, { backgroundColor: token + '15' }]}>
              <Ionicons name={stat.icon as keyof typeof Ionicons.glyphMap} size={22} color={token} />
            </View>
            <View style={styles.statBody}>
              <Text style={styles.statValue}>{stat.value}</Text>
            </View>
          </View>
        );
      })}
    </View>
  );
}

```

```

        <Text style={styles.statLabel}>{stat.label}</Text>
      </View>
      <View style={styles.statChangeWrap}>
        <Text style={[styles.statChange, { color: changeColor }]}>{stat.change}</Text>
      </View>
    </View>
  );
  })}
</View>
);
}
/* ===== 子组件: 搜索与筛选 ===== */
function SearchFilterBar({
  keyword,
  onChangeKeyword,
  sortLabel,
  onOpenSort,
  onOpenFilter,
  activeFilterCount,
}): {
  keyword: string;
  onChangeKeyword: (value: string) => void;
  sortLabel: string;
  onOpenSort: () => void;
  onOpenFilter: () => void;
  activeFilterCount: number;
}) {
  return (
    <View style={styles.searchBar}>
      <View style={styles.searchInputWrap}>
        <Ionicons name="search-outline" size={18} color={colors.textHint} />
        <TextInput
          style={styles.searchInput}
          placeholder="搜索岗位名称..."
          placeholderTextColor={colors.textHint}
          value={keyword}
          onChangeText={onChangeKeyword}
          returnKeyType="search"
        />
        {keyword.length > 0 && (
          <Pressable onPress={() => onChangeKeyword('')} hitSlop={8}>
            <Ionicons name="close-circle" size={16} color={colors.textHint} />
          </Pressable>
        )}
      </View>
      <Pressable style={styles.filterBtn} onPress={onOpenFilter}>
        <Ionicons name="funnel-outline" size={18} color={colors.textPrimary} />
        <Text style={styles.filterBtnText}>筛选</Text>
        {activeFilterCount > 0 && (
          <View style={styles.filterBadge}>
            <Text style={styles.filterBadgeText}>{activeFilterCount}</Text>
          </View>
        )}
      </Pressable>
      <Pressable style={styles.sortBtn} onPress={onOpenSort}>
        <Text style={styles.sortBtnText}>{sortLabel}</Text>
        <Ionicons name="chevron-down" size={14} color={colors.textHint} />
      </Pressable>
    </View>
  );
}
/* ===== 子组件: 底部弹层 ===== */
function Sheet({
  visible,
  title,
  onClose,
  children,
  footer,
}): {
  visible: boolean;
  title: string;
  onClose: () => void;

```

```

children: ReactNode;
footer?: ReactNode;
}) {
  return (
    <Modal visible={visible} transparent animationType="slide" onRequestClose={onClose}>
      <Pressable style={styles.modalOverlay} onPress={onClose}>
        /* affordance-ok: 吞冒泡，子弹层内容区不关闭 */
        <Pressable style={styles.sheetPanel} onPress={() => {}}>
          <View style={styles.sheetHeader}>
            <Text style={styles.sheetTitle}>{title}</Text>
            <Pressable onPress={onClose} hitSlop={8}>
              <Icons name="close" size={22} color={colors.textSecondary} />
            </Pressable>
          </View>
          <ScrollView showsVerticalScrollIndicator={false} style={styles.sheetBody}>
            {children}
          </ScrollView>
          {footer ? <View style={styles.sheetFooter}>{footer}</View> : null}
        </Pressable>
      </Pressable>
    </Modal>
  );
}
/** 弹层里的一组多选筛选（「不限」= 清空该组） */
function FilterGroup({
  title,
  options,
  selected,
  onToggle,
  onClear,
}): {
  title: string;
  options: string[];
  selected: string[];
  onToggle: (value: string) => void;
  onClear: () => void;
}) {
  return (
    <View style={styles.filterGroup}>
      <Text style={styles.filterGroupTitle}>{title}</Text>
      <View style={styles.optionRow}>
        {['不限', ...options].map((option) => {
          const isNone = option === '不限';
          const active = isNone ? selected.length === 0 : selected.includes(option);
          return (
            <Pressable
              key={option}
              style={[styles.optionChip, active && styles.optionChipActive]}
              onPress={() => (isNone ? onClear() : onToggle(option))}
            >
              <Text style={[styles.optionText, active && styles.optionTextActive]}>{option}</Text>
            </Pressable>
          );
        })}
      </View>
    </View>
  );
}
/* ===== 子组件: 岗位卡片 ===== */
/** 状态角标: 只有草稿与已下线才需要标出来，招聘中靠绿点即可。文案取自接口层 getJobStatusLabel */
/** 岗位卡片。编辑入口已摘除 — 编辑态需要给发布页加预填数据，本轮未做，不留点了没反应的按钮 */
function JobCard({
  job,
  refreshing,
  onRefresh,
  onToggleStatus,
}): {
  job: JobManagementJob;
  /** 该岗位是否有写操作进行中 */
  refreshing: boolean;
  onRefresh: (job: JobManagementJob) => void;
  onToggleStatus: (job: JobManagementJob) => void;
}

```

```

    }) {
    const isActive = job.status === 'active';
    /** 状态切换动作（按钮图标/文案、确认框文案、目标状态）一律由接口层给出 */
    const statusAction = buildJobStatusAction(job);
    return (
      <View style={styles.jobCard}>
        <View style={styles.jobHeader}>
          <View style={styles.jobTitleWrap}>
            <Text style={styles.jobTitle}>{job.title}</Text>
            {isActive ? <View style={styles.activeDot} /> : null}
          </View>
          {!isActive && (
            <View style={styles.statusTag}>
              <Text style={styles.statusTagText}>{getJobStatusLabel(job.status)}</Text>
            </View>
          )}
        </View>
        <Text style={styles.jobSalary}>
          {job.salary} · {job.salaryMonths}薪
        </Text>
        <Text style={styles.jobMeta}>{job.city} · {job.experience} · {job.education}</Text>
        <View style={styles.jobScoreRow}>
          <Text style={styles.jobScoreLabel}>综合评分</Text>
          <Text style={styles.jobScoreValue}>{job.overallScore}</Text>
          <View style={styles.rankTag}>
            <Text style={styles.rankTagText}>{job.rankText}</Text>
          </View>
        </View>
        <View style={styles.jobTags}>
          {job.tags.map((tag) => (
            <View key={tag} style={styles.jobTag}>
              <Text style={styles.jobTagText}>{tag}</Text>
            </View>
          ))}
        </View>
        <View style={styles.jobStatsRow}>
          <View style={styles.jobStat}>
            <Text style={styles.jobStatValue}>{job.applicants}</Text>
            <Text style={styles.jobStatLabel}>投递</Text>
          </View>
          <View style={styles.jobStat}>
            <Text style={styles.jobStatValue}>{formatCompact(job.views)}</Text>
            <Text style={styles.jobStatLabel}>查看</Text>
          </View>
          <View style={styles.jobStat}>
            <Text style={styles.jobStatValue}>{job.matches}</Text>
            <Text style={styles.jobStatLabel}>匹配</Text>
          </View>
        </View>
        <View style={styles.jobActions}>
          {isActive && (
            <Pressable
              style={[[styles.jobActionBtn, refreshing && styles.jobActionBtnDisabled]]}
              onPress={() => onRefresh(job)}
              disabled={refreshing}
            >
              <Ionicons name="refresh-outline" size={16} color={colors.primary} />
              <Text style={styles.jobActionText}>
                {job.refreshedAt ? '再次刷新' : '刷新'}
              </Text>
            </Pressable>
          )}
          <Pressable
            style={[[styles.jobActionBtn, refreshing && styles.jobActionBtnDisabled]]}
            onPress={() => onToggleStatus(job)}
            disabled={refreshing}
          >
            /** 图标与文案同出一处（接口层的 buildJobStatusAction），不各写一半 */
            <Ionicons
              name={statusAction.icon as keyof typeof Ionicons.glyphMap}
              size={16}
              color={statusAction.destructive ? colors.action : colors.primary}

```

```

    />
    <Text style={[styles.jobActionText, statusAction.destructive && styles.jobActionTextAction]}}
      {statusAction.label}
    </Text>
  </Pressable>
</View>
<Text style={styles.jobTimeHint}>
  {job.refreshedAt
    ? `最近刷新: ${formatDateTime(job.refreshedAt)}`
    : `发布: ${formatDateTime(job.publishedAt)}`}
</Text>
</View>
);
}
/* ===== 子组件: 空状态 ===== */
function EmptyState({ benefits, onPublish }: { benefits: string[]; onPublish: () => void }) {
  return (
    <View style={styles.emptyCard}>
      <View style={styles.emptyIconWrap}>
        <Icons name="business-outline" size={48} color={colors.primary} />
      </View>
      <Text style={styles.emptyTitle}>还没有发布岗位</Text>
      <Text style={styles.emptySubtitle}>发布你的第一个岗位, 开始招聘优秀人才</Text>
      <Pressable style={styles.emptyBtn} onPress={onPublish} hitSlop={6}>
        <Text style={styles.emptyBtnText}>发布岗位</Text>
      </Pressable>
      <View style={styles.benefitCard}>
        <Text style={styles.benefitTitle}>发布岗位后你可以</Text>
        {benefits.map((benefit, index) => (
          <View key={index} style={styles.benefitRow}>
            <Icons name="checkmark-circle" size={18} color={colors.success} />
            <Text style={styles.benefitText}>{benefit}</Text>
          </View>
        ))}
      </View>
    </View>
  );
}
/* ===== 页面 ===== */
export default function JobManagementScreen() {
  const router = useRouter();
  const {
    management,
    jobs,
    filterOptions,
    activeTab,
    setActiveTab,
    keyword,
    setKeyword,
    sortKey,
    setSortKey,
    filters,
    applyFilters,
    resetFilters,
    activeFilterCount,
    isFiltered,
    setStatus,
    refreshJobItem,
    mutatingId,
    loading,
    error,
    refresh,
  } = useJobManagement();
  // 弹层内的临时选择(点「确定」后才生效), 与 `app/(tabs)/jobs.tsx` 的筛选面板同一套做法
  const [filterVisible, setFilterVisible] = useState(false);
  const [sortVisible, setSortVisible] = useState(false);
  const [draft, setDraft] = useState(filters);
  const sortLabel = JOB_MANAGEMENT_SORTS.find((s) => s.key === sortKey)?.label ?? '最新发布';
  const openFilter = () => {
    setDraft(filters);
    setFilterVisible(true);
  };
};

```

```

/** 切换草稿里的某个筛选值（多选：已在则移除） */
const toggleDraft = (key: 'cities' | 'departments', value: string) => {
  setDraft((prev) => {
    const current = prev[key];
    return {
      ...prev,
      [key]: current.includes(value) ? current.filter((v) => v !== value) : [...current, value],
    };
  });
};

/** 下架 / 上架 / 重新上架。下架是破坏性操作，先二次确认。文案按岗位状态由接口层给出 */
const handleToggleStatus = (job: JobManagementJob) => {
  const action = buildJobStatusAction(job);
  Alert.alert(action.confirmTitle, action.confirmMessage, [
    { text: '取消', style: 'cancel' },
    {
      text: action.confirmButton,
      style: action.destructive ? 'destructive' : 'default',
      onPress: async () => {
        try {
          await setStatus(job.id, action.nextStatus);
        } catch {
          // 写入失败要如实告知，不能让用户以为已经生效
          Alert.alert('操作失败', '岗位状态未能保存，请稍后重试');
        }
      },
    },
  ],
  );
};

const handleRefreshJob = async (job: JobManagementJob) => {
  try {
    await refreshJobItem(job.id);
  } catch {
    Alert.alert('操作失败', '刷新未能保存，请稍后重试');
  }
};

if (loading) {
  return (
    <SafeAreaView style={styles.container} edges={['top']}>
      <Header companyName={ENTERPRISE_DISPLAY_NAME} onBack={() => router.back()} />
      <LoadingState />
    </SafeAreaView>
  );
}

if (error || !management) {
  return (
    <SafeAreaView style={styles.container} edges={['top']}>
      <Header companyName={ENTERPRISE_DISPLAY_NAME} onBack={() => router.back()} />
      <ErrorState onRetry={refresh} />
    </SafeAreaView>
  );
}

return (
  <SafeAreaView style={styles.container} edges={['top', 'bottom']}>
    <Header companyName={management.companyName} onBack={() => router.back()} />
    <TabBar tabs={management.tabs} active={activeTab} onChange={setActiveTab} />
    <ScrollView contentContainerStyle={styles.body} showsVerticalScrollIndicator={false}>
      {management.hasData ? (
        <View>
          <StatsOverview stats={management.stats} />
          <SearchFilterBar
            keyword={keyword}
            onChangeKeyword={setKeyword}
            sortLabel={sortLabel}
            onOpenSort={() => setSortVisible(true)}
            onOpenFilter={openFilter}
            activeFilterCount={activeFilterCount}
          />
          {jobs.length > 0 ? (
            jobs.map((job) => (
              <JobCard
                key={job.id}

```

```

        job={job}
        refreshing={mutatingId === job.id}
        onRefresh={handleRefreshJob}
        onToggleStatus={handleToggleStatus}
      />
    ))
  ) : (
    /* 列表为空有两种原因，说清楚是哪一种，并给出对应的下一步 */
    <View style={styles.emptyResult}>
      <Ionicons name="search-outline" size={40} color={colors.textHint} />
      <Text style={styles.emptyResultText}>
        {isFiltered ? '没有符合条件的岗位' : '这个分类下还没有岗位'}
      </Text>
      {isFiltered && (
        <Pressable
          style={styles.emptyResultBtn}
          onPress={() => {
            setKeyword('');
            resetFilters();
          }}
        >
          <Text style={styles.emptyResultBtnText}>清空筛选条件</Text>
        </Pressable>
      )}
    </View>
  )}
</View>
) : (
  <EmptyState
    benefits={management.benefits}
    onPublish={() => router.push('/job-publish')}
  />
)
</ScrollView>
{/* 排序弹层 */}
<Sheet
  visible={sortVisible}
  title="排序方式"
  onClose={() => setSortVisible(false)}
>
  {JOB_MANAGEMENT_SORTS.map((option) => (
    <Pressable
      key={option.key}
      style={styles.sortOption}
      onPress={() => {
        setSortKey(option.key);
        setSortVisible(false);
      }}
    >
      <Text
        style={[styles.sortOptionText, sortKey === option.key && styles.sortOptionTextActive]}
      >
        {option.label}
      </Text>
      {sortKey === option.key && (
        <Ionicons name="checkmark" size={18} color={colors.primary} />
      )}
    </Pressable>
  ))}
</Sheet>
{/* 筛选弹层。选项来自真实岗位数据，不写死 */}
<Sheet
  visible={filterVisible}
  title="筛选岗位"
  onClose={() => setFilterVisible(false)}
  footer={
    <View style={styles.sheetActions}>
      <Pressable style={styles.resetBtn} onPress={() => setDraft({ cities: [], departments: [] })}>
        <Text style={styles.resetBtnText}>重置</Text>
      </Pressable>
      <Pressable
        style={styles.confirmBtn}
      >

```

```

        onPress={() => {
          applyFilters(draft);
          setFilterVisible(false);
        }}
      >
      <Text style={styles.confirmBtnText}>确定</Text>
    </Pressable>
  </View>
}
>
<FilterGroup
  title="工作城市"
  options={filterOptions.cities}
  selected={draft.cities}
  onToggle={(value) => toggleDraft('cities', value)}
  onClear={() => setDraft((prev) => ({ ...prev, cities: [] })))}
/>
<FilterGroup
  title="所属部门"
  options={filterOptions.departments}
  selected={draft.departments}
  onToggle={(value) => toggleDraft('departments', value)}
  onClear={() => setDraft((prev) => ({ ...prev, departments: [] })))}
/>
</Sheet>
</SafeAreaView>
);
}
const styles = StyleSheet.create({
  container: { flex: 1, backgroundColor: colors.pageBg },
  header: {
    flexDirection: 'row',
    alignItems: 'center',
    justifyContent: 'space-between',
    paddingHorizontal: 12,
    paddingVertical: 12,
    backgroundColor: colors.white,
  },
  headerBtn: { width: 40, alignItems: 'center' },
  headerCenter: { alignItems: 'center' },
  headerCompany: { fontSize: 12, color: colors.textSecondary },
  headerTitle: { fontSize: 17, fontWeight: '700', color: colors.textPrimary, marginTop: 2 },
  publishBtn: {
    flexDirection: 'row',
    alignItems: 'center',
    backgroundColor: colors.primaryBtn,
    paddingHorizontal: 12,
    paddingVertical: 6,
    borderRadius: 16,
  },
  publishBtnText: {
    color: colors.white,
    fontSize: 13,
    fontWeight: '600',
    marginLeft: 2,
  },
  tabBar: {
    flexDirection: 'row',
    backgroundColor: colors.white,
    borderBottomWidth: 1,
    borderBottomColor: colors.borderLight,
  },
  tabItem: {
    flex: 1,
    alignItems: 'center',
    paddingTop: 8,
  },
  tabLabel: {
    fontSize: 14,
    color: colors.textSecondary,
    paddingBottom: 8,
  },
});

```

```

tabLabelActive: {
  color: colors.textPrimary,
  fontWeight: '600',
},
tabCount: {
  fontSize: 12,
  color: colors.textHint,
  marginLeft: 2,
},
tabCountActive: {
  color: colors.primary,
},
tabIndicator: {
  height: 2,
  width: 24,
  borderRadius: 1,
  backgroundColor: 'transparent',
  marginBottom: -1,
},
tabIndicatorActive: {
  backgroundColor: colors.primary,
},
body: {
  padding: 8,
  paddingBottom: 24,
},
center: {
  flex: 1,
  alignItems: 'center',
  justifyContent: 'center',
  gap: 12,
},
tip: { fontSize: 14, color: colors.textSecondary },
retryBtn: {
  marginTop: 8,
  paddingHorizontal: 24,
  paddingVertical: 8,
  borderRadius: 18,
  backgroundColor: colors.primaryBtn,
},
retryText: { color: colors.white, fontSize: 13, fontWeight: '600' },
statsGrid: {
  flexDirection: 'row',
  flexWrap: 'wrap',
  gap: 8,
  marginBottom: 12,
},
statCard: {
  width: '48%',
  flexDirection: 'row',
  alignItems: 'center',
  backgroundColor: colors.white,
  borderRadius: 8,
  padding: 8,
},
statIconWrap: {
  width: 40,
  height: 40,
  borderRadius: 10,
  alignItems: 'center',
  justifyContent: 'center',
  marginRight: 10,
},
statBody: { flex: 1 },
statValue: {
  fontSize: 18,
  fontWeight: '800',
  color: colors.textPrimary,
},
statLabel: {
  fontSize: 12,
  color: colors.textSecondary,

```

```

    marginTop: 2,
  },
  statChangeWrap: {
    alignSelf: 'flex-start',
  },
  statChange: {
    fontSize: 12,
    fontWeight: '600',
  },
  searchBar: {
    flexDirection: 'row',
    alignItems: 'center',
    marginBottom: 12,
    gap: 8,
  },
  searchInputWrap: {
    flex: 1,
    flexDirection: 'row',
    alignItems: 'center',
    backgroundColor: colors.white,
    borderRadius: 8,
    paddingHorizontal: 10,
    paddingVertical: 8,
  },
  searchInput: {
    flex: 1,
    marginLeft: 8,
    fontSize: 14,
    color: colors.textPrimary,
  },
  filterBtn: {
    flexDirection: 'row',
    alignItems: 'center',
    backgroundColor: colors.white,
    borderRadius: 8,
    paddingHorizontal: 10,
    paddingVertical: 8,
  },
  filterBtnText: {
    fontSize: 13,
    color: colors.textPrimary,
    marginLeft: 4,
  },
  sortBtn: {
    flexDirection: 'row',
    alignItems: 'center',
    backgroundColor: colors.white,
    borderRadius: 8,
    paddingHorizontal: 10,
    paddingVertical: 8,
  },
  sortBtnText: {
    fontSize: 13,
    color: colors.textPrimary,
    marginRight: 2,
  },
  jobCard: {
    backgroundColor: colors.white,
    borderRadius: 8,
    padding: 8,
    marginBottom: 8,
  },
  jobHeader: {
    flexDirection: 'row',
    alignItems: 'center',
    justifyContent: 'space-between',
  },
  jobTitleWrap: {
    flexDirection: 'row',
    alignItems: 'center',
  },
  jobTitle: {

```

```

        fontSize: 16,
        fontWeight: '700',
        color: colors.textPrimary,
    },
    activeDot: {
        width: 8,
        height: 8,
        borderRadius: 4,
        backgroundColor: colors.success,
        marginLeft: 8,
    },
    jobSalary: {
        fontSize: 15,
        fontWeight: '700',
        color: colors.action,
        marginTop: 6,
    },
    jobMeta: {
        fontSize: 13,
        color: colors.textSecondary,
        marginTop: 4,
    },
    jobScoreRow: {
        flexDirection: 'row',
        alignItems: 'center',
        marginTop: 10,
    },
    jobScoreLabel: {
        fontSize: 13,
        color: colors.textSecondary,
    },
    jobScoreValue: {
        fontSize: 14,
        fontWeight: '800',
        color: colors.primary,
        marginLeft: 6,
    },
    rankTag: {
        backgroundColor: colors.primaryLight,
        borderRadius: 8,
        paddingHorizontal: 8,
        paddingVertical: 2,
        marginLeft: 8,
    },
    rankTagText: {
        fontSize: 11,
        color: colors.primaryText,
        fontWeight: '600',
    },
    jobTags: {
        flexDirection: 'row',
        flexWrap: 'wrap',
        gap: 8,
        marginTop: 10,
    },
    jobTag: {
        backgroundColor: colors.pageBg,
        borderRadius: 6,
        paddingHorizontal: 8,
        paddingVertical: 3,
    },
    jobTagText: {
        fontSize: 11,
        color: colors.textSecondary,
    },
    jobStatsRow: {
        flexDirection: 'row',
        marginTop: 14,
        padding: 14,
        borderTopWidth: 1,
        borderTopColor: colors.borderLight,
    },

```

```

jobStat: {
  flex: 1,
  alignItems: 'center',
},
jobStatValue: {
  fontSize: 16,
  fontWeight: '700',
  color: colors.textPrimary,
},
jobStatLabel: {
  fontSize: 11,
  color: colors.textSecondary,
  marginTop: 2,
},
jobActions: {
  flexDirection: 'row',
  justifyContent: 'flex-end',
  marginTop: 12,
  gap: 12,
},
jobActionBtn: {
  flexDirection: 'row',
  alignItems: 'center',
},
jobActionBtnDisabled: {
  opacity: 0.4,
},
jobActionText: {
  fontSize: 13,
  color: colors.primary,
  marginLeft: 4,
},
jobActionTextAction: {
  color: colors.action,
},
jobTimeHint: {
  marginTop: 10,
  fontSize: 11,
  color: colors.textHint,
},
statusTag: {
  backgroundColor: colors.pageBg,
  borderRadius: 8,
  paddingHorizontal: 8,
  paddingVertical: 2,
},
statusTagText: {
  fontSize: 11,
  color: colors.textSecondary,
  fontWeight: '600',
},
/* ---- 筛选标记 ---- */
filterBadge: {
  minWidth: 16,
  height: 16,
  borderRadius: 8,
  backgroundColor: colors.primaryBtn,
  alignItems: 'center',
  justifyContent: 'center',
  paddingHorizontal: 4,
  marginLeft: 4,
},
filterBadgeText: {
  fontSize: 10,
  color: colors.white,
  fontWeight: '700',
},
/* ---- 筛选结果为空 ---- */
emptyResult: {
  alignItems: 'center',
  paddingVertical: 48,
  gap: 12,

```

```

},
emptyResultText: {
  fontSize: 14,
  color: colors.textSecondary,
},
emptyResultBtn: {
  marginTop: 4,
  paddingHorizontal: 20,
  paddingVertical: 8,
  borderRadius: 18,
  backgroundColor: colors.primaryLight,
},
emptyResultBtnText: {
  fontSize: 13,
  fontWeight: '600',
  color: colors.primaryText,
},
/* ---- 底部弹层 ---- */
modalOverlay: {
  flex: 1,
  backgroundColor: colors.scrim,
  justifyContent: 'flex-end',
},
sheetPanel: {
  backgroundColor: colors.white,
  borderTopLeftRadius: 16,
  borderTopRightRadius: 16,
  paddingBottom: 24,
  maxHeight: '80%',
},
sheetHeader: {
  flexDirection: 'row',
  alignItems: 'center',
  justifyContent: 'space-between',
  paddingHorizontal: 16,
  paddingVertical: 14,
  borderBottomWidth: 1,
  borderBottomColor: colors.borderLight,
},
sheetTitle: {
  fontSize: 16,
  fontWeight: '700',
  color: colors.textPrimary,
},
sheetBody: {
  paddingHorizontal: 16,
},
sheetFooter: {
  paddingHorizontal: 16,
  paddingTop: 12,
},
sheetActions: {
  flexDirection: 'row',
  gap: 12,
},
resetBtn: {
  flex: 1,
  alignItems: 'center',
  paddingVertical: 12,
  borderRadius: 22,
  borderWidth: 1,
  borderColor: colors.border,
},
resetBtnText: {
  fontSize: 15,
  color: colors.textSecondary,
  fontWeight: '600',
},
confirmBtn: {
  flex: 2,
  alignItems: 'center',
  paddingVertical: 12,

```

```

        borderRadius: 22,
        backgroundColor: colors.primaryBtn,
    },
    confirmBtnText: {
        fontSize: 15,
        color: colors.white,
        fontWeight: '700',
    },
    sortOption: {
        flexDirection: 'row',
        alignItems: 'center',
        justifyContent: 'space-between',
        paddingVertical: 14,
        borderBottomWidth: 1,
        borderBottomColor: colors.borderLight,
    },
    sortOptionText: {
        fontSize: 15,
        color: colors.textPrimary,
    },
    sortOptionTextActive: {
        color: colors.primaryText,
        fontWeight: '700',
    },
    /* ---- 筛选分组 ---- */
    filterGroup: {
        marginTop: 16,
    },
    filterGroupTitle: {
        fontSize: 14,
        fontWeight: '700',
        color: colors.textPrimary,
        marginBottom: 10,
    },
    optionRow: {
        flexDirection: 'row',
        flexWrap: 'wrap',
        gap: 10,
    },
    optionChip: {
        paddingHorizontal: 14,
        paddingVertical: 7,
        borderRadius: 16,
        backgroundColor: colors.pageBg,
        borderWidth: 1,
        borderColor: colors.borderLight,
    },
    optionChipActive: {
        backgroundColor: colors.primaryLight,
        borderColor: colors.primary,
    },
    optionText: {
        fontSize: 13,
        color: colors.textSecondary,
    },
    optionTextActive: {
        color: colors.primaryText,
        fontWeight: '600',
    },
    emptyCard: {
        backgroundColor: colors.white,
        borderRadius: 8,
        padding: 8,
        alignItems: 'center',
    },
    emptyIconWrap: {
        width: 88,
        height: 88,
        borderRadius: 44,
        backgroundColor: colors.primaryLight,
        alignItems: 'center',
        justifyContent: 'center',
    },

```

```

    marginBottom: 16,
  },
  emptyTitle: {
    fontSize: 18,
    fontWeight: '700',
    color: colors.textPrimary,
    marginBottom: 8,
  },
  emptySubtitle: {
    fontSize: 13,
    color: colors.textSecondary,
    textAlign: 'center',
    marginBottom: 20,
  },
  emptyBtn: {
    backgroundColor: colors.primaryBtn,
    paddingHorizontal: 32,
    paddingVertical: 12,
    borderRadius: 24,
    marginBottom: 24,
  },
  emptyBtnText: {
    color: colors.white,
    fontSize: 15,
    fontWeight: '600',
  },
  benefitCard: {
    width: '100%',
    backgroundColor: colors.primaryLight,
    borderRadius: 8,
    padding: 8,
    alignItems: 'flex-start',
  },
  benefitTitle: {
    fontSize: 14,
    fontWeight: '700',
    color: colors.primaryText,
    marginBottom: 12,
  },
  benefitRow: {
    flexDirection: 'row',
    alignItems: 'center',
    marginBottom: 10,
  },
  benefitText: {
    fontSize: 13,
    color: colors.textPrimary,
    marginLeft: 10,
  },
  },
});
import { Icons } from '@expo/vector-icons';
import { useRouter } from 'expo-router';
import { useState } from 'react';
import {
  ActivityIndicator,
  Pressable,
  ScrollView,
  StyleSheet,
  Text,
  TextInput,
  View,
} from 'react-native';
import { SafeAreaView } from 'react-native-safe-area-context';
import { useTalentRecommendation } from '../../src/hooks';
import { ENTERPRISE_DISPLAY_NAME } from '../../src/constants';
import { colors } from '../../src/theme';
import type { TalentCandidate, TalentRecommendationTab } from '../../src/types';
/* ===== 子组件: 顶部导航 ===== */
function Header({
  companyName,
  onBack,
  searching,

```

```

onToggleSearch,
}: {
  companyName: string;
  onBack: () => void;
  /** 是否处于搜索态; 加载中/加载失败时没有数据可搜, 不传即不渲染搜索按钮 */
  searching?: boolean;
  onToggleSearch?: () => void;
}) {
  return (
    <View style={styles.header}>
      <Pressable style={styles.headerBtn} onPress={onBack} hitSlop={8}>
        <Ionicons name="chevron-back" size={24} color={colors.textPrimary} />
      </Pressable>
      <View style={styles.headerCenter}>
        <Text style={styles.headerCompany}>{companyName}</Text>
        <Text style={styles.headerTitle}>人才推荐</Text>
      </View>
      <View style={styles.headerActions}>
        /** 原先这里是「搜索 + 漏斗」两个图标: 漏斗没有对应页面, 先摘掉;
            搜索改成真的关键词过滤 (姓名 / 职位 / 技能 / 学校) */
        {onToggleSearch && (
          <Pressable style={styles.headerActionBtn} onPress={onToggleSearch} hitSlop={6}>
            <Ionicons
              name={searching ? 'close' : 'search-outline'}
              size={22}
              color={colors.textPrimary}
            />
          </Pressable>
        )}
      </View>
    </View>
  );
}
/* ===== 子组件: 搜索栏 ===== */
function SearchBar({
  value,
  onChange,
  onClear,
}): {
  value: string;
  onChange: (text: string) => void;
  onClear: () => void;
}) {
  return (
    <View style={styles.searchBar}>
      <Ionicons name="search-outline" size={18} color={colors.textHint} />
      <TextInput
        style={styles.searchInput}
        value={value}
        onChangeText={onChange}
        placeholder="搜索姓名 / 职位 / 技能"
        placeholderTextColor={colors.textHint}
        autoFocus
        returnKeyType="search"
      />
      {value !== '' && (
        <Pressable onPress={onClear} hitSlop={8}>
          <Ionicons name="close-circle" size={16} color={colors.textHint} />
        </Pressable>
      )}
    </View>
  );
}
/* ===== 子组件: Tab 栏 ===== */
function TabBar({
  tabs,
  active,
  onChange,
}): {
  tabs: TalentRecommendationTab[];
  active: string;
  onChange: (key: string) => void;
}

```

```

    }) {
      return (
        <View style={styles.tabBar}>
          {tabs.map((tab) => {
            const isActive = tab.key === active;
            return (
              <Pressable key={tab.key} style={styles.tabItem} onPress={() => onChange(tab.key)}>
                <Text style={[styles.tabLabel, isActive && styles.tabLabelActive]}>
                  {tab.label}
                <Text style={[styles.tabCount, isActive && styles.tabCountActive]}>{tab.count}</Text>
              </Text>
              <View style={[styles.tabIndicator, isActive && styles.tabIndicatorActive]} />
            </Pressable>
          )};
        </View>
      );
    }
  }
}
/* ===== 子组件: 今日推荐头部 ===== */
/**
 * 今日推荐头部。
 * 原先右上角有「查看人才市场报告」入口，但项目没有该页面 — 按「不留假交互」的口径摘掉。
 */
function TodayHeader({ count }: { count: number }) {
  return (
    <View style={styles.todayHeader}>
      <View>
        <Text style={styles.todayTitle}>今日为你推荐</Text>
        <View style={styles.todayCountRow}>
          <Text style={styles.todayCount}>{count}</Text>
          <Text style={styles.todayUnit}>位优质候选人</Text>
        </View>
      </View>
    </View>
  );
}
/* ===== 子组件: 候选人卡片 ===== */
function CandidateCard({ candidate }: { candidate: TalentCandidate }) {
  const router = useRouter();
  return (
    <View style={styles.candidateCard}>
      <View style={styles.candidateHeader}>
        <View style={styles.avatar}>
          <Text style={styles.avatarText}>{candidate.avatar}</Text>
        </View>
        <View style={styles.candidateInfo}>
          <View style={styles.nameRow}>
            <Text style={styles.candidateName}>{candidate.name}</Text>
            <View style={styles.matchTag}>
              <Text style={styles.matchTagText}>{candidate.matchPercent}%</Text>
            </View>
          </View>
          <Text style={styles.candidateTitle}>{candidate.title}</Text>
          <Text style={styles.candidateMeta}>
            {candidate.age}岁 · {candidate.school}
          </Text>
          <Text style={styles.candidateSalary}>{candidate.expectedSalary}</Text>
        </View>
      </View>
      <View style={styles.scoreRow}>
        <Text style={styles.scoreLabel}>综合评分</Text>
        <Text style={styles.scoreValue}>{candidate.overallScore}</Text>
        <Text style={styles.percentileText}>超越 {candidate.percentile}% 候选人</Text>
      </View>
      <View style={styles.tagRow}>
        {candidate.tags.map((tag) => (
          <View key={tag} style={styles.tag}>
            <Text style={styles.tagText}>{tag}</Text>
          </View>
        ))}
      </View>
      <View style={styles.candidateActions}>

```

```

    <Pressable style={styles.chatBtn} onPress={() => router.navigate('/enterprise-chat')}>
      <Text style={styles.chatBtnText}>发起沟通</Text>
    </Pressable>
    <Pressable style={styles.detailBtn} onPress={() => router.push('/candidate-detail?id=' +
candidate.id)}>
      <Text style={styles.detailBtnText}>查看详情</Text>
    </Pressable>
  </View>
</View>
);
}
/* ===== 子组件: 人才池数据 ===== */
function TalentPool({ stats, actions, onPressAction }: { stats: { id: string; value: string; label: string
}[]; actions: PoolActionItem[]; onPressAction: (action: PoolActionItem) => void })
{
  return (
    <View style={styles.poolCard}>
      <Text style={styles.poolTitle}>人才池数据</Text>
      <View style={styles.poolStatsRow}>
        {stats.map((stat) => (
          <View key={stat.id} style={styles.poolStat}>
            <Text style={styles.poolStatValue}>{stat.value}</Text>
            <Text style={styles.poolStatLabel}>{stat.label}</Text>
          </View>
        ))}
      </View>
      <View style={styles.poolActions}>
        {actions.map((action) => (
          <Pressable key={action.label} style={styles.poolAction} onPress={() => onPressAction(action)}>
            <Ionicons name={action.icon as keyof typeof Ionicons.glyphMap} size={20}
color={colors.textSecondary}
/>
            <Text style={styles.poolActionText}>{action.label}</Text>
          </Pressable>
        ))}
      </View>
    </View>
  );
}
/* ===== 子组件: 空状态 ===== */
function EmptyState({
  actions,
  dataReference,
  onFilter,
}: {
  actions: string[];
  dataReference: { label: string; value: string }[];
  onFilter: () => void;
}) {
  return (
    <View style={styles.emptyCard}>
      <View style={styles.emptyIconWrap}>
        <Ionicons name="people-outline" size={48} color={colors.primary} />
      </View>
      <Text style={styles.emptyTitle}>暂无匹配候选人</Text>
      <Text style={styles.emptySubtitle}>当前筛选条件下没有较适合的候选人，试试调整筛选条件</Text>
      <Pressable style={styles.emptyBtn} onPress={onFilter} hitSlop={6}>
        <Text style={styles.emptyBtnText}>调整筛选条件</Text>
      </Pressable>
      <View style={styles.actionCard}>
        <View style={styles.actionHeader}>
          <Ionicons name="bulb-outline" size={18} color={colors.compareOrange} />
          <Text style={styles.actionTitle}>推荐操作</Text>
        </View>
        {actions.map((action, index) => (
          <View key={index} style={styles.actionRow}>
            <View style={styles.actionDot} />
            <Text style={styles.actionText}>{action}</Text>
          </View>
        ))}
      </View>
    </View>
  );
}

```

```

<Text style={styles.dataTitle}>数据参考</Text>
{dataReference.map((item, index) => (
  <View key={index} style={styles.dataRow}>
    <Ionicons name="trending-up-outline" size={16} color={colors.success} />
    <Text style={styles.dataText}>
      {item.label}
      <Text style={styles.dataValue}> {item.value}</Text>
    </Text>
  </View>
)}}
</View>
</View>
);
}
/* ===== 页面 ===== */
/**
 * 人才池快捷操作项。`route` 必填 — 每个入口都必须有落点，
 * 批量发邀请 / 批量发 Offer / 群发通知三项因项目内没有对应页面，已摘掉。
 */
type PoolActionItem = { icon: string; label: string; route: string };
const POOL_ACTIONS: PoolActionItem[] = [
  { icon: 'calendar-outline', label: '面试评分', route: '/interview-score' },
];
export default function TalentRecommendationScreen() {
  const router = useRouter();
  const {
    recommendation,
    activeTab,
    setActiveTab,
    keyword,
    setKeyword,
    candidates,
    isEmptySearch,
    loading,
    error,
    refresh,
  } = useTalentRecommendation();
  const [searching, setSearching] = useState(false);
  const toggleSearch = () => {
    if (searching) setKeyword('');
    setSearching((prev) => !prev);
  };
  /** 人才池快捷操作：每个入口都由数据保证有落点，页面不再需要「没页面就弹提示」的分支 */
  const handlePoolAction = (action: PoolActionItem) => {
    router.push(action.route);
  };
  /** 空状态里的「调整筛选条件」：把筛选真正复位（切回全部 + 清关键词） */
  const handleFilter = () => {
    setActiveTab('all');
    setKeyword('');
  };
  if (loading) {
    return (
      <SafeAreaView style={styles.container} edges={['top']}>
        <Header companyName={ENTERPRISE_DISPLAY_NAME} onBack={() => router.back()} />
        <View style={styles.center}>
          <ActivityIndicator size="large" color={colors.primary} />
        </View>
      </SafeAreaView>
    );
  }
  if (error || !recommendation) {
    return (
      <SafeAreaView style={styles.container} edges={['top']}>
        <Header companyName={ENTERPRISE_DISPLAY_NAME} onBack={() => router.back()} />
        <View style={styles.center}>
          <Ionicons name="alert-circle-outline" size={40} color={colors.danger} />
          <Text style={styles.tip}>加载失败</Text>
          <Pressable style={styles.retryBtn} onPress={refresh}>
            <Text style={styles.retryText}>重试</Text>
          </Pressable>
        </View>
      </SafeAreaView>
    );
  }
}

```

```

    </SafeAreaView>
  );
}
return (
  <SafeAreaView style={styles.container} edges={['top', 'bottom']}>
    <Header
      companyName={recommendation.companyName}
      onBack={() => router.back()}
      searching={searching}
      onToggleSearch={toggleSearch}
    />
    <TabBar tabs={recommendation.tabs} active={activeTab} onChange={setActiveTab} />
    {searching && (
      <SearchBar value={keyword} onChange={setKeyword} onClear={() => setKeyword('')} />
    )}
    <ScrollView contentContainerStyle={styles.body} showsVerticalScrollIndicator={false}>
      {recommendation.hasData ? (
        <View>
          <TodayHeader count={recommendation.todayCount} />
          {candidates.length === 0 ? (
            <View style={styles.emptyList}>
              <Ionicons name="search-outline" size={32} color={colors.textHint} />
              <Text style={styles.emptyListText}>
                {isEmptySearch
                  ? `没有匹配「${keyword.trim()}」的候选人`
                  : `这个筛选条件下暂时没有候选人`}
              </Text>
              <Pressable style={styles.emptyListBtn} onPress={handleFilter} hitSlop={6}>
                <Text style={styles.emptyListBtnText}>重置筛选</Text>
              </Pressable>
            </View>
          ) : (
            candidates.map((candidate) => (
              <CandidateCard key={candidate.id} candidate={candidate} />
            ))
          )}
          <TalentPool
            stats={recommendation.poolStats}
            actions={POOL_ACTIONS}
            onPressAction={handlePoolAction}
          />
        </View>
      ) : (
        <EmptyState
          actions={recommendation.recommendedActions}
          dataReference={recommendation.dataReference}
          onFilter={handleFilter}
        />
      )}
    </ScrollView>
  </SafeAreaView>
);
}
const styles = StyleSheet.create({
  container: { flex: 1, backgroundColor: colors.pageBg },
  header: {
    flexDirection: 'row',
    alignItems: 'center',
    justifyContent: 'space-between',
    paddingHorizontal: 12,
    paddingVertical: 12,
    backgroundColor: colors.white,
  },
  headerBtn: { width: 40, alignItems: 'center' },
  headerCenter: { alignItems: 'center' },
  headerCompany: { fontSize: 12, color: colors.textSecondary },
  headerTitle: { fontSize: 17, fontWeight: '700', color: colors.textPrimary, marginTop: 2 },
  headerActions: {
    flexDirection: 'row',
    alignItems: 'center',
  },
  headerActionBtn: {

```

```

        width: 36,
        alignItems: 'center',
      },
      searchBar: {
        flexDirection: 'row',
        alignItems: 'center',
        gap: 8,
        backgroundColor: colors.white,
        paddingHorizontal: 16,
        paddingBottom: 10,
      },
      searchInput: {
        flex: 1,
        backgroundColor: colors.pageBg,
        borderRadius: 10,
        paddingHorizontal: 12,
        paddingVertical: 8,
        fontSize: 14,
        color: colors.textPrimary,
      },
      emptyList: {
        alignItems: 'center',
        gap: 10,
        padding: 8,
        backgroundColor: colors.white,
        borderRadius: 8,
      },
      emptyListText: { fontSize: 13, color: colors.textSecondary },
      emptyListBtn: {
        paddingHorizontal: 18,
        paddingVertical: 8,
        borderRadius: 16,
        backgroundColor: colors.primaryLight,
      },
      emptyListBtnText: { fontSize: 13, fontWeight: '600', color: colors.primaryText },
      tabBar: {
        flexDirection: 'row',
        backgroundColor: colors.white,
        borderBottomWidth: 1,
        borderBottomColor: colors.borderLight,
      },
      tabItem: {
        flex: 1,
        alignItems: 'center',
        paddingTop: 8,
      },
      tabLabel: {
        fontSize: 14,
        color: colors.textSecondary,
        paddingBottom: 8,
      },
      tabLabelActive: {
        color: colors.textPrimary,
        fontWeight: '600',
      },
      tabCount: {
        fontSize: 12,
        color: colors.textHint,
        marginLeft: 2,
      },
      tabCountActive: {
        color: colors.primary,
      },
      tabIndicator: {
        height: 2,
        width: 24,
        borderRadius: 1,
        backgroundColor: 'transparent',
        marginBottom: -1,
      },
      tabIndicatorActive: {
        backgroundColor: colors.primary,
      },

```

```

},
body: {
  padding: 8,
  paddingBottom: 24,
},
center: {
  flex: 1,
  alignItems: 'center',
  justifyContent: 'center',
  gap: 12,
},
tip: { fontSize: 14, color: colors.textSecondary },
retryBtn: {
  marginTop: 8,
  paddingHorizontal: 24,
  paddingVertical: 8,
  borderRadius: 18,
  backgroundColor: colors.primaryBtn,
},
retryText: { color: colors.white, fontSize: 13, fontWeight: '600' },
todayHeader: {
  flexDirection: 'row',
  alignItems: 'center',
  justifyContent: 'space-between',
  marginBottom: 12,
},
todayTitle: {
  fontSize: 13,
  color: colors.textSecondary,
},
todayCountRow: {
  flexDirection: 'row',
  alignItems: 'baseline',
  marginTop: 2,
},
todayCount: {
  fontSize: 28,
  fontWeight: '800',
  color: colors.primary,
},
todayUnit: {
  fontSize: 13,
  color: colors.textSecondary,
  marginLeft: 4,
},
reportBtn: {
  flexDirection: 'row',
  alignItems: 'center',
  backgroundColor: colors.primaryLight,
  borderRadius: 14,
  paddingHorizontal: 10,
  paddingVertical: 6,
},
reportBtnText: {
  fontSize: 12,
  color: colors.primary,
  fontWeight: '600',
  marginRight: 2,
},
candidateCard: {
  backgroundColor: colors.white,
  borderRadius: 8,
  padding: 8,
  marginBottom: 8,
},
candidateHeader: {
  flexDirection: 'row',
},
avatar: {
  width: 52,
  height: 52,
  borderRadius: 26,

```

```

        backgroundColor: colors.primaryLight,
        alignItems: 'center',
        justifyContent: 'center',
        marginRight: 12,
      },
      avatarText: {
        fontSize: 20,
        fontWeight: '700',
        color: colors.primary,
      },
      candidateInfo: {
        flex: 1,
      },
      nameRow: {
        flexDirection: 'row',
        alignItems: 'center',
        justifyContent: 'space-between',
      },
      candidateName: {
        fontSize: 16,
        fontWeight: '700',
        color: colors.textPrimary,
      },
      matchTag: {
        backgroundColor: colors.successLight,
        borderRadius: 10,
        paddingHorizontal: 8,
        paddingVertical: 2,
      },
      matchTagText: {
        fontSize: 12,
        fontWeight: '800',
        color: colors.successText,
      },
      candidateTitle: {
        fontSize: 13,
        color: colors.textSecondary,
        marginTop: 2,
      },
      candidateMeta: {
        fontSize: 12,
        color: colors.textHint,
        marginTop: 4,
      },
      candidateSalary: {
        fontSize: 13,
        color: colors.action,
        fontWeight: '600',
        marginTop: 4,
      },
      scoreRow: {
        flexDirection: 'row',
        alignItems: 'center',
        marginTop: 12,
      },
      scoreLabel: {
        fontSize: 13,
        color: colors.textSecondary,
      },
      scoreValue: {
        fontSize: 14,
        fontWeight: '800',
        color: colors.primary,
        marginLeft: 6,
      },
      percentileText: {
        fontSize: 12,
        color: colors.textHint,
        marginLeft: 8,
      },
      tagRow: {
        flexDirection: 'row',

```

```

    flexWrap: 'wrap',
    gap: 8,
    marginTop: 10,
  },
  tag: {
    backgroundColor: colors.pageBg,
    borderRadius: 6,
    paddingHorizontal: 8,
    paddingVertical: 3,
  },
  tagText: {
    fontSize: 11,
    color: colors.textSecondary,
  },
  candidateActions: {
    flexDirection: 'row',
    marginTop: 14,
    gap: 10,
  },
  chatBtn: {
    flex: 1,
    backgroundColor: colors.primaryBtn,
    borderRadius: 18,
    paddingVertical: 10,
    alignItems: 'center',
  },
  chatBtnText: {
    color: colors.white,
    fontSize: 14,
    fontWeight: '600',
  },
  detailBtn: {
    flex: 1,
    backgroundColor: colors.primaryLight,
    borderRadius: 18,
    paddingVertical: 10,
    alignItems: 'center',
  },
  detailBtnText: {
    color: colors.primary,
    fontSize: 14,
    fontWeight: '600',
  },
  poolCard: {
    backgroundColor: colors.white,
    borderRadius: 8,
    padding: 8,
    marginTop: 8,
  },
  poolTitle: {
    fontSize: 15,
    fontWeight: '700',
    color: colors.textPrimary,
    marginBottom: 12,
  },
  poolStatsRow: {
    flexDirection: 'row',
    justifyContent: 'space-between',
    marginBottom: 16,
  },
  poolStat: {
    flex: 1,
    alignItems: 'center',
  },
  poolStatValue: {
    fontSize: 18,
    fontWeight: '800',
    color: colors.textPrimary,
  },
  poolStatLabel: {
    fontSize: 11,
    color: colors.textSecondary,

```

```

    marginTop: 4,
  },
  poolActions: {
    flexDirection: 'row',
    justifyContent: 'space-around',
    padding: 14,
    borderTopWidth: 1,
    borderTopColor: colors.borderLight,
  },
  poolAction: {
    alignItems: 'center',
  },
  poolActionText: {
    fontSize: 11,
    color: colors.textSecondary,
    marginTop: 4,
  },
  emptyCard: {
    backgroundColor: colors.white,
    borderRadius: 8,
    padding: 8,
    alignItems: 'center',
  },
  emptyIconWrap: {
    width: 88,
    height: 88,
    borderRadius: 44,
    backgroundColor: colors.primaryLight,
    alignItems: 'center',
    justifyContent: 'center',
    marginBottom: 16,
  },
  emptyTitle: {
    fontSize: 18,
    fontWeight: '700',
    color: colors.textPrimary,
    marginBottom: 8,
  },
  emptySubtitle: {
    fontSize: 13,
    color: colors.textSecondary,
    textAlign: 'center',
    marginBottom: 20,
  },
  emptyBtn: {
    backgroundColor: colors.primaryBtn,
    padding: 32, 12,
    borderRadius: 24,
    marginBottom: 24,
  },
  emptyBtnText: {
    color: colors.white,
    fontSize: 15,
    fontWeight: '600',
  },
  actionCard: {
    width: '100%',
    backgroundColor: colors.actionLight,
    borderRadius: 8,
    padding: 8,
    marginBottom: 8,
    alignItems: 'flex-start',
  },
  actionHeader: {
    flexDirection: 'row',
    alignItems: 'center',
    marginBottom: 12,
  },
  actionTitle: {
    fontSize: 14,
    fontWeight: '700',
  },

```

```

        + " (SELECT COUNT(DISTINCT CASE WHEN duplicate_history=1 THEN candidate_user_id END) FROM
candidate_flags) AS duplicate_history_candidates,"
        + " (SELECT COUNT(DISTINCT CASE WHEN invalid_score=1 THEN candidate_user_id END) FROM
candidate_flags) AS invalid_score_candidates,"
        + " (SELECT COUNT(DISTINCT CASE WHEN version_changed=1 THEN candidate_user_id END) FROM
candidate_flags) AS version_changed_candidates,"
        + " (SELECT COUNT(DISTINCT CASE WHEN changed=1 THEN candidate_user_id END) FROM
candidate_flags) AS changed_candidates,"
        + " (SELECT COUNT(DISTINCT CASE WHEN unchanged=1 THEN candidate_user_id END) FROM
candidate_flags) AS unchanged_candidates,"
        + " (SELECT COUNT(DISTINCT CASE WHEN changed=1 AND unchanged=1 THEN candidate_user_id END)
FROM candidate_flags) AS changed_unchanged_intersection,"
        + " ra.comparable_rows, ra.comparable_candidates, ra.changed_rows, ra.changed_candidates,
ra.delta_sum, ra.abs_delta_sum"
        + " FROM rollback_aggregate ra";
    List<AppealAggregateSummary> result = queryWithTimeout(sql, (rs, rowNum) -> new
AppealAggregateSummary(
        rs.getInt("appeal_count"), rs.getInt("sample_size"),
        statusMap(rs.getInt("pending_events"), rs.getInt("approved_events"),
rs.getInt("rejected_events"),
            rs.getInt("cancelled_events"), rs.getInt("rolled_back_events"),
rs.getInt("unknown_events")),
        statusMap(rs.getInt("pending_candidates"), rs.getInt("approved_candidates"),
rs.getInt("rejected_candidates"),
            rs.getInt("cancelled_candidates"), rs.getInt("rolled_back_candidates"),
rs.getInt("unknown_candidates")),
        rs.getInt("approved_ever_candidates"), rs.getInt("approval_denominator"),
        rs.getInt("rolled_back_candidates"), rs.getInt("duplicate_appeal_candidates"),
        rs.getInt("comparable_rows"), rs.getInt("comparable_candidates"), rs.getInt("changed_rows"),
        rs.getInt("changed_candidates"), rs.getInt("unchanged_candidates"),
        rs.getInt("changed_unchanged_intersection"), rs.getBigDecimal("delta_sum"),
rs.getBigDecimal("abs_delta_sum"),
        rs.getInt("duplicate_history_candidates"), rs.getInt("invalid_score_candidates"),
rs.getInt("version_changed_candidates")), from,
to);
    return result.isEmpty() ? AppealAggregateSummary.empty() : result.get(0);
}
private static Map<String, Integer> statusMap(int pending, int approved, int rejected, int cancelled,
int rolledBack, int unknown)
{
    Map<String, Integer> map = new LinkedHashMap<>();
    map.put("PENDING", pending); map.put("APPROVED", approved); map.put("REJECTED", rejected);
    map.put("CANCELLED", cancelled); map.put("ROLLED_BACK", rolledBack); map.put("UNKNOWN", unknown);
    return map;
}
/** 维度/状态/规则/分数的数据库频数聚合; candidate_count 为 distinct candidate. */
public List<SnapshotAggregateRow> findSnapshotAggregates(String period) {
    return queryWithTimeout("SELECT dimension, status, rules_version, score, " +
        "COUNT(DISTINCT student_user_id) AS candidate_count " +
        "FROM candidate_dimension_scores WHERE period = ? " +
        "GROUP BY dimension, status, rules_version, score",
        (rs, rowNum) -> new SnapshotAggregateRow(rs.getString("dimension"), rs.getString("status"),
            rs.getString("rules_version"), rs.getBigDecimal("score"),
rs.getInt("candidate_count")),
period);
}
public int findSnapshotPopulation(String period) {
    List<Integer> result = queryWithTimeout(
        "SELECT COUNT(DISTINCT student_user_id) AS candidate_count " +
        "FROM candidate_dimension_scores WHERE period = ?",
        (rs, rowNum) -> rs.getInt("candidate_count"), period);
    return result.isEmpty() ? 0 : result.get(0);
}
private <T> List<T> queryWithTimeout(String sql, RowMapper<T> mapper, Object... args) {
    return jdbcTemplate.query(connection -> {
        PreparedStatement statement = connection.prepareStatement(sql);
        statement.setQueryTimeout(5);
        for (int i = 0; i < args.length; i++) statement.setObject(i + 1, args[i]);
        return statement;
    }, mapper);
}
private static BigDecimal scaleScore(BigDecimal score) {

```

```

        return score == null ? null : score.setScale(2, RoundingMode.HALF_UP);
    }
    record SnapshotAggregateRow(String dimension, String status, String rulesVersion, BigDecimal score, int
candidateCount)
    {}
    record OverviewAggregateData(int populationSize, int computableCount, Map<String, Integer> versionCounts,
Map<String, List<FairnessStatisticsCalculator.ScoreFrequency>>
frequenciesByVersion)
    {
        static OverviewAggregateData empty() {
            Map<String, Integer> versions = new LinkedHashMap<>();
            versions.put("candidate-scoring-v0.1", 0); versions.put("candidate-scoring-v0.2", 0);
            versions.put("UNKNOWN", 0); versions.put("MIXED", 0);
            Map<String, List<FairnessStatisticsCalculator.ScoreFrequency>> frequencies = new
LinkedHashMap<>();
            frequencies.put("candidate-scoring-v0.1", List.of()); frequencies.put("candidate-scoring-v0.2",
List.of());
            return new OverviewAggregateData(0, 0, versions, frequencies);
        }
    }
    record AppealAggregateSummary(int appealCount, int sampleSize, Map<String, Integer> statusEventCounts,
Map<String, Integer> statusCandidateCounts, int approvedEverCount,
int approvalDenominatorCount, int rolledBackCandidateCount,
int duplicateAppealCandidateCount, int comparableRollbackCount,
int comparableRollbackCandidateCount, int changedCount, int
changedCandidateCount,
int unchangedCandidateCount, int changedUnchangedIntersectionCount,
BigDecimal deltaSum, BigDecimal absDeltaSum,
int duplicateRollbackHistoryCount, int invalidScoreRollbackCount,
int ruleVersionChangedRollbackCount) {
        static AppealAggregateSummary empty() {
            return new AppealAggregateSummary(0, 0, statusMap(0, 0, 0, 0, 0, 0), statusMap(0, 0, 0, 0, 0, 0),
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, BigDecimal.ZERO, BigDecimal.ZERO, 0, 0, 0);
        }
    }
    private record OverviewSummaryRow(int populationSize, int computableCount, int mixedCount, int
unknownCount,
int v01Count, int v02Count) {}
    private record OverviewFrequencyRow(String version, BigDecimal score, int candidateCount) {}
}
package com.mydreamjob.scoringmonitoring;
import com.mydreamjob.auth.User;
import com.mydreamjob.auth.UserRepository;
import com.mydreamjob.common.exception.BusinessException;
import com.mydreamjob.common.exception.ErrorCode;
import lombok.RequiredArgsConstructor;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;
import java.math.BigDecimal;
import java.math.RoundingMode;
import java.time.Instant;
import java.time.LocalDate;
import java.time.LocalDateTime;
import java.time.ZoneId;
import java.time.format.DateTimeFormatter;
import java.time.format.DateTimeParseException;
import java.time.temporal.ChronoUnit;
import java.util.ArrayList;
import java.util.LinkedHashMap;
import java.util.List;
import java.util.Map;
import java.util.Set;
@Service
@RequiredArgsConstructor
public class FairnessMonitoringService {
    private static final ZoneId ZONE = ZoneId.of("Asia/Shanghai");
    private static final DateTimeFormatter PERIOD = DateTimeFormatter.ofPattern("yyyy-MM");
    private static final Set<String> DIMENSIONS = Set.of("EDUCATION", "INTERNSHIP", "ABILITY", "COMPLIANCE",
"STABILITY", "GROWTH", "INTEGRITY", "ACTIVITY");
    private static final Set<String> KNOWN_RULES = Set.of("candidate-scoring-v0.1",
"candidate-scoring-v0.2");
    private static final Set<String> KNOWN_STATUSES = Set.of("OK", "ESTIMATED", "PENDING",

```

```

"INSUFFICIENT_DATA");
    private final FairnessMonitoringRepository repository;
    private final UserRepository userRepository;
    public void assertAdmin(Long userId) {
        User user = userRepository.findById(userId).orElseThrow(() -> new
BusinessException(ErrorCode.USER_NOT_FOUND));
        if (user.getRoles() == null || user.getRoles().stream().noneMatch(r -> r == User.Role.ADMIN)) {
            throw new BusinessException(ErrorCode.FORBIDDEN);
        }
    }
    @Transactional(readOnly = true)
    public FairnessResponse.Overview overview(Long adminUserId, String period) {
        assertAdmin(adminUserId);
        Instant captured = Instant.now();
        String validPeriod = validatePeriod(period, captured);
        FairnessMonitoringRepository.OverviewAggregateData aggregate =
repository.findOverviewAggregateData(validPeriod);
        if (aggregate == null) aggregate = FairnessMonitoringRepository.OverviewAggregateData.empty();
        return overviewFromAggregate(validPeriod, captured, aggregate);
    }
    private FairnessResponse.Overview overviewFromAggregate(String period, Instant captured,
FairnessMonitoringRepository.OverviewAggregateD
ata aggregate)
    {
        int population = aggregate.populationSize();
        int computable = aggregate.computableCount();
        Map<String, Integer> versions = new LinkedHashMap<>(aggregate.versionCounts());
        Map<String, FairnessResponse.Distribution> byVersion = new LinkedHashMap<>();
        Map<String, Boolean> suppressedVersions = new LinkedHashMap<>();
        aggregate.frequenciesByVersion().forEach((version, frequencies) -> {
            int versionPopulation = versions.getOrDefault(version, 0);
            int validSample = frequencyCount(frequencies);
            boolean subsetSuppressed = versionPopulation > 0
                && FairnessDisclosureGuard.suppressRatio(versionPopulation, validSample);
            suppressedVersions.put(version, subsetSuppressed);
            FairnessResponse.Distribution distribution = FairnessStatisticsCalculator.distributionFromFrequen
cies(frequencies);
            if (subsetSuppressed) {
                byVersion.put(version, suppressedDistribution());
            } else if (distribution.getAvailability() != null && !"NO_DATA".equals(distribution.getAvailabili
ty()))
            {
                byVersion.put(version, distribution);
            }
        });
        boolean ratioSuppressed = FairnessDisclosureGuard.suppressRatio(population, computable);
        boolean versionSuppressed = FairnessDisclosureGuard.suppressCells(population, versions.values(),
true);
        // Each version is an independent disclosure universe. An unsafe
        // version is suppressed as a cell, while safe sibling versions may
        // still be shown when the global universe itself is safe.
        boolean versionSubsetSuppressed = suppressedVersions.values().stream().anyMatch(Boolean.TRUE::equals)
;
        boolean distributionSuppressed = byVersion.values().stream()
            .anyMatch(d -> "SUPPRESSED".equals(d.getAvailability()));
        boolean derivedSuppressed = ratioSuppressed || versionSuppressed || versionSubsetSuppressed;
        FairnessResponse.Distribution single = null;
        if (!ratioSuppressed && !versionSuppressed && !versionSubsetSuppressed && byVersion.size() == 1) {
            single = byVersion.values().iterator().next();
        }
        boolean insufficient = population > 0 && population < FairnessDisclosureGuard.MIN_INDEPENDENT_CANDIDA
TES;
        String availability = population == 0 ? "NO_DATA" : insufficient ? "INSUFFICIENT_SAMPLE"
            : (derivedSuppressed || distributionSuppressed ? "SUPPRESSED" : "AVAILABLE");
        String reason = population == 0 ? "NO_SNAPSHOT_POPULATION" : insufficient ?
"INDEPENDENT_CANDIDATE_THRESHOLD_NOT_MET"
            : (derivedSuppressed ? "SMALL_CELL_OR_COMPLEMENT_RISK"
                : distributionSuppressed ? "SMALL_BUCKET_OR_COMPLEMENT_RISK" : null);
        boolean anyVersion = aggregate.frequenciesByVersion().values().stream().anyMatch(v -> v != null &&
!v.isEmpty());
        if (!anyVersion && population >= FairnessDisclosureGuard.MIN_INDEPENDENT_CANDIDATES) reason =
"NO_COHERENT_VERSION_SCORE";
    }

```

```

        return FairnessResponse.Overview.builder().basis("CURRENT_STORED_SNAPSHOTS").scope("SCORED_SNAPSHOT_POPULATION")
            .period(period).capturedAt(captured).populationSize(population)
            .computableSnapshotRatio(ratioSuppressed ? null : FairnessStatisticsCalculator.ratio(computable, population))
            .confidence(FairnessStatisticsCalculator.confidence(population))
            .riskLevel(insufficient ? "INSUFFICIENT_SAMPLE" : derivedSuppressed ? null : highestRisk(byVersion.values()))
            // A protected version count plus a safe sibling count can
            // reconstruct the protected total; hide the version-count
            // group whenever any version cell is unsafe.
            .versionCounts(ratioSuppressed || versionSuppressed || versionSubsetSuppressed ? null : versions)
            .mixedVersionCount(ratioSuppressed || versionSuppressed || versionSubsetSuppressed ? null : versions.getDefault("MIXED", 0))
            .unknownVersionCount(ratioSuppressed || versionSuppressed || versionSubsetSuppressed ? null : versions.getDefault("UNKNOWN", 0))
            .distribution(ratioSuppressed || versionSuppressed ? null : single)
            // When only a version-local complement is unsafe, preserve
            // safe sibling distributions but keep the unsafe cell
            // explicitly SUPPRESSED with no numeric statistics.
            .distributionsByRulesVersion(ratioSuppressed || versionSuppressed ? null : byVersion)
            .coherentVersionSubset(ratioSuppressed || versionSuppressed ? null : new ArrayList<>(byVersion.keySet()))
            .availability(availability).reason(reason).build();
    }
    @Transactional(readOnly = true)
    public FairnessResponse.Dimensions dimensions(Long adminUserId, String period, String requestedDimension)
    {
        assertAdmin(adminUserId);
        Instant captured = Instant.now();
        String validPeriod = validatePeriod(period, captured);
        if (requestedDimension != null && !DIMENSIONS.contains(requestedDimension)) {
            throw new BusinessException(ErrorCodes.BAD_REQUEST, "invalid dimension");
        }
        List<FairnessMonitoringRepository.SnapshotAggregateRow> aggregates = repository.findSnapshotAggregates(validPeriod);
        int population = repository.findSnapshotPopulation(validPeriod);
        List<String> targets = requestedDimension == null ? DIMENSIONS.stream().sorted().toList() : List.of(requestedDimension);
        List<FairnessResponse.DimensionMetric> result = new ArrayList<>();
        for (String dimension : targets) {
            List<FairnessMonitoringRepository.SnapshotAggregateRow> items = aggregates.stream().filter(r -> dimension.equals(r.dimension())).toList();
            int rowCount = sum(items);
            int missing = Math.max(0, population - rowCount);
            int nullScore = sum(items.stream().filter(r -> r.score() == null).toList());
            int valid = sum(items.stream().filter(this::isValidAggregate).toList());
            Map<String, Integer> statuses = groupCounts(items, true);
            Map<String, Integer> versions = groupCounts(items, false);
            Map<String, List<FairnessStatisticsCalculator.ScoreFrequency>> frequencies = new LinkedHashMap<>();
            for (var item : items) if (isValidAggregate(item)) {
                frequencies.computeIfAbsent(normaliseVersion(item.rulesVersion()), ignored -> new ArrayList<>())
                    .add(new FairnessStatisticsCalculator.ScoreFrequency(item.score(), item.candidateCount()));
            }
            boolean bucketSuppressed = frequencies.values().stream().map(FairnessStatisticsCalculator::distributionFromFrequencies).anyMatch(d -> "SUPPRESSED".equals(d.getAvailability()));
            Map<String, FairnessResponse.Distribution> byVersion = new LinkedHashMap<>();
            Map<String, Boolean> suppressedVersions = new LinkedHashMap<>();
            frequencies.forEach((v, f) -> {
                int versionPopulation = versions.getDefault(v, 0);
                boolean subsetSuppressed = versionPopulation > 0
                    && FairnessDisclosureGuard.suppressRatio(versionPopulation, frequencyCount(f));
                suppressedVersions.put(v, subsetSuppressed);
                byVersion.put(v, subsetSuppressed);
            });
        }
    }

```

```

        ? suppressedDistribution()
        : FairnessStatisticsCalculator.distributionFromFrequencies(f));
    });
    FairnessResponse.Distribution distribution = FairnessStatisticsCalculator.distributionFromFrequencies(
        frequencies.values().stream().flatMap(List::stream).toList());
    int inconsistent = sum(items.stream().filter(r -> ("OK".equals(r.status()) ||
"ESTIMATED".equals(r.status())) != (r.score() !=
null)).toList());
    int unknownStatus = sum(items.stream().filter(r -> !KNOWN_STATUSES.contains(r.status())).toList()
    );
    int outOfRange = sum(items.stream().filter(r -> r.score() != null &&
!validScore(r.score())).toList());
    int unknownRules = sum(items.stream().filter(r -> !KNOWN_RULES.contains(r.rulesVersion())).toList
    ());
    boolean versionSubsetSuppressed = suppressedVersions.values().stream().anyMatch(Boolean.TRUE::equ
als);
    boolean baseSuppress = FairnessDisclosureGuard.suppressCells(population,
        List.of(rowCount, missing, nullScore, valid, inconsistent, unknownStatus, outOfRange,
unknownRules),
        true)
        || FairnessDisclosureGuard.suppressCells(population, statuses.values(), true)
        || FairnessDisclosureGuard.suppressCells(population, versions.values(), true)
        || "SUPPRESSED".equals(distribution.getAvailability())
        || bucketSuppressed;
    // A version-local unsafe complement suppresses aggregate fields
    // that could be combined with a safe sibling to recover it, but
    // does not erase a safe sibling distribution.
    boolean suppress = baseSuppress || versionSubsetSuppressed;
    boolean insufficient = population > 0 && (population <
FairnessDisclosureGuard.MIN_INDEPENDENT_CANDIDATES
        || (valid > 0 && valid < FairnessDisclosureGuard.MIN_INDEPENDENT_CANDIDATES));
    String availability = population == 0 ? "NO_DATA" : insufficient ? "INSUFFICIENT_SAMPLE"
        : suppress ? "SUPPRESSED" : valid == 0 ? "NO_DATA" : "AVAILABLE";
    String reason = population == 0 ? "NO_SNAPSHOT_POPULATION" : insufficient ?
"INDEPENDENT_CANDIDATE_THRESHOLD_NOT_MET"
        : suppress ? "SMALL_CELL_OR_COMPLEMENT_RISK" : valid == 0 ? "NO_VALID_SCORE" : null;
    result.add(FairnessResponse.DimensionMetric.builder().dimension(dimension).populationSize(populat
ion)
        .rowCount(suppress ? null : rowCount).missingRowCount(suppress ? null : missing)
        .validSampleSize(suppress ? null : valid).confidence(suppress ? null :
FairnessStatisticsCalculator.confidence(valid))
        .missingRowRate(suppress ? null : FairnessStatisticsCalculator.ratio(missing,
population))
        .nullScoreRate(suppress ? null : FairnessStatisticsCalculator.ratio(nullScore,
population))
        .validCoverage(suppress ? null : FairnessStatisticsCalculator.ratio(valid, population))
        .statusCounts(suppress ? null : statuses).versionCounts(suppress ? null : versions)
        .inconsistentRecordCount(suppress ? null : inconsistent).unknownStatusCount(suppress ?
null : unknownStatus)
        .outOfRangeScoreCount(suppress ? null : outOfRange).unknownRulesVersionCount(suppress ?
null : unknownRules)
        .distribution(baseSuppress || versionSubsetSuppressed ? null : distribution)
        .distributionsByRulesVersion(baseSuppress ? null : byVersion)
        .availability(availability).reason(reason).build());
    }
    boolean allNoData = result.stream().allMatch(r -> "NO_DATA".equals(r.getAvailability()));
    boolean anyInsufficient = result.stream().anyMatch(r -> "INSUFFICIENT_SAMPLE".equals(r.getAvailabilit
y()));
    boolean anySuppressed = result.stream().anyMatch(r -> "SUPPRESSED".equals(r.getAvailability()));
    return FairnessResponse.Dimensions.builder().basis("CURRENT_STORED_SNAPSHOTS").scope("SCORED_SNAPSHOT
_POPULATION")
        .period(validPeriod).capturedAt(captured).dimensions(result)
        .availability(allNoData ? "NO_DATA" : anySuppressed ? "SUPPRESSED" : anyInsufficient ?
"INSUFFICIENT_SAMPLE" :
"AVAILABLE")
        .reason(allNoData ? "NO_SNAPSHOT_POPULATION" : anySuppressed ?
"SMALL_CELL_OR_COMPLEMENT_RISK"
        : anyInsufficient ? "INDEPENDENT_CANDIDATE_THRESHOLD_NOT_MET" : null).build();
    }
    @Transactional(readOnly = true)
    public FairnessResponse.Unavailable drift(Long adminUserId) {

```

```

        assertAdmin(adminUserId);
        return unavailable("IMMUTABLE_SCORE_HISTORY_NOT_AVAILABLE");
    }
    @Transactional(readOnly = true)
    public FairnessResponse.Unavailable rankingStability(Long adminUserId) {
        assertAdmin(adminUserId);
        return unavailable("RANKING_SNAPSHOT_HISTORY_NOT_AVAILABLE");
    }
    @Transactional(readOnly = true)
    public FairnessResponse.Appeals appeals(Long adminUserId, LocalDate from, LocalDate to) {
        assertAdmin(adminUserId);
        Instant captured = Instant.now();
        LocalDate today = LocalDateTime.ofInstant(captured, ZONE).toLocalDate();
        LocalDate end = to == null ? today : to;
        LocalDate start = from == null ? end.minusDays(30) : from;
        if (from == null ^ to == null) throw new BusinessException(ErrorCode.BAD_REQUEST, "from and to must
be provided together");
        if (!start.isBefore(end) || end.isAfter(today) || ChronoUnit.DAYS.between(start, end) > 90) {
            throw new BusinessException(ErrorCode.BAD_REQUEST, "invalid appeal date range");
        }
        FairnessMonitoringRepository.AppealAggregateSummary aggregate =
repository.findAppealAggregateSummary(
            start.atStartOfDay(ZONE).toLocalDateTime(), end.atStartOfDay(ZONE).toLocalDateTime());
        if (aggregate == null) aggregate = FairnessMonitoringRepository.AppealAggregateSummary.empty();
        return appealsFromSummary(start, end, captured, aggregate);
    }
    private FairnessResponse.Appeals appealsFromSummary(LocalDate start, LocalDate end, Instant captured,
FairnessMonitoringRepository.AppealAggregateSummary
a) {
        int population = a.sampleSize();
        boolean enough = population >= FairnessDisclosureGuard.MIN_INDEPENDENT_CANDIDATES;
        boolean comparableEnough = a.comparableRollbackCandidateCount() >=
FairnessDisclosureGuard.MIN_INDEPENDENT_CANDIDATES;
        boolean statusSuppressed = FairnessDisclosureGuard.suppressCells(population,
a.statusCandidateCounts().values(),
true);
        boolean reasonSuppressed = FairnessDisclosureGuard.suppressCells(population,
List.of(a.duplicateRollbackHistoryCount(), a.invalidScoreRollbackCount(),
a.ruleVersionChangedRollbackCount()),
true);
        boolean duplicateSuppressed = FairnessDisclosureGuard.suppressCells(population,
List.of(a.duplicateAppealCandidateCount()),
true);
        boolean approvalSuppressed = FairnessDisclosureGuard.suppressCells(population,
List.of(a.approvedEverCount()),
true)
        || FairnessDisclosureGuard.suppressRatio(a.approvalDenominatorCount(),
a.approvedEverCount());
        boolean rollbackSuppressed = FairnessDisclosureGuard.suppressRatio(a.approvedEverCount(),
a.rolledBackCandidateCount());
        // All statistics derived from comparable rollback rows share one
        // disclosure decision. A small comparable population must not leave
        // changed counts/rates or deltas visible beside a suppressed denominator.
        boolean comparableGroupSuppressed = a.comparableRollbackCount() > 0
            && FairnessDisclosureGuard.suppressCells(
                a.comparableRollbackCandidateCount(),
                List.of(a.changedCandidateCount(), a.unchangedCandidateCount()), true);
        boolean hidden = statusSuppressed || reasonSuppressed || duplicateSuppressed || approvalSuppressed
            || rollbackSuppressed;
        boolean comparableHidden = hidden || comparableGroupSuppressed;
        String availability = population == 0 ? "NO_DATA" : !enough ? "INSUFFICIENT_SAMPLE"
            : a.comparableRollbackCount() > 0 && !comparableEnough ? "INSUFFICIENT_SAMPLE"
            : (hidden || comparableGroupSuppressed) ? "SUPPRESSED" : "AVAILABLE";
        String reason = population == 0 ? "NO_APPEAL_POPULATION" : !enough ?
"INDEPENDENT_CANDIDATE_THRESHOLD_NOT_MET"
            : a.comparableRollbackCount() > 0 && !comparableEnough ?
"COMPARABLE_CANDIDATE_THRESHOLD_NOT_MET"
            : (hidden || comparableGroupSuppressed) ? "SMALL_CELL_OR_COMPLEMENT_RISK" : null;
        Map<String, Integer> reasons = new LinkedHashMap<>();
        reasons.put("DUPLICATE_ROLLBACK_HISTORY", a.duplicateRollbackHistoryCount());
        reasons.put("INVALID_SCORE", a.invalidScoreRollbackCount());
        reasons.put("RULE_VERSION_CHANGED", a.ruleVersionChangedRollbackCount());
    }

```

```

        BigDecimal meanDelta = a.comparableRollbackCount() == 0 ? null
            : a.deltaSum().divide(BigDecimal.valueOf(a.comparableRollbackCount()), 2,
RoundingMode.HALF_UP);
        BigDecimal meanAbs = a.comparableRollbackCount() == 0 ? null
            : a.absDeltaSum().divide(BigDecimal.valueOf(a.comparableRollbackCount()), 2,
RoundingMode.HALF_UP);
        boolean hideComparable = comparableHidden || !comparableEnough;
        return FairnessResponse.Appeals.builder().basis("APPEAL_CREATION_QUEUE_CURRENT_STATE")
            .from(start.toString()).to(end.toString()).capturedAt(captured).appealCount(a.appealCount()).
sampleSize(population)
            .statusCounts(hidden ? null : a.statusCandidateCounts()).statusEventCounts(hidden ? null :
a.statusEventCounts())
            .statusCandidateCounts(hidden ? null : a.statusCandidateCounts()).confidence(FairnessStatisti
csCalculator.confidence(population))
            .approvedEverCount(!hidden && enough ? a.approvedEverCount() : null)
            .approvalRate(hidden ? null : FairnessStatisticsCalculator.ratio(a.approvedEverCount(),
a.approvalDenominatorCount()))
            .rollbackExecutionRate(hidden ? null : FairnessStatisticsCalculator.ratio(a.rolledBackCandida
teCount(), a.approvedEverCount()))
            .duplicateAppealCount(!hidden && enough ? a.duplicateAppealCandidateCount() : null)
            .duplicateAppealRate(!hidden && enough ? FairnessStatisticsCalculator.ratio(a.duplicateAppeal
CandidateCount(), population) :
null)
            .comparableRollbackCount(hideComparable ? null : a.comparableRollbackCount())
            .comparableRollbackCandidateCount(hideComparable ? null :
a.comparableRollbackCandidateCount())
            .changedCount(comparableHidden ? null : a.changedCount())
            .changedCandidateCount(comparableHidden ? null : a.changedCandidateCount())
            .actualScoreChangeRate(comparableHidden ? null
                : FairnessStatisticsCalculator.ratio(a.changedCount(), a.comparableRollbackCount()))
            .meanDelta(comparableHidden ? null : meanDelta)
            .meanAbsoluteDelta(comparableHidden ? null : meanAbs)
            .duplicateRollbackHistoryCount(hidden ? null : a.duplicateRollbackHistoryCount())
            .invalidScoreRollbackCount(hidden ? null : a.invalidScoreRollbackCount())
            .ruleVersionChangedRollbackCount(hidden ? null : a.ruleVersionChangedRollbackCount())
            .reasonCounts(hidden ? null : reasons).availability(availability).reason(reason)
            .exposureAppealRateAvailability("UNAVAILABLE").exposureAppealRateReason("EXPOSURE_DENOMINATOR
_NOT_AVAILABLE").build();
    }
    private int sum(List<FairnessMonitoringRepository.SnapshotAggregateRow> rows) {
        return rows.stream().mapToInt(FairnessMonitoringRepository.SnapshotAggregateRow::candidateCount).sum(
);
    }
    private Map<String, Integer> groupCounts(List<FairnessMonitoringRepository.SnapshotAggregateRow> rows,
boolean status)
    {
        Map<String, Integer> result = new LinkedHashMap<>();
        for (var row : rows) {
            String key = status ? (KNOWN_STATUSES.contains(row.status()) ? row.status() : "UNKNOWN") :
normaliseVersion(row.rulesVersion());
            result.merge(key, row.candidateCount(), Integer::sum);
        }
        return result;
    }
    private boolean isValidAggregate(FairnessMonitoringRepository.SnapshotAggregateRow row) {
        return ("OK".equals(row.status()) || "ESTIMATED".equals(row.status())) && validScore(row.score());
    }
    private boolean validScore(BigDecimal value) {
        return value != null && value.compareTo(BigDecimal.ZERO) >= 0 &&
value.compareTo(BigDecimal.valueOf(100)) <=
0;
    }
    private String normaliseVersion(String value) { return KNOWN_RULES.contains(value) ? value : "UNKNOWN"; }
    private String highestRisk(Iterable<FairnessResponse.Distribution> distributions) {
        String risk = null;
        for (var d : distributions) {
            if ("HIGH".equals(d.getRiskLevel())) return "HIGH";
            if ("WARNING".equals(d.getRiskLevel())) risk = "WARNING";
            if ("NORMAL".equals(d.getRiskLevel()) && risk == null) risk = "NORMAL";
        }
        return risk;
    }
}

```

```

private int frequencyCount(List<FairnessStatisticsCalculator.ScoreFrequency> frequencies) {
    return frequencies == null ? 0 : frequencies.stream().filter(f -> f != null && f.count() > 0)
        .mapToInt(FairnessStatisticsCalculator.ScoreFrequency::count).sum();
}
private FairnessResponse.Distribution suppressedDistribution() {
    return FairnessResponse.Distribution.builder()
        .availability("SUPPRESSED")
        .reason("SMALL_CELL_OR_COMPLEMENT_RISK")
        .build();
}
private FairnessResponse.Unavailable unavailable(String reason) {
    return FairnessResponse.Unavailable.builder().availability("UNAVAILABLE").reason(reason).capturedAt(Instant.now()).build();
}
private String validatePeriod(String value, Instant capturedAt) {
    if (value == null || !value.matches("\\d{4}-\\d{2}")) throw new
BusinessException(ErrorCode.BAD_REQUEST, "period must be
yyyy-MM");
    try {
        LocalDate parsed = LocalDate.parse(value + "-01", DateTimeFormatter.ISO_LOCAL_DATE);
        String current = LocalDateTime.ofInstant(capturedAt, ZONE).toLocalDate().format(PERIOD);
        if (value.compareTo(current) > 0) throw new BusinessException(ErrorCode.BAD_REQUEST, "period
cannot be in the
future");
        return parsed.format(PERIOD);
    } catch (DateTimeParseException e) {
        throw new BusinessException(ErrorCode.BAD_REQUEST, "invalid period");
    }
}
}
package com.mydreamjob.scoringmonitoring;
import com.fasterxml.jackson.annotation.JsonInclude;
import lombok.AllArgsConstructor;
import lombok.Builder;
import lombok.Data;
import lombok.NoArgsConstructor;
import java.math.BigDecimal;
import java.time.Instant;
import java.util.List;
import java.util.Map;
/** V43 只读聚合白名单响应。所有内部用户、申诉和快照 ID 均留在服务端。 */
public final class FairnessResponse {
    private FairnessResponse() {}
    @Data
    @Builder
    @NoArgsConstructor
    @AllArgsConstructor
    @JsonInclude(JsonInclude.Include.NON_NULL)
    public static class Overview {
        private String basis;
        private String scope;
        private String period;
        private Instant capturedAt;
        private int populationSize;
        private BigDecimal computableSnapshotRatio;
        private String confidence;
        private String riskLevel;
        private Map<String, Integer> versionCounts;
        private Integer mixedVersionCount;
        private Integer unknownVersionCount;
        private Distribution distribution;
        private Map<String, Distribution> distributionsByRulesVersion;
        private List<String> coherentVersionSubset;
        private String availability;
        private String reason;
    }
    @Data
    @Builder
    @NoArgsConstructor
    @AllArgsConstructor
    @JsonInclude(JsonInclude.Include.NON_NULL)
    public static class Dimensions {

```

```

        private String basis;
        private String scope;
        private String period;
        private Instant capturedAt;
        private List<DimensionMetric> dimensions;
        private String availability;
        private String reason;
    }
    @Data
    @Builder
    @NoArgsConstructor
    @AllArgsConstructor
    @JsonInclude(JsonInclude.Include.NON_NULL)
    public static class DimensionMetric {
        private String dimension;
        private int populationSize;
        private Integer rowCount;
        private Integer missingRowCount;
        private Integer validSampleSize;
        private String confidence;
        private BigDecimal missingRowRate;
        private BigDecimal nullScoreRate;
        private BigDecimal validCoverage;
        private Map<String, Integer> statusCounts;
        private Map<String, Integer> versionCounts;
        private Integer inconsistentRecordCount;
        private Integer unknownStatusCount;
        private Integer outOfRangeScoreCount;
        private Integer unknownRulesVersionCount;
        private Distribution distribution;
        private Map<String, Distribution> distributionsByRulesVersion;
        private String availability;
        private String reason;
    }
    @Data
    @Builder
    @NoArgsConstructor
    @AllArgsConstructor
    @JsonInclude(JsonInclude.Include.NON_NULL)
    public static class Distribution {
        private Integer sampleSize;
        private BigDecimal mean;
        private BigDecimal median;
        private BigDecimal p10;
        private BigDecimal p25;
        private BigDecimal p75;
        private BigDecimal p90;
        private BigDecimal min;
        private BigDecimal max;
        private BigDecimal standardDeviation;
        private Map<String, Integer> buckets;
        private BigDecimal concentrationRate;
        private String riskLevel;
        private String availability;
        private String reason;
    }
    @Data
    @Builder
    @NoArgsConstructor
    @AllArgsConstructor
    @JsonInclude(JsonInclude.Include.NON_NULL)
    public static class Unavailable {
        private String availability;
        private String reason;
        private Instant capturedAt;
    }
    @Data
    @Builder
    @NoArgsConstructor
    @AllArgsConstructor
    @JsonInclude(JsonInclude.Include.NON_NULL)
    public static class Appeals {

```

```

private String basis;
private String from;
private String to;
private Instant capturedAt;
private int appealCount;
private int sampleSize;
private Map<String, Integer> statusCounts;
private Map<String, Integer> statusEventCounts;
private Map<String, Integer> statusCandidateCounts;
private String confidence;
private Integer approvedEverCount;
private BigDecimal approvalRate;
private BigDecimal rollbackExecutionRate;
private Integer duplicateAppealCount;
private BigDecimal duplicateAppealRate;
private Integer comparableRollbackCount;
private Integer comparableRollbackCandidateCount;
private Integer changedCount;
private Integer changedCandidateCount;
private BigDecimal actualScoreChangeRate;
private BigDecimal meanDelta;
private BigDecimal meanAbsoluteDelta;
private Integer duplicateRollbackHistoryCount;
private Integer invalidScoreRollbackCount;
private Integer ruleVersionChangedRollbackCount;
private Map<String, Integer> reasonCounts;
private String availability;
private String reason;
private String exposureAppealRateAvailability;
private String exposureAppealRateReason;
}
}
package com.mydreamjob.scoringmonitoring;
import java.math.BigDecimal;
import java.math.MathContext;
import java.math.RoundingMode;
import java.util.Comparator;
import java.util.LinkedHashMap;
import java.util.List;
import java.util.Map;
/** 无数据库的确定性统计函数。所有样本量均为独立候选人数。 */
public final class FairnessStatisticsCalculator {
    private static final MathContext MC = MathContext.DECIMAL128;
    private static final BigDecimal HUNDRED = BigDecimal.valueOf(100);
    private FairnessStatisticsCalculator() {}
    /** 数据库频数聚合的最小输入，不展开为候选人列表。 */
    public record ScoreFrequency(BigDecimal score, int count) {}
    public static String confidence(int n) {
        if (n < 30) return "INSUFFICIENT_SAMPLE";
        if (n < 100) return "LOW_CONFIDENCE";
        return "NORMAL_CONFIDENCE";
    }
    public static FairnessResponse.Distribution distribution(List<BigDecimal> input) {
        if (input == null || input.isEmpty()) {
            return FairnessResponse.Distribution.builder().sampleSize(0)
                .availability("NO_DATA").reason("NO_VALID_SCORE").build();
        }
        Map<BigDecimal, Integer> counts = new LinkedHashMap<>();
        input.stream().filter(v -> v != null).forEach(v -> counts.merge(v, 1, Integer::sum));
        return distributionFromFrequencies(counts.entrySet().stream()
            .map(e -> new ScoreFrequency(e.getKey(), e.getValue())).toList());
    }
    public static FairnessResponse.Distribution distributionFromFrequencies(List<ScoreFrequency> input) {
        if (input == null || input.isEmpty()) {
            return FairnessResponse.Distribution.builder().sampleSize(0)
                .availability("NO_DATA").reason("NO_VALID_SCORE").build();
        }
        List<ScoreFrequency> values = input.stream()
            .filter(v -> v != null && v.score() != null && v.count() > 0)
            .sorted(Comparator.comparing(ScoreFrequency::score)).toList();
        int n = values.stream().mapToInt(ScoreFrequency::count).sum();
        if (n == 0) {

```

```

        return FairnessResponse.Distribution.builder().sampleSize(0)
            .availability("NO_DATA").reason("NO_VALID_SCORE").build();
    }
    if (n < 30) {
        return FairnessResponse.Distribution.builder().sampleSize(n)
            .availability("INSUFFICIENT_SAMPLE")
            .reason("INDEPENDENT_CANDIDATE_THRESHOLD_NOT_MET").build();
    }
    Map<String, Integer> buckets = buckets(values);
    boolean suppressed = FairnessDisclosureGuard.suppressCells(n, buckets.values(), false);
    if (suppressed) {
        return FairnessResponse.Distribution.builder().sampleSize(null)
            .availability("SUPPRESSED").reason("SMALL_BUCKET_OR_COMPLEMENT_RISK").build();
    }
    BigDecimal meanExact = values.stream()
        .map(v -> v.score().multiply(BigDecimal.valueOf(v.count()), MC))
        .reduce(BigDecimal.ZERO, (a, b) -> a.add(b, MC))
        .divide(BigDecimal.valueOf(n), MC);
    BigDecimal variance = values.stream()
        .map(v -> v.score().subtract(meanExact, MC).pow(2, MC)
            .multiply(BigDecimal.valueOf(v.count()), MC))
        .reduce(BigDecimal.ZERO, (a, b) -> a.add(b, MC))
        .divide(BigDecimal.valueOf(n), MC);
    BigDecimal std = variance.max(BigDecimal.ZERO).sqrt(MC)
        .setScale(2, RoundingMode.HALF_UP);
    int maxBucket = buckets.values().stream().mapToInt(Integer::intValue).max().orElse(0);
    BigDecimal concentrationExact = BigDecimal.valueOf(maxBucket).multiply(HUNDRED, MC)
        .divide(BigDecimal.valueOf(n), MC);
    String risk = concentrationExact.compareTo(BigDecimal.valueOf(85)) >= 0 ? "HIGH"
        : concentrationExact.compareTo(BigDecimal.valueOf(70)) >= 0 ? "WARNING" : "NORMAL";
    return FairnessResponse.Distribution.builder()
        .sampleSize(n).availability("AVAILABLE")
        .mean(meanExact.setScale(2, RoundingMode.HALF_UP))
        .median(weightedPercentile(values, new BigDecimal("0.5")))
        .p10(weightedPercentile(values, new BigDecimal("0.1")))
        .p25(weightedPercentile(values, new BigDecimal("0.25")))
        .p75(weightedPercentile(values, new BigDecimal("0.75")))
        .p90(weightedPercentile(values, new BigDecimal("0.9")))
        .min(values.get(0).score().setScale(2, RoundingMode.HALF_UP))
        .max(values.get(values.size() - 1).score().setScale(2, RoundingMode.HALF_UP))
        .standardDeviation(std).buckets(buckets)
        .concentrationRate(concentrationExact.setScale(2, RoundingMode.HALF_UP))
        .riskLevel(risk).build();
}

public static BigDecimal ratio(int numerator, int denominator) {
    if (denominator <= 0) return null;
    return BigDecimal.valueOf(numerator).multiply(HUNDRED, MC)
        .divide(BigDecimal.valueOf(denominator), MC).setScale(2, RoundingMode.HALF_UP);
}

public static BigDecimal percentile(List<BigDecimal> sorted, BigDecimal p) {
    if (sorted == null || sorted.isEmpty()) return null;
    if (sorted.size() == 1) return sorted.get(0).setScale(2, RoundingMode.HALF_UP);
    BigDecimal h = BigDecimal.valueOf(sorted.size() - 1).multiply(p, MC);
    int i = h.setScale(0, RoundingMode.FLOOR).intValue();
    BigDecimal f = h.subtract(BigDecimal.valueOf(i), MC);
    BigDecimal left = sorted.get(i);
    BigDecimal right = sorted.get(Math.min(i + 1, sorted.size() - 1));
    return left.add(right.subtract(left, MC).multiply(f, MC), MC)
        .setScale(2, RoundingMode.HALF_UP);
}

private static BigDecimal weightedPercentile(List<ScoreFrequency> values, BigDecimal p) {
    int n = values.stream().mapToInt(ScoreFrequency::count).sum();
    if (n == 1) return values.get(0).score().setScale(2, RoundingMode.HALF_UP);
    BigDecimal h = BigDecimal.valueOf(n - 1).multiply(p, MC);
    int lower = h.setScale(0, RoundingMode.FLOOR).intValue();
    BigDecimal fraction = h.subtract(BigDecimal.valueOf(lower), MC);
    BigDecimal left = valueAt(values, lower);
    BigDecimal right = valueAt(values, Math.min(lower + 1, n - 1));
    return left.add(right.subtract(left, MC).multiply(fraction, MC), MC)
        .setScale(2, RoundingMode.HALF_UP);
}

private static BigDecimal valueAt(List<ScoreFrequency> values, int index) {

```

```

    int seen = 0;
    for (ScoreFrequency value : values) {
        if (index < seen + value.count()) return value.score();
        seen += value.count();
    }
    return values.get(values.size() - 1).score();
}

private static Map<String, Integer> buckets(List<ScoreFrequency> values) {
    Map<String, Integer> result = new LinkedHashMap<>();
    for (int i = 0; i < 10; i++) result.put(i * 10 + "-" + (i == 9 ? 100 : i * 10 + 9), 0);
    for (ScoreFrequency value : values) {
        int bucket = value.score().compareTo(BigDecimal.valueOf(100)) >= 0 ? 9
            : value.score().max(BigDecimal.ZERO).divide(BigDecimal.TEN, 0,
RoundingMode.FLOOR).intValue();
        String key = bucket * 10 + "-" + (bucket == 9 ? 100 : bucket * 10 + 9);
        result.computeIfPresent(key, (k, v) -> v + value.count());
    }
    return result;
}
}

package com.mydreamjob.security;
import com.mydreamjob.auth.SessionAuthenticationDetails;
import org.springframework.security.authentication.AnonymousAuthenticationToken;
import org.springframework.security.authentication.InsufficientAuthenticationException;
import org.springframework.security.core.Authentication;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.stereotype.Component;
/**
 * 安全关键路径的当前用户解析器。
 *
 * <p>身份<strong>只</strong>来自 JWT / SecurityContext, 不接受任何请求参数传入的 userId。
 * 所有非法主体一律转换为既有的“未登录”语义（{@link InsufficientAuthenticationException}
 * 由 {@code GlobalExceptionHandler} 映射为 HTTP 401），从而避免
 * {@code NumberFormatException} 逃逸为 HTTP 500。</p>
 *
 * <p>明确拒绝: Authentication 为 null、未认证、{@link AnonymousAuthenticationToken}、
 * 名称为空、名称为 anonymousUser、名称不是合法 Long、userId &lt;= 0。</p>
 *
 * <p>本类<strong>只</strong>负责身份解析, 不承担角色校验与资源归属校验—
 * 那两类判断仍由既有业务逻辑负责。</p>
 */
@Component
public class CurrentUserAccessor {
    /** 固定安全文本: 不回显任何非法 principal。 */
    static final String UNAUTHENTICATED_MESSAGE = "authentication required";
    private static final String ANONYMOUS_PRINCIPAL = "anonymousUser";
    /** 解析当前请求的用户 ID（读取 SecurityContext）。 */
    public Long requireUserId() {
        return requireUserId(SecurityContextHolder.getContext().getAuthentication());
    }
    /**
     * 解析当前请求**已验证**的登录会话 ID（S02）。
     *
     * <p>会话 ID 只可能由认证过滤器在 Access Token 通过完整校验后写入
     * {@code Authentication.details}; 客户端无法通过参数或请求体提供。
     * 取不到时一律按未登录处理。</p>
     */
    public String requireSessionId() {
        Authentication authentication = SecurityContextHolder.getContext().getAuthentication();
        if (authentication == null
            || !authentication.isAuthenticated()
            || authentication instanceof AnonymousAuthenticationToken) {
            throw unauthenticated();
        }
        if (authentication.getDetails() instanceof SessionAuthenticationDetails details) {
            String sessionId = details.getSessionId();
            if (sessionId != null && !sessionId.isBlank()) {
                return sessionId;
            }
        }
        throw unauthenticated();
    }
}

```

```

/**
 * 解析用户 ID。任何非法输入都抛出未登录异常，绝不抛出 NumberFormatException。
 *
 * @param authentication 待校验的认证对象（可为 null）
 * @return 大于 0 的已验证用户 ID
 */
public Long requireUserId(Authentication authentication) {
    if (authentication == null
        || !authentication.isAuthenticated()
        || authentication instanceof AnonymousAuthenticationToken) {
        throw unauthenticated();
    }
    String name = authentication.getName();
    if (name == null || name.isBlank() || ANONYMOUS_PRINCIPAL.equals(name)) {
        throw unauthenticated();
    }
    if (!isAsciiDigits(name)) {
        throw unauthenticated();
    }
    long userId;
    try {
        userId = Long.parseLong(name);
    } catch (NumberFormatException e) {
        // 位数溢出等极端情况：同样按未登录处理，不回显原始值
        throw unauthenticated();
    }
    if (userId <= 0) {
        throw unauthenticated();
    }
    return userId;
}
/** 严格校验：仅接受 ASCII 数字，排除 "+42"、"-42"、空白与 Unicode 数字。 */
private boolean isAsciiDigits(String value) {
    for (int i = 0; i < value.length(); i++) {
        char c = value.charAt(i);
        if (c < '0' || c > '9') {
            return false;
        }
    }
    return true;
}
private InsufficientAuthenticationException unauthenticated() {
    return new InsufficientAuthenticationException(UNAUTHENTICATED_MESSAGE);
}
}
package com.mydreamjob.security;
import com.mydreamjob.auth.User;
import com.mydreamjob.auth.UserRepository;
import com.mydreamjob.common.exception.BusinessException;
import com.mydreamjob.common.exception.ErrorCode;
import lombok.RequiredArgsConstructor;
import org.springframework.stereotype.Component;
/**
 * 业务角色授权守卫（R01-B/B1）。
 *
 * <p>把散落在各 Service/Controller 中的"读取用户 → 判断角色 → 抛错"重复实现，
 * 收束到唯一实现点。</p>
 *
 * <h2>职责边界</h2>
 * <ul>
 * <li><strong>只做角色授权</strong>，不做资源归属（ownership）校验 —
 * 归属仍由各业务 Service 负责（如 {@code assertOwnerOrThrow}）；</li>
 * <li><strong>不解析 SecurityContext</strong>： "当前登录用户是谁"仍由
 * {@link CurrentUserAccessor} 负责，本类只接收已解析出的 {@code userId}；</li>
 * <li><strong>不处理 requestId</strong>，不读请求头；</li>
 * <li><strong>不提供任何"由调用方传入角色"的入口</strong>：角色只能来自
 * {@link User.Role} 枚举常量，客户端输入无法影响判定；</li>
 * <li><strong>不给 ADMIN 隐式超级权限</strong>： {@code requireStudent}/{@code requireCompany}
 * 对 ADMIN 同样拒绝，除非调用方显式传入 {@code requireAny(..., ADMIN)}；</li>
 * <li>无副作用：只读用户表，不写库、不写审计、不记日志（尤其不记 userId / 手机号 / Token）。</li>
 * </ul>
 */

```

```

* <h2>角色来源</h2>
* <p>角色取自 {@code user.getRoles()}, 即 {@code user_roles} 关联表
* — 与本类替换掉的全部既有 guard 完全一致。{@code users.role} 单值列用于 JWT 声明签发,
* <strong>不</strong>参与此处授权判定: 把两者混用会改变既有语义。</p>
*
* <h2>失败关闭</h2>
* <ul>
* <li>用户不存在 → {@link ErrorCode#USER_NOT_FOUND} (沿用既有错误码); </li>
* <li>用户存在但 <strong>inactive</strong> → {@link ErrorCode#FORBIDDEN}
* (比既有实现更严格: 被禁用的用户不再因"角色正确"而通过); </li>
* <li>{@code user_roles} 中缺少所需角色 → {@link ErrorCode#FORBIDDEN}
* (数据不一致时同样是拒绝, 而不是放行)。</li>
* </ul>
*
* <h2>查询次数</h2>
* <p>{@code requireXxx(Long)} 每次调用恰好 <strong>1</strong> 次 {@code findById},
* 不做跨请求缓存(用户被注销/禁用后必须立即失效)。调用方若已加载
* {@link User}, 应使用 {@code requireXxx(User)} 重载以避免第二次查询。</p>
*/

```

@Component

@RequiredArgsConstructor

```

public class UserRoleGuard {
    private final UserRepository userRepository;
    // — 按 userId 校验 (每次 1 次查询) —————
    public void requireStudent(Long userId) {
        requireRole(requireUser(userId), User.Role.STUDENT);
    }
    public void requireCompany(Long userId) {
        requireRole(requireUser(userId), User.Role.COMPANY);
    }
    public void requireAdmin(Long userId) {
        requireRole(requireUser(userId), User.Role.ADMIN);
    }
    /** 允许集中的任意一个角色 (集合由调用方以枚举常量给出, 不接受客户端输入)。 */
    public void requireAny(Long userId, User.Role... allowedRoles) {
        requireRole(requireUser(userId), allowedRoles);
    }
    // — 按已加载实体校验 (0 次查询) —————
    public void requireStudent(User user) {
        requireRole(user, User.Role.STUDENT);
    }
    public void requireCompany(User user) {
        requireRole(user, User.Role.COMPANY);
    }
    public void requireAdmin(User user) {
        requireRole(user, User.Role.ADMIN);
    }
    public void requireAny(User user, User.Role... allowedRoles) {
        requireRole(user, allowedRoles);
    }
    // — 基础能力 —————
    /**
     * 加载用户并校验其可用 (存在且 active)。
     *
     * @throws BusinessException {@link ErrorCode#USER_NOT_FOUND} 用户不存在;
     *                             {@link ErrorCode#FORBIDDEN} 用户存在但已禁用
     */
    public User requireUser(Long userId) {
        if (userId == null || userId <= 0) {
            throw new BusinessException(ErrorCode.USER_NOT_FOUND);
        }
        User user = userRepository.findById(userId)
            .orElseThrow(() -> new BusinessException(ErrorCode.USER_NOT_FOUND));
        if (!isActive(user)) {
            // 失败关闭: 被禁用的用户不得因角色正确而通过授权。
            throw new BusinessException(ErrorCode.FORBIDDEN);
        }
        return user;
    }
}
/**
 * 校验已加载用户是否具备所需角色之一。
 */

```

```

* <p>角色来源为 {@code user_roles}; 缺失任一所需角色即拒绝。</p>
*/
public void requireRole(User user, User.Role... allowedRoles) {
    if (user == null) {
        throw new BusinessException(ErrorCode.USER_NOT_FOUND);
    }
    if (!isActive(user)) {
        throw new BusinessException(ErrorCode.FORBIDDEN);
    }
    if (allowedRoles == null || allowedRoles.length == 0) {
        throw new BusinessException(ErrorCode.FORBIDDEN);
    }
    if (user.getRoles() == null) {
        throw new BusinessException(ErrorCode.FORBIDDEN);
    }
    for (User.Role allowed : allowedRoles) {
        if (allowed != null && user.getRoles().contains(allowed)) {
            return;
        }
    }
    throw new BusinessException(ErrorCode.FORBIDDEN);
}

private static boolean isActive(User user) {
    return user.getActive() != null && user.getActive();
}
}

package com.mydreamjob.talentdiscovery;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.stereotype.Component;
import javax.crypto.Mac;
import javax.crypto.spec.SecretKeySpec;
import java.nio.charset.StandardCharsets;
import java.time.LocalDate;
import java.util.Base64;
/** 生成与公司、岗位和日期桶绑定的不可逆候选人展示键。 */
@Component
public class CandidateKeyGenerator {
    private static final String HMAC = "HmacSHA256";
    private final byte[] secret;
    public CandidateKeyGenerator(@Value("${app.jwt.secret:}") String configuredSecret) {
        this.secret = (configuredSecret == null ? "" : configuredSecret).getBytes(StandardCharsets.UTF_8);
    }
    public String generate(Long companyId, Long jobId, Long studentUserId, LocalDate capturedDate) {
        if (companyId == null || jobId == null || studentUserId == null || capturedDate == null ||
secret.length == 0)
        {
            throw new IllegalArgumentException("candidate key parameters are invalid");
        }
        String message = "talent-discovery:v1|" + companyId + "|" + jobId + "|" + studentUserId + "|" +
capturedDate;
        try {
            Mac mac = Mac.getInstance(HMAC);
            mac.init(new SecretKeySpec(secret, HMAC));
            return Base64.getUrlEncoder().withoutPadding().encodeToString(mac.doFinal(message.getBytes(Standar
rdCharsets.UTF_8)));
        } catch (Exception e) {
            throw new IllegalStateException("unable to generate candidate key", e);
        }
    }
}

package com.mydreamjob.talentdiscovery;
import jakarta.persistence.Column;
import jakarta.persistence.Entity;
import jakarta.persistence.EntityListeners;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.GenerationType;
import jakarta.persistence.Id;
import jakarta.persistence.Table;
import jakarta.persistence.Version;
import lombok.AllArgsConstructor;
import lombok.Builder;
import lombok.Data;

```

```

import lombok.NoArgsConstructor;
import org.hibernate.annotations.JdbcTypeCode;
import org.hibernate.type.SqlTypes;
import org.springframework.data.annotation.CreatedDate;
import org.springframework.data.annotation.LastModifiedDate;
import org.springframework.data.jpa.domain.support.AuditingEntityListener;
import java.time.LocalDateTime;
/** 学生对“被企业发现”的显式授权。无记录等价于关闭。 */
@Data
@Builder
@NoArgsConstructor
@AllArgsConstructor
@Entity
@Table(name = "student_talent_discovery_preferences")
@EntityListeners(AuditingEntityListener.class)
public class StudentTalentDiscoveryPreference {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    @Column(name = "student_user_id", nullable = false, unique = true)
    private Long studentUserId;
    @JdbcTypeCode(SqlTypes.TINYINT)
    @Column(nullable = false)
    @Builder.Default
    private Boolean enabled = false;
    @Version
    @Column(nullable = false)
    @Builder.Default
    private Long version = 0L;
    @CreatedDate
    @Column(name = "created_at", nullable = false, updatable = false)
    private LocalDateTime createdAt;
    @LastModifiedDate
    @Column(name = "updated_at", nullable = false)
    private LocalDateTime updatedAt;
}
package com.mydreamjob.talentdiscovery;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;
import java.util.Collection;
import java.util.List;
import java.util.Optional;
@Repository
public interface StudentTalentDiscoveryPreferenceRepository
    extends JpaRepository<StudentTalentDiscoveryPreference, Long> {
    Optional<StudentTalentDiscoveryPreference> findByStudentUserId(Long studentUserId);
    List<StudentTalentDiscoveryPreference> findByStudentUserIdInAndEnabledTrue(Collection<Long>
studentUserIds);
}
package com.mydreamjob.talentdiscovery;
import com.fasterxml.jackson.core.type.TypeReference;
import com.fasterxml.jackson.databind.ObjectMapper;
import lombok.RequiredArgsConstructor;
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.stereotype.Component;
import java.math.BigDecimal;
import java.sql.Timestamp;
import java.time.LocalDateTime;
import java.util.ArrayList;
import java.util.Collection;
import java.util.Collections;
import java.util.HashMap;
import java.util.HashSet;
import java.util.LinkedHashMap;
import java.util.List;
import java.util.Map;
import java.util.Set;
/**
 * V41 的批量只读数据源。这里集中保存资格 SQL 与聚合结果读取, Controller/Service 不执行 SQL。
 * 所有 ID 都通过 JDBC 参数绑定; 仅动态生成与参数数量相同的 ? 占位符。
 */
@Component

```

```

@RequiredArgsConstructor
public class TalentDiscoveryCandidateSource {
    private static final Set<String> RIASEC_DIMENSIONS = Set.of("R", "I", "A", "S", "E", "C");
    private static final Set<String> ABILITY_DIMENSIONS = Set.of("LOGIC", "COMMUNICATION", "EXECUTION",
"LEARNING");
    private static final TypeReference<List<Map<String, Object>>> LIST = new TypeReference<>() { };
    private static final TypeReference<List<String>> STRINGS = new TypeReference<>() { };
    private final JdbcTemplate jdbcTemplate;
    private final ObjectMapper objectMapper;
    /**
     * 读取全部合格候选, 不做数据库分页。账号注销限制和当前企业任意历史投递均在 SQL 中排除。
     */
    public List<CandidateSeed> findEligibleCandidates(Long companyId) {
        if (companyId == null) return List.of();
        String sql = ""
            SELECT u.id AS student_user_id, p.location, p.education, p.graduation_year,
                p.skills_json, p.updated_at
            FROM users u
            JOIN student_talent_discovery_preferences td ON td.student_user_id = u.id AND td.enabled = 1
            JOIN student_profiles p ON p.user_id = u.id
            WHERE u.role = 'STUDENT'
                AND u.is_active = 1
                AND NOT EXISTS (
                    SELECT 1 FROM account_deletions ad
                    WHERE ad.user_id = u.id AND ad.status IN ('PENDING', 'COMPLETED')
                )
                AND NOT EXISTS (
                    SELECT 1 FROM applications a
                    JOIN jobs j ON j.id = a.job_id
                    WHERE a.student_user_id = u.id AND j.company_id = ?
                )
            ORDER BY p.updated_at DESC, u.id DESC
        "";
        return jdbcTemplate.query(sql, (rs, rowNum) -> new CandidateSeed(
            rs.getLong("student_user_id"),
            rs.getString("location"),
            rs.getString("education"),
            rs.getObject("graduation_year", Short.class) == null ? null :
rs.getObject("graduation_year", Short.class).intValue(),
            rs.getString("skills_json"),
            timestamp(rs.getTimestamp("updated_at"))), companyId);
    }
    /** 批量读取偏好, 一次调用对应一条带 IN 参数的 SQL。 */
    public Map<Long, PreferenceSignal> findPreferences(Collection<Long> studentIds) {
        if (studentIds == null || studentIds.isEmpty()) return Map.of();
        List<Long> ids = distinctIds(studentIds);
        String sql = "SELECT student_user_id, expected_salary_min, expected_salary_max, preferred_cities,
employment_types,
"
            + "preferred_industries, preferred_job_types, remote_allowed FROM student_preferences WHERE
student_user_id IN
("
            + placeholders(ids.size()) + ")";
        Object[] args = ids.toArray();
        Map<Long, PreferenceSignal> out = new HashMap<>();
        jdbcTemplate.query(sql, rs -> {
            Long id = rs.getLong("student_user_id");
            out.put(id, new PreferenceSignal(
                rs.getObject("expected_salary_min", Integer.class),
                rs.getObject("expected_salary_max", Integer.class),
                readStrings(rs.getString("preferred_cities")),
                readStrings(rs.getString("employment_types")),
                readStrings(rs.getString("preferred_industries")),
                readStrings(rs.getString("preferred_job_types")),
                bool(rs.getObject("remote_allowed"))));
        }, args);
        return out;
    }
    /** 批量读取每位学生最新一份 RIASEC 结果: 先按时间选一条, 再解析, 禁止取历史最高。 */
    public Map<Long, ScoreSignal> findLatestRiasec(Collection<Long> studentIds) {
        if (studentIds == null || studentIds.isEmpty()) return Map.of();
        List<Long> ids = distinctIds(studentIds);

```

```

String sql = "SELECT a.student_user_id, a.id AS attempt_id, a.completed_at, r.answered_count,
r.dimension_scores_text
"
    + "FROM assessment_results r JOIN assessment_attempts a ON a.id=r.attempt_id "
    + "JOIN assessment_definitions d ON d.id=a.definition_id "
    + "WHERE a.student_user_id IN (" + placeholders(ids.size()) + ") "
    + "AND a.status='COMPLETED' AND d.type_code='HOLLAND_RIASEC' "
    + "ORDER BY a.student_user_id, a.completed_at DESC, a.id DESC";
Map<Long, ScoreSignal> out = new LinkedHashMap<>();
jdbcTemplate.query(sql, rs -> {
    Long id = rs.getLong("student_user_id");
    if (!out.containsKey(id)) {
        out.put(id, new ScoreSignal(parseScores(rs.getString("dimension_scores_text"),
RIASEC_DIMENSIONS,
            rs.getObject("answered_count", Integer.class), 30),
            timestamp(rs.getTimestamp("completed_at")), null));
    }
}, ids.toArray());
return out;
}
/** 批量读取每位学生最新、尚未过期的能力结果; valid_until 等于 capturedNow 不可用。 */
public Map<Long, ScoreSignal> findLatestAbility(Collection<Long> studentIds, LocalDateTime capturedNow) {
    if (studentIds == null || studentIds.isEmpty() || capturedNow == null) return Map.of();
    List<Long> ids = distinctIds(studentIds);
    String sql = "SELECT a.student_user_id, a.id AS attempt_id, a.completed_at, a.answered_count,
r.valid_until, r.dimension_scores_json
"
    + "FROM ability_results r JOIN ability_attempts a ON a.id=r.attempt_id "
    + "JOIN ability_definitions d ON d.id=a.definition_id "
    + "WHERE a.student_user_id IN (" + placeholders(ids.size()) + ") "
    + "AND a.status='COMPLETED' AND d.type_code='WORKPLACE_ABILITY' AND r.valid_until > ? "
    + "ORDER BY a.student_user_id, a.completed_at DESC, a.id DESC";
    List<Object> args = new ArrayList<>(ids);
    args.add(capturedNow);
    Map<Long, ScoreSignal> out = new LinkedHashMap<>();
    jdbcTemplate.query(sql, rs -> {
        Long id = rs.getLong("student_user_id");
        if (!out.containsKey(id)) {
            out.put(id, new ScoreSignal(parseScores(rs.getString("dimension_scores_json"),
ABILITY_DIMENSIONS,
                rs.getObject("answered_count", Integer.class), 24),
                timestamp(rs.getTimestamp("completed_at")),
timestamp(rs.getTimestamp("valid_until"))));
        }
    }, args.toArray());
    return out;
}
private Map<String, BigDecimal> parseScores(String raw, Set<String> expected, Integer totalAnswered, int
requiredTotal)
{
    if (raw == null || raw.isBlank() || totalAnswered == null || totalAnswered != requiredTotal) return
Map.of();
    try {
        List<Map<String, Object>> rows = objectMapper.readValue(raw, LIST);
        if (rows == null || rows.size() != expected.size()) return Map.of();
        Map<String, BigDecimal> out = new LinkedHashMap<>();
        for (Map<String, Object> row : rows) {
            // V35 RIASEC 持久化字段名是 dimension; V36 能力结果字段名是 code。
            // 按测评实际 JSON 结构选择字段, 禁止把任意字段静默当成维度。
            Object codeValue = expected.equals(ABILITY_DIMENSIONS)
                ? row.get("code")
                : (row.containsKey("dimension") ? row.get("dimension") : row.get("code"));
            if (!(codeValue instanceof String code) || !expected.contains(code) ||
out.containsKey(code)) return
Map.of();
            BigDecimal score = decimal(row.get("score"));
            if (score == null || score.compareTo(BigDecimal.ZERO) < 0 ||
score.compareTo(BigDecimal.valueOf(100)) > 0) return
Map.of();
            if (expected.equals(ABILITY_DIMENSIONS)) {
                Integer perDimension = exactInteger(row.get("answeredCount"));
                if (perDimension == null || perDimension != 6) return Map.of();
            }
        }
    }
}

```

```

        }
        out.put(code, score);
    }
    return out.keySet().equals(expected) ? out : Map.of();
} catch (Exception ignored) {
    return Map.of();
}
}

private List<String> readStrings(String raw) {
    if (raw == null || raw.isBlank()) return List.of();
    try {
        List<String> values = objectMapper.readValue(raw, STRINGS);
        if (values == null) return List.of();
        return values.stream().filter(v -> v != null && !v.trim().isEmpty()).map(String::trim).toList();
    } catch (Exception ignored) {
        return List.of();
    }
}

private BigDecimal decimal(Object value) {
    if (value == null) return null;
    try { return new BigDecimal(String.valueOf(value)); } catch (Exception e) { return null; }
}

private Integer exactInteger(Object value) {
    BigDecimal number = decimal(value);
    if (number == null) return null;
    try { return number.stripTrailingZeros().intValueExact(); } catch (ArithmeticException e) { return
null;
}
}

private boolean bool(Object value) {
    if (value instanceof Boolean b) return b;
    if (value instanceof Number n) return n.intValue() != 0;
    return "true".equalsIgnoreCase(String.valueOf(value)) || "1".equals(String.valueOf(value));
}

private static LocalDateTime timestamp(Timestamp timestamp) {
    return timestamp == null ? null : timestamp.toLocalDateTime();
}

private static List<Long> distinctIds(Collection<Long> ids) {
    return ids.stream().filter(java.util.Objects::nonNull).distinct().toList();
}

private static String placeholders(int count) {
    return String.join(",", Collections.nCopies(count, "?"));
}

public record CandidateSeed(Long studentUserId, String city, String education, Integer graduationYear,
    String skillsJson, LocalDateTime profileUpdatedAt) { }

public record PreferenceSignal(Integer expectedSalaryMin, Integer expectedSalaryMax, List<String> cities,
    List<String> employmentTypes, List<String> industries, List<String>
jobTypes,
    boolean remoteAllowed) { }

public record ScoreSignal(Map<String, BigDecimal> scores, LocalDateTime completedAt, LocalDateTime
validUntil) {
}
}

package com.mydreamjob.talentdiscovery;
import com.mydreamjob.audit.AuditContext;
import com.mydreamjob.common.api.ApiResponse;
import com.mydreamjob.common.exception.BusinessException;
import com.mydreamjob.common.exception.ErrorCode;
import com.mydreamjob.talentdiscovery.dto.TalentDiscoveryPageResponse;
import com.mydreamjob.talentdiscovery.dto.TalentDiscoveryPreferenceResponse;
import com.mydreamjob.talentdiscovery.dto.TalentDiscoveryPreferenceUpdateRequest;
import io.swagger.v3.oas.annotations.Operation;
import io.swagger.v3.oas.annotations.media.Content;
import io.swagger.v3.oas.annotations.media.Schema;
import io.swagger.v3.oas.annotations.responses.ApiResponses;
import io.swagger.v3.oas.annotations.security.SecurityRequirement;
import io.swagger.v3.oas.annotations.tags.Tag;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.validation.Valid;
import lombok.RequiredArgsConstructor;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PutMapping;

```

```

import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RestController;
/** V41 学生授权与企业人才发现接口; 身份只从 SecurityContext 获取。 */
@RestController
@RequiredArgsConstructor
@Tag(name = "企业人才发现", description = "学生显式授权后, 认证企业按自有岗位发现匿名候选人")
public class TalentDiscoveryController {
    /** R01-A4: 身份只来自安全上下文, 不接受请求参数传入的 userId。 */
    private final com.mydreamjob.security.CurrentUserAccessor currentUserAccessor;
    /** R01-A4: requestId 只来自过滤器规范化后的唯一产出点。 */
    private final com.mydreamjob.common.logging.RequestIdAccessor requestIdAccessor;
    private final TalentDiscoveryService service;
    @GetMapping("/profile/me/talent-discovery")
    @Operation(summary = "查询人才发现授权", security = @SecurityRequirement(name = "bearerAuth"))
    @ApiResponse(@io.swagger.v3.oas.annotations.responses.ApiResponse(responseCode = "200", content =
@Content(schema = @Schema(ref = "#/components/schemas/ApiResponseTalentDiscoveryPreferenceResponse"))))
    public ApiResponse<TalentDiscoveryPreferenceResponse> getPreference(HttpServletRequest request) {
        return ApiResponse.ok(service.getPreference(currentUserAccessor.requireUserId(),
requestIdAccessor.get(request)));
    }
    @PutMapping("/profile/me/talent-discovery")
    @Operation(summary = "更新人才发现授权", description = "仅当前学生本人可更新; expectedVersion
用于乐观锁并发控制。", security = @SecurityRequirement(name =
"bearerAuth"))
    @ApiResponses(@io.swagger.v3.oas.annotations.responses.ApiResponse(responseCode = "200", content =
@Content(schema = @Schema(ref = "#/components/schemas/ApiResponseTalentDiscoveryPreferenceResponse"))))
    public ApiResponse<TalentDiscoveryPreferenceResponse> updatePreference(
        @Valid @RequestBody TalentDiscoveryPreferenceUpdateRequest body, HttpServletRequest request) {
        return ApiResponse.ok(service.updatePreference(currentUserAccessor.requireUserId(), body,
AuditContext.from(request)), requestIdAccessor.get(request));
    }
    @GetMapping("/companies/me/jobs/{jobId}/talent-recommendations")
    @Operation(summary = "查询岗位人才发现推荐", description =
"仅已认证企业可查询本人已发布且具备审核通过画像的岗位; 候选卡不包含学生身份或测评详情。", security =
@SecurityRequirement(name =
"bearerAuth"))
    @ApiResponses(@io.swagger.v3.oas.annotations.responses.ApiResponse(responseCode = "200", content =
@Content(schema = @Schema(ref = "#/components/schemas/ApiResponseTalentDiscoveryPageResponse"))))
    public ApiResponse<TalentDiscoveryPageResponse> recommendations(
        @PathVariable Long jobId,
        @RequestParam(defaultValue = "0") int page,
        @RequestParam(defaultValue = "20") int size,
        HttpServletRequest request) {
        return ApiResponse.ok(service.recommendations(currentUserAccessor.requireUserId(), jobId, page,
size), requestIdAccessor.get(request));
    }
}
package com.mydreamjob.talentdiscovery;
import org.springframework.stereotype.Component;
import java.math.BigDecimal;
import java.math.RoundingMode;
import java.util.ArrayList;
import java.util.List;
import java.util.Optional;
/** V41 人才发现评分纯函数; 不访问数据库, 也不生成对外的数值分数。 */
@Component
public class TalentDiscoveryScoreCalculator {
    public static final BigDecimal PREFERENCE_WEIGHT = new BigDecimal("40");
    public static final BigDecimal RIASec_WEIGHT = new BigDecimal("35");
    public static final BigDecimal ABILITY_WEIGHT = new BigDecimal("25");
    private static final BigDecimal MIN = BigDecimal.ZERO;
    private static final BigDecimal MAX = new BigDecimal("100");
    public record Calculation(BigDecimal fitScore, String fitBand, List<String> signalsAvailable) { }
    /** 缺失信号不按 0 处罚, 仅对有效信号权重重新归一化。 */
    public Optional<Calculation> calculate(BigDecimal preference, BigDecimal riasec, BigDecimal ability) {
        BigDecimal numerator = BigDecimal.ZERO;
        BigDecimal denominator = BigDecimal.ZERO;
        List<String> available = new ArrayList<>();
        if (valid(preference)) { numerator = numerator.add(preference.multiply(PREFERENCE_WEIGHT));
denominator = denominator.add(PREFERENCE_WEIGHT); available.add("PREFERENCE");

```

```

    }
    if (valid(riasec)) { numerator = numerator.add(riasec.multiply(RIASEC_WEIGHT)); denominator =
denominator.add(RIASEC_WEIGHT); available.add("RIASEC");
    }
    if (valid(ability)) { numerator = numerator.add(ability.multiply(ABILITY_WEIGHT)); denominator =
denominator.add(ABILITY_WEIGHT); available.add("ABILITY");
    }
    if (denominator.signum() == 0) return Optional.empty();
    BigDecimal score = numerator.divide(denominator, 2, RoundingMode.HALF_UP);
    return Optional.of(new Calculation(score, band(score), List.copyOf(available)));
}
public String band(BigDecimal score) {
    if (score == null) return null;
    if (score.compareTo(new BigDecimal("75")) >= 0) return "HIGH";
    if (score.compareTo(new BigDecimal("60")) >= 0) return "MEDIUM";
    return "BASIC";
}
private boolean valid(BigDecimal value) {
    return value != null && value.compareTo(MIN) >= 0 && value.compareTo(MAX) <= 0
        && value.scale() < 1000;
}
}
package com.mydreamjob.talentdiscovery;
import com.fasterxml.jackson.core.type.TypeReference;
import com.fasterxml.jackson.databind.ObjectMapper;
import com.mydreamjob.audit.AuditContext;
import com.mydreamjob.audit.AuditService;
import com.mydreamjob.auth.User;
import com.mydreamjob.auth.UserRepository;
import com.mydreamjob.common.exception.BusinessException;
import com.mydreamjob.common.exception.ErrorCode;
import com.mydreamjob.company.Company;
import com.mydreamjob.company.CompanyRepository;
import com.mydreamjob.job.Job;
import com.mydreamjob.job.JobRepository;
import com.mydreamjob.recommendation.JobPreferenceScoreCalculator;
import com.mydreamjob.recommendation.JobRecommendationProfile;
import com.mydreamjob.recommendation.JobRecommendationProfileRepository;
import com.mydreamjob.talentdiscovery.dto.TalentDiscoveryCandidateResponse;
import com.mydreamjob.talentdiscovery.dto.TalentDiscoveryPageResponse;
import com.mydreamjob.talentdiscovery.dto.TalentDiscoveryPreferenceResponse;
import com.mydreamjob.talentdiscovery.dto.TalentDiscoveryPreferenceUpdateRequest;
import lombok.RequiredArgsConstructor;
import org.springframework.dao.DataIntegrityViolationException;
import org.springframework.orm.ObjectOptimisticLockingFailureException;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;
import java.math.BigDecimal;
import java.math.RoundingMode;
import java.time.LocalDate;
import java.time.LocalDateTime;
import java.util.ArrayList;
import java.util.Comparator;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.Set;
import java.util.regex.Pattern;
/** V41 企业人才发现服务。只读候选池，不写推荐历史、曝光或候选评分快照。 */
@Service
@RequiredArgsConstructor
public class TalentDiscoveryService {
    private static final ObjectMapper JSON = new ObjectMapper();
    private static final TypeReference<Map<String, Object>> MAP = new TypeReference<>() { };
    private static final TypeReference<List<String>> LIST = new TypeReference<>() { };
    private static final int BATCH_SIZE = 500;
    private static final Pattern SAFE_SKILL = Pattern.compile("[\\p{L}\\p{N}]+#./( ) ·\\- _ ]+");
    private final com.mydreamjob.security.UserRoleGuard userRoleGuard;
    private final StudentTalentDiscoveryPreferenceRepository preferenceRepository;
    private final UserRepository userRepository;
    private final CompanyRepository companyRepository;
    private final JobRepository jobRepository;

```

```

private final JobRecommendationProfileRepository profileRepository;
private final TalentDiscoveryCandidateSource candidateSource;
private final TalentDiscoveryScoreCalculator scoreCalculator;
private final CandidateKeyGenerator candidateKeyGenerator;
private final AuditService auditService;
@Transactional(readOnly = true)
public TalentDiscoveryPreferenceResponse getPreference(Long userId) {
    userRoleGuard.requireStudent(userId);
    StudentTalentDiscoveryPreference value = preferenceRepository.findByStudentUserId(userId).orElse(null
);
    return toPreference(value);
}
@Transactional
public TalentDiscoveryPreferenceResponse updatePreference(Long userId,
                                                         TalentDiscoveryPreferenceUpdateRequest request,
                                                         AuditContext context) {
    userRoleGuard.requireStudent(userId);
    if (request == null || request.getEnabled() == null) {
        throw new BusinessException(ErrorCodes.BAD_REQUEST);
    }
    StudentTalentDiscoveryPreference existing = preferenceRepository.findByStudentUserId(userId).orElse(n
ull);
    Long expected = request.getExpectedVersion();
    if (existing == null && expected != null && expected != 0L) {
        throw new BusinessException(ErrorCodes.PRIVACY_SETTINGS_CONCURRENT_UPDATE);
    }
    if (existing != null && expected != null && !java.util.Objects.equals(existing.getVersion(),
expected))
    {
        throw new BusinessException(ErrorCodes.PRIVACY_SETTINGS_CONCURRENT_UPDATE);
    }
    try {
        StudentTalentDiscoveryPreference saved;
        if (existing == null) {
            saved = preferenceRepository.saveAndFlush(StudentTalentDiscoveryPreference.builder()
                .studentUserId(userId).enabled(request.getEnabled()).version(0L).build());
        } else {
            existing.setEnabled(request.getEnabled());
            saved = preferenceRepository.saveAndFlush(existing);
        }
        String requestId = context == null ? null : context.getRequestId();
        String details = "{\n\"enabled\": \" + saved.getEnabled()
            + \"\n\"version\": \" + saved.getVersion()
            + \"\n\"requestId\": \" + (requestId == null ? \"null\" : quote(requestId)) + \"\n\"}";
        auditService.log(userId, "TALENT_DISCOVERY_PREFERENCE_UPDATED",
            "STUDENT_TALENT_DISCOVERY_PREFERENCE", saved.getId(), details, context);
        return toPreference(saved);
    } catch (ObjectOptimisticLockingFailureException | DataIntegrityViolationException ex) {
        throw new BusinessException(ErrorCodes.PRIVACY_SETTINGS_CONCURRENT_UPDATE);
    }
}
@Transactional(readOnly = true)
public TalentDiscoveryPageResponse recommendations(Long userId, Long jobId, int page, int size) {
    validatePage(jobId, page, size);
    Company company = requireVerifiedCompany(userId);
    Job job = jobRepository.findById(jobId).orElseThrow(() -> new
BusinessException(ErrorCodes.JOB_NOT_FOUND));
    if (!company.getId().equals(job.getCompanyId())) {
        throw new BusinessException(ErrorCodes.FORBIDDEN);
    }
    if (!"PUBLISHED".equals(job.getStatus())) {
        throw new BusinessException(ErrorCodes.INVALID_JOB_STATUS_TRANSITION);
    }
    JobRecommendationProfile profile = profileRepository.findById(jobId).orElse(null);
    if (profile == null || !Boolean.TRUE.equals(profile.getEnabled())
        || !JobRecommendationProfile.APPROVED.equals(profile.getReviewStatus())) {
        return page("UNAVAILABLE", List.of(), 0, page, size, "该岗位尚无审核通过的推荐画像");
    }
    LocalDateTime capturedNow = LocalDateTime.now();
    List<TalentDiscoveryCandidateSource.CandidateSeed> seeds =
candidateSource.findEligibleCandidates(company.getId());
    if (seeds.isEmpty()) {

```

```

        return page("NO_ELIGIBLE_CANDIDATE", List.of(), 0, page, size,
"暂未发现已授权且满足基础条件的候选人");
    }
    Map<Long, TalentDiscoveryCandidateSource.PreferenceSignal> preferences = new HashMap<>();
    Map<Long, TalentDiscoveryCandidateSource.ScoreSignal> riasec = new HashMap<>();
    Map<Long, TalentDiscoveryCandidateSource.ScoreSignal> ability = new HashMap<>();
    for (int from = 0; from < seeds.size(); from += BATCH_SIZE) {
        int to = Math.min(from + BATCH_SIZE, seeds.size());
        List<Long> ids = seeds.subList(from, to).stream()
            .map(TalentDiscoveryCandidateSource.CandidateSeed::studentUserId).toList();
        preferences.putAll(candidateSource.findPreferences(ids));
        riasec.putAll(candidateSource.findLatestRiasec(ids));
        ability.putAll(candidateSource.findLatestAbility(ids, capturedNow));
    }
    List<ScoredCandidate> scored = new ArrayList<>();
    for (TalentDiscoveryCandidateSource.CandidateSeed seed : seeds) {
        BigDecimal p = preferenceScore(preferences.get(seed.studentUserId()), profile, job);
        TalentDiscoveryCandidateSource.ScoreSignal iSignal = riasec.get(seed.studentUserId());
        BigDecimal i = weighted(iSignal == null ? Map.of() : iSignal.scores(),
parseMap(profile.getRiasecWeightsJson()), Set.of("R", "I", "A", "S", "E",
"C"));
        TalentDiscoveryCandidateSource.ScoreSignal aSignal = ability.get(seed.studentUserId());
        BigDecimal a = weighted(aSignal == null ? Map.of() : aSignal.scores(),
parseMap(profile.getAbilityExpectationsJson()), Set.of("LOGIC", "COMMUNICATION", "EXECUTION",
"LEARNING"));
        var calculation = scoreCalculator.calculate(p, i, a);
        if (calculation.isEmpty()) continue;
        TalentDiscoveryScoreCalculator.Calculation value = calculation.get();
        scored.add(new ScoredCandidate(seed, value));
    }
    if (scored.isEmpty()) {
        return page("NO_ELIGIBLE_CANDIDATE", List.of(), 0, page, size,
            "暂未发现已授权且满足基础条件的候选人");
    }
    scored.sort(Comparator.comparing((ScoredCandidate c) -> c.calculation.fitScore()).reversed()
        .thenComparing(c -> c.seed.profileUpdatedAt(), Comparator.nullsLast(Comparator.reverseOrder()
)))
        .thenComparing(c -> c.seed.studentUserId(), Comparator.reverseOrder()));
    int total = scored.size();
    long offset = (long) page * size;
    int from = offset >= total ? total : (int) offset;
    int to = Math.min(from + size, total);
    List<TalentDiscoveryCandidateResponse> items = scored.subList(from, to).stream()
        .map(c -> toCandidate(c, company.getId(), job.getId(), capturedNow.toLocalDate())).toList();
    return page("PERSONALIZED", items, total, page, size, null);
}

private TalentDiscoveryCandidateResponse toCandidate(ScoredCandidate scored, Long companyId, Long jobId,
LocalDate date)
{
    TalentDiscoveryCandidateSource.CandidateSeed seed = scored.seed;
    return TalentDiscoveryCandidateResponse.builder()
        .candidateKey(candidateKeyGenerator.generate(companyId, jobId, seed.studentUserId(), date))
        .candidateLabel("开放人才候选人")
        .city(safeText(seed.city(), 100))
        .education(safeText(seed.education(), 100))
        .graduationYear(seed.graduationYear())
        .skills(safeSkills(seed.skillsJson()))
        .fitBand(scored.calculation.fitBand())
        .summary(summary(scored.calculation.fitBand()))
        .signalsAvailable(scored.calculation.signalsAvailable())
        .build();
}
/**
 * R01-C: 改为委托 {@link JobPreferenceScoreCalculator}。
 *
 * <p>原先此处内联了与 {@code JobRecommendationService} 完全相同的一套 30/20/20/15/15 规则
 * (维度顺序、权重、匹配语义、{@code matched*100/available}、scale=2、HALF_UP 与
 * available==0 → null 均逐字一致)。收束后两处共用唯一实现，**行为不变**。</p>
 */
private BigDecimal preferenceScore(TalentDiscoveryCandidateSource.PreferenceSignal preference,
    JobRecommendationProfile profile, Job job) {
    if (preference == null) return null;

```

```

return JobPreferenceScoreCalculator.calculate(
    new JobPreferenceScoreCalculator.Criteria(
        preference.cities(),
        preference.employmentTypes(),
        preference.expectedSalaryMin(),
        preference.industries(),
        preference.jobTypes(),
        new JobPreferenceScoreCalculator.Target(
            job.getCity(),
            job.getEmploymentType(),
            job.getSalaryMax(),
            profile.getIndustryCode(),
            profile.getJobFamilyCode()));
}
private BigDecimal weighted(Map<String, BigDecimal> scores, Map<String, BigDecimal> expected,
Set<String> allowed)
{
    if (scores.isEmpty() || expected.isEmpty()) return null;
    BigDecimal numerator = BigDecimal.ZERO;
    BigDecimal denominator = BigDecimal.ZERO;
    for (Map.Entry<String, BigDecimal> entry : expected.entrySet()) {
        if (!allowed.contains(entry.getKey()) || !scores.containsKey(entry.getKey()) || entry.getValue()
== null || entry.getValue().signum() <= 0)
continue;
        numerator = numerator.add(scores.get(entry.getKey()).multiply(entry.getValue()));
        denominator = denominator.add(entry.getValue());
    }
    return denominator.signum() == 0 ? null : numerator.divide(denominator, 2, RoundingMode.HALF_UP);
}
private Map<String, BigDecimal> parseMap(String raw) {
    if (raw == null || raw.isBlank()) return Map.of();
    try {
        Map<String, Object> input = JSON.readValue(raw, MAP);
        Map<String, BigDecimal> result = new HashMap<>();
        for (Map.Entry<String, Object> entry : input.entrySet()) {
            if (entry.getKey() == null || entry.getValue() == null) return Map.of();
            BigDecimal value = new BigDecimal(String.valueOf(entry.getValue()));
            if (value.compareTo(BigDecimal.ZERO) < 0 || value.compareTo(BigDecimal.valueOf(100)) > 0)
return Map.of();
            result.put(entry.getKey(), value);
        }
        return result;
    } catch (Exception ignored) { return Map.of(); }
}
private List<String> safeSkills(String raw) {
    if (raw == null || raw.isBlank()) return List.of();
    try {
        List<String> values = JSON.readValue(raw, LIST);
        if (values == null) return List.of();
        return values.stream().map(v -> safeSkill(v)).filter(v -> v != null && !v.isBlank())
            .distinct().limit(10).toList();
    } catch (Exception ignored) { return List.of(); }
}
private String safeSkill(String value) {
    String text = safeText(value, 40);
    return text != null && SAFE_SKILL.matcher(text).matches() ? text : null;
}
private String safeText(String value, int max) {
    if (value == null) return null;
    String text = value.trim();
    if (text.isEmpty()) return null;
    return text.length() <= max ? text : text.substring(0, max);
}
private String summary(String band) {
    return switch (band) {
        case "HIGH" -> "岗位画像与候选人公开资料存在较高匹配度";
        case "MEDIUM" -> "岗位画像与候选人公开资料存在一定匹配度";
        default -> "候选人满足部分岗位画像条件";
    };
}
private Company requireVerifiedCompany(Long userId) {
    User user = requireUser(userId);

```

```

        if (!hasRole(user, User.Role.COMPANY)) throw new BusinessException(ErrorCode.FORBIDDEN);
        Company company = companyRepository.findByOwnerId(userId)
            .orElseThrow(() -> new BusinessException(ErrorCode.COMPANY_NOT_FOUND));
        if (!Boolean.TRUE.equals(company.getVerified())) throw new BusinessException(ErrorCode.FORBIDDEN);
        return company;
    }
    private User requireUser(Long userId) {
        if (userId == null) throw new BusinessException(ErrorCode.UNAUTHORIZED);
        return userRepository.findById(userId).orElseThrow(() -> new
BusinessException(ErrorCode.USER_NOT_FOUND));
    }
    private boolean hasRole(User user, User.Role role) {
        return user != null && ((user.getRoles() != null && user.getRoles().contains(role)) ||
user.getRole() ==
role);
    }
    private void validatePage(Long jobId, int page, int size) {
        if (jobId == null || jobId <= 0 || page < 0 || size < 1 || size > 50) {
            throw new BusinessException(ErrorCode.BAD_REQUEST);
        }
    }
    private TalentDiscoveryPreferenceResponse toPreference(StudentTalentDiscoveryPreference value) {
        if (value == null) return TalentDiscoveryPreferenceResponse.builder().enabled(false).version(0L).build();
        return TalentDiscoveryPreferenceResponse.builder().enabled(Boolean.TRUE.equals(value.getEnabled()))
            .version(value.getVersion() == null ? 0L : value.getVersion())
            .createdAt(value.getCreatedAt()).updatedAt(value.getUpdatedAt()).build();
    }
    private TalentDiscoveryPageResponse page(String mode, List<TalentDiscoveryCandidateResponse> items, long
total,
        int page, int size, String message) {
        int totalPages = total == 0 ? 0 : (int) Math.ceil((double) total / size);
        return TalentDiscoveryPageResponse.builder().mode(mode).items(items).total(total)
            .page(page).size(size).totalPages(totalPages).message(message).build();
    }
    private String quote(String value) {
        return "\"" + value.replace("\\", "\\\\").replace("\"", "\\\"") + "\"";
    }
    private record ScoredCandidate(TalentDiscoveryCandidateSource.CandidateSeed seed,
        TalentDiscoveryScoreCalculator.Calculation calculation) { }
}
package com.mydreamjob.talentdiscovery.dto;
import io.swagger.v3.oas.annotations.media.Schema;
import lombok.AllArgsConstructor;
import lombok.Builder;
import lombok.Data;
import lombok.NoArgsConstructor;
import java.util.List;
@Data
@Builder
@NoArgsConstructor
@AllArgsConstructor
@Schema(description = "企业人才发现候选人卡片; 不含候选人身份、联系方式或测评分数")
public class TalentDiscoveryCandidateResponse {
    private String candidateKey;
    private String candidateLabel;
    private String city;
    private String education;
    private Integer graduationYear;
    private List<String> skills;
    private String fitBand;
    private String summary;
    private List<String> signalsAvailable;
}
package com.mydreamjob.talentdiscovery.dto;
import io.swagger.v3.oas.annotations.media.Schema;
import lombok.AllArgsConstructor;
import lombok.Builder;
import lombok.Data;
import lombok.NoArgsConstructor;
import java.util.List;
@Data

```

```

@Builder
@NoArgsConstructor
@AllArgsConstructor
public class TalentDiscoveryPageResponse {
    @Schema(description = "PERSONALIZED、UNAVAILABLE 或 NO_ELIGIBLE_CANDIDATE")
    private String mode;
    private List<TalentDiscoveryCandidateResponse> items;
    private long total;
    private int page;
    private int size;
    private int totalPages;
    private String message;
}

package com.mydreamjob.talentdiscovery.dto;
import com.fasterxml.jackson.annotation.JsonInclude;
import io.swagger.v3.oas.annotations.media.Schema;
import lombok.AllArgsConstructor;
import lombok.Builder;
import lombok.Data;
import lombok.NoArgsConstructor;
import java.time.LocalDateTime;
@Data
@Builder
@NoArgsConstructor
@AllArgsConstructor
@JsonInclude(JsonInclude.Include.NON_NULL)
public class TalentDiscoveryPreferenceResponse {
    @Schema(description = "是否允许进入企业人才发现候选池")
    private Boolean enabled;
    @Schema(description = "乐观锁版本号")
    private Long version;
    private LocalDateTime createdAt;
    private LocalDateTime updatedAt;
}

package com.mydreamjob.talentdiscovery.dto;
import io.swagger.v3.oas.annotations.media.Schema;
import jakarta.validation.constraints.NotNull;
import jakarta.validation.constraints.PositiveOrZero;
import lombok.Data;
@Data
public class TalentDiscoveryPreferenceUpdateRequest {
    @NotNull
    @Schema(description = "是否允许进入企业人才发现候选池", requiredMode = Schema.RequiredMode.REQUIRED)
    private Boolean enabled;
    @PositiveOrZero
    @Schema(description = "期望的当前版本; 首次创建可省略或传 0")
    private Long expectedVersion;
}

package com.mydreamjob.talentinvitation;
import jakarta.persistence.*;
import lombok.*;
import org.hibernate.annotations.JdbcTypeCode;
import org.hibernate.type.SqlTypes;
import org.springframework.data.annotation.CreatedDate;
import org.springframework.data.annotation.LastModifiedDate;
import org.springframework.data.jpa.domain.support.AuditingEntityListener;
import java.time.LocalDateTime;
@Data
@Builder
@NoArgsConstructor
@AllArgsConstructor
@Entity
@Table(name = "talent_invitations")
@EntityListeners(AuditingEntityListener.class)
public class TalentInvitation {
    public static final String PENDING = "PENDING";
    public static final String ACCEPTED = "ACCEPTED";
    public static final String REJECTED = "REJECTED";
    public static final String EXPIRED = "EXPIRED";
    @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    @Column(name = "company_id", nullable = false) private Long companyId;
}

```

```

@Column(name = "job_id", nullable = false) private Long jobId;
@Column(name = "student_user_id", nullable = false) private Long studentUserId;
/**
 * V40 已在 MySQL 中以 CHAR(64) 建表; 显式声明 JDBC 类型, 避免
 * Hibernate 将 String 默认推断为 VARCHAR(64) 导致 ddl-auto=validate 失败。
 */
@JdbcTypeCode(SqlTypes.CHAR)
@Column(name = "candidate_key_digest", nullable = false, length = 64, columnDefinition = "CHAR(64)")
private String candidateKeyDigest;
@Column(nullable = false, length = 300) private String message;
@Column(nullable = false, length = 20) @Builder.Default private String status = PENDING;
@Column(name = "idempotency_key", nullable = false, length = 64) private String idempotencyKey;
@Column(name = "active_key", length = 200) private String activeKey;
@Column(name = "conversation_id", unique = true) private Long conversationId;
@Column(name = "expires_at", nullable = false) private LocalDateTime expiresAt;
@Column(name = "decided_at") private LocalDateTime decidedAt;
@Version @Column(nullable = false) @Builder.Default private Long version = 0L;
@CreatedDate @Column(name = "created_at", nullable = false, updatable = false) private LocalDateTime
createdAt;
@LastModifiedDate @Column(name = "updated_at", nullable = false) private LocalDateTime updatedAt;
}
package com.mydreamjob.talentinvitation;
import com.mydreamjob.talentdiscovery.CandidateKeyGenerator;
import com.mydreamjob.talentdiscovery.TalentDiscoveryCandidateSource;
import lombok.RequiredArgsConstructor;
import org.springframework.stereotype.Component;
import java.nio.charset.StandardCharsets;
import java.security.MessageDigest;
import java.time.LocalDate;
import java.time.LocalDateTime;
import java.util.Map;
import java.util.List;
/** 在当前 V41 合格候选池内重算 opaque key, 绝不通过 key 反解学生 ID。 */
@Component
@RequiredArgsConstructor
public class TalentInvitationCandidateResolver {
    private final TalentDiscoveryCandidateSource candidateSource;
    private final CandidateKeyGenerator candidateKeyGenerator;
    public Resolution resolve(Long companyId, Long jobId, String candidateKey, LocalDateTime capturedNow) {
        if (companyId == null || jobId == null || candidateKey == null || capturedNow == null) {
            return Resolution.none();
        }
        byte[] expected = candidateKey.getBytes(StandardCharsets.UTF_8);
        Long match = null;
        int matches = 0;
        List<TalentDiscoveryCandidateSource.CandidateSeed> candidates =
candidateSource.findEligibleCandidates(companyId);
        List<Long> candidateIds = candidates.stream().map(TalentDiscoveryCandidateSource.CandidateSeed::stude
ntUserId).toList();
        Map<Long, TalentDiscoveryCandidateSource.PreferenceSignal> preferences =
safeMap(candidateSource.findPreferences(candidateIds));
        Map<Long, TalentDiscoveryCandidateSource.ScoreSignal> riasec =
safeMap(candidateSource.findLatestRiasec(candidateIds));
        Map<Long, TalentDiscoveryCandidateSource.ScoreSignal> ability =
safeMap(candidateSource.findLatestAbility(candidateIds,
capturedNow));
        LocalDate bucket = capturedNow.toLocalDate();
        for (TalentDiscoveryCandidateSource.CandidateSeed candidate : candidates) {
            // 候选池要求至少有一个可用推荐信号; 缺失信号不得按 0 分放行。
            if (!hasSignal(candidate.studentUserId(), preferences, riasec, ability)) continue;
            String generated = candidateKeyGenerator.generate(companyId, jobId, candidate.studentUserId(),
bucket);
            if (MessageDigest.isEqual(expected, generated.getBytes(StandardCharsets.UTF_8))) {
                match = candidate.studentUserId();
                matches++;
            }
        }
        return matches == 1 ? new Resolution(match, true) : Resolution.none();
    }
    private boolean hasSignal(Long studentId,
        Map<Long, TalentDiscoveryCandidateSource.PreferenceSignal> preferences,
        Map<Long, TalentDiscoveryCandidateSource.ScoreSignal> riasec,

```

```

        Map<Long, TalentDiscoveryCandidateSource.ScoreSignal> ability) {
    TalentDiscoveryCandidateSource.PreferenceSignal p = preferences.get(studentId);
    if (p != null && (p.expectedSalaryMin() != null || p.expectedSalaryMax() != null
        || !p.cities().isEmpty() || !p.employmentTypes().isEmpty()
        || !p.industries().isEmpty() || !p.jobTypes().isEmpty())) return true;
    TalentDiscoveryCandidateSource.ScoreSignal i = riasec.get(studentId);
    if (i != null && i.scores() != null && !i.scores().isEmpty()) return true;
    TalentDiscoveryCandidateSource.ScoreSignal a = ability.get(studentId);
    return a != null && a.scores() != null && !a.scores().isEmpty();
}
private <K, V> Map<K, V> safeMap(Map<K, V> value) {
    return value == null ? Map.of() : value;
}
public record Resolution(Long studentUserId, boolean matched) {
    public static Resolution none() { return new Resolution(null, false); }
}
}
package com.mydreamjob.talentinvitation;
import com.mydreamjob.audit.AuditContext;
import com.mydreamjob.common.api.ApiResponse;
import com.mydreamjob.common.exception.BusinessException;
import com.mydreamjob.common.exception.ErrorCode;
import com.mydreamjob.talentinvitation.dto.*;
import io.swagger.v3.oas.annotations.Operation;
import io.swagger.v3.oas.annotations.media.Content;
import io.swagger.v3.oas.annotations.media.Schema;
import io.swagger.v3.oas.annotations.responses.ApiResponses;
import io.swagger.v3.oas.annotations.security.SecurityRequirement;
import io.swagger.v3.oas.annotations.tags.Tag;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.validation.Valid;
import lombok.RequiredArgsConstructor;
import org.springframework.web.bind.annotation.*;
@RestController
@RequiredArgsConstructor
@Tag(name = "企业人才邀约", description = "企业发起邀约，学生同意后建立站内会话")
public class TalentInvitationController {
    /** R01-A4: 身份只来自安全上下文，不接受请求参数传入的 userId。 */
    private final com.mydreamjob.security.CurrentUserAccessor currentUserAccessor;
    /** R01-A4: requestId 只来自过滤器规范化后的唯一产出点。 */
    private final com.mydreamjob.common.logging.RequestIdAccessor requestIdAccessor;
    private final TalentInvitationService service;
    @PostMapping("/companies/me/jobs/{jobId}/talent-invitations")
    @Operation(summary = "创建人才邀约", security = @SecurityRequirement(name = "bearerAuth"))
    @ApiResponses({@io.swagger.v3.oas.annotations.responses.ApiResponse(responseCode = "200", content =
        @Content(schema = @Schema(ref = "#/components/schemas/ApiResponseCompanyTalentInvitationResponse"))))
    public ApiResponse<CompanyTalentInvitationResponse> create(@PathVariable Long jobId,
        @Valid @RequestBody
        CreateTalentInvitationRequest
        body,
        HttpServletRequest request) {
        return ApiResponse.ok(service.create(currentUserAccessor.requireUserId(), jobId, body,
            AuditContext.from(request)), requestIdAccessor.get(request));
    }
    @GetMapping("/companies/me/talent-invitations")
    @Operation(summary = "企业邀约列表", security = @SecurityRequirement(name = "bearerAuth"))
    @ApiResponses({@io.swagger.v3.oas.annotations.responses.ApiResponse(responseCode = "200", content =
        @Content(schema = @Schema(ref = "#/components/schemas/ApiResponseCompanyTalentInvitationPageResponse"))))
    public ApiResponse<TalentInvitationPageResponse<CompanyTalentInvitationResponse>> listCompany(
        @RequestParam(defaultValue = "0") int page, @RequestParam(defaultValue = "20") int size,
        HttpServletRequest request) {
        return ApiResponse.ok(service.listCompany(currentUserAccessor.requireUserId(), page, size,
            requestIdAccessor.get(request));
    }
    @GetMapping("/companies/me/talent-invitations/{invitationId}")
    @Operation(summary = "企业邀约详情", security = @SecurityRequirement(name = "bearerAuth"))
    @ApiResponses({@io.swagger.v3.oas.annotations.responses.ApiResponse(responseCode = "200", content =
        @Content(schema = @Schema(ref = "#/components/schemas/ApiResponseCompanyTalentInvitationResponse"))))
    public ApiResponse<CompanyTalentInvitationResponse> getCompany(@PathVariable Long invitationId,
        HttpServletRequest request) {
        return ApiResponse.ok(service.getCompany(currentUserAccessor.requireUserId(), invitationId,
            AuditContext.from(request)), requestIdAccessor.get(request));
    }
}

```

```

    }
    @GetMapping("/profile/me/talent-invitations")
    @Operation(summary = "学生收到的邀约列表", security = @SecurityRequirement(name = "bearerAuth"))
    @ApiResponse(@io.swagger.v3.oas.annotations.responses.ApiResponse(responseCode = "200", content =
    @Content(schema = @Schema(ref = "#/components/schemas/ApiResponseStudentTalentInvitationPageResponse"))))
    public ApiResponse<TalentInvitationPageResponse<StudentTalentInvitationResponse>> listStudent(
        @RequestParam(defaultValue = "0") int page, @RequestParam(defaultValue = "20") int size,
        HttpServletRequest request) {
        return ApiResponse.ok(service.listStudent(currentUserAccessor.requireUserId(), page, size),
        requestIdAccessor.get(request));
    }
    @GetMapping("/profile/me/talent-invitations/{invitationId}")
    @Operation(summary = "学生邀约详情", security = @SecurityRequirement(name = "bearerAuth"))
    @ApiResponse(@io.swagger.v3.oas.annotations.responses.ApiResponse(responseCode = "200", content =
    @Content(schema = @Schema(ref = "#/components/schemas/ApiResponseStudentTalentInvitationResponse"))))
    public ApiResponse<StudentTalentInvitationResponse> getStudent(@PathVariable Long invitationId,
        HttpServletRequest request) {
        return ApiResponse.ok(service.getStudent(currentUserAccessor.requireUserId(), invitationId,
        AuditContext.from(request)), requestIdAccessor.get(request));
    }
    @PostMapping("/profile/me/talent-invitations/{invitationId}/decision")
    @Operation(summary = "处理人才邀约", description = "学生接受后才创建 TALENT_INVITATION 站内会话", security
    = @SecurityRequirement(name =
    "bearerAuth"))
    @ApiResponse(@io.swagger.v3.oas.annotations.responses.ApiResponse(responseCode = "200", content =
    @Content(schema = @Schema(ref = "#/components/schemas/ApiResponseStudentTalentInvitationResponse"))))
    public ApiResponse<StudentTalentInvitationResponse> decide(@PathVariable Long invitationId,
        @Valid @RequestBody
        TalentInvitationDecisionRequest
        body,
        HttpServletRequest request) {
        return ApiResponse.ok(service.decide(currentUserAccessor.requireUserId(), invitationId, body,
        AuditContext.from(request)), requestIdAccessor.get(request));
    }
}
package com.mydreamjob.talentinvitation;
import com.mydreamjob.common.exception.BusinessException;
import com.mydreamjob.common.exception.ErrorCode;
import com.mydreamjob.job.Job;
import com.mydreamjob.messaging.Conversation;
import com.mydreamjob.messaging.ConversationMember;
import com.mydreamjob.messaging.ConversationMemberRepository;
import com.mydreamjob.messaging.ConversationRepository;
import com.mydreamjob.safety.BlockRelationRepository;
import lombok.RequiredArgsConstructor;
import org.springframework.dao.DataIntegrityViolationException;
import org.springframework.stereotype.Component;
import org.springframework.transaction.annotation.Transactional;
import java.util.List;
/** 仅供已接受邀约调用的会话创建器；不会放宽现有 DIRECT 会话规则。 */
@Component
@RequiredArgsConstructor
public class TalentInvitationConversationService {
    private final ConversationRepository conversationRepository;
    private final ConversationMemberRepository memberRepository;
    private final BlockRelationRepository blockRelationRepository;
    @Transactional
    public Long createForAcceptedInvitation(TalentInvitation invitation, Long companyOwnerUserId, Job job) {
        if (invitation.getConversationId() != null) return invitation.getConversationId();
        if (blockRelationRepository.existsByBlockerUserIdAndBlockedUserId(companyOwnerUserId,
        invitation.getStudentUserId())
        || blockRelationRepository.existsByBlockerUserIdAndBlockedUserId(invitation.getStudentUserId(
        ), companyOwnerUserId))
        {
            throw new BusinessException(ErrorCode.FORBIDDEN);
        }
        Conversation conversation = conversationRepository.saveAndFlush(Conversation.builder()
            .createdBy(companyOwnerUserId)
            .title(job == null ? "人才邀约沟通" : job.getTitle())
            .type("TALENT_INVITATION")
            .build());
        try {

```

```

        memberRepository.saveAll(List.of(
            ConversationMember.builder().conversationId(conversation.getId()).userId(companyOwnerUser
Id).build(),
            ConversationMember.builder().conversationId(conversation.getId()).userId(invitation.getSt
udentUserId()).build()));
        memberRepository.flush();
    } catch (DataIntegrityViolationException ex) {
        // invitation 行由上游锁定; 若数据库异常, 事务回滚, 调用方不会看到半成品会话。
        throw ex;
    }
    return conversation.getId();
}
}
package com.mydreamjob.talentinvitation;
import com.mydreamjob.audit.AuditContext;
import com.mydreamjob.audit.AuditService;
import lombok.RequiredArgsConstructor;
import org.springframework.stereotype.Component;
import org.springframework.transaction.annotation.Propagation;
import org.springframework.transaction.annotation.Transactional;
import java.time.LocalDateTime;
import java.util.LinkedHashMap;
import java.util.Map;
/** 把 EXPIRED 状态、历史和审计提交在独立事务中, 避免随后返回错误导致回滚。 */
@Component
@RequiredArgsConstructor
public class TalentInvitationExpiryService {
    private final TalentInvitationRepository invitationRepository;
    private final TalentInvitationHistoryRepository historyRepository;
    private final AuditService auditService;
    @Transactional(propagation = Propagation.REQUIRES_NEW)
    public boolean expireIfDue(Long invitationId, LocalDateTime now, AuditContext context) {
        if (invitationId == null || now == null) return false;
        TalentInvitation current = invitationRepository.findById(invitationId).orElse(null);
        if (current == null || !TalentInvitation.PENDING.equals(current.getStatus()) ||
current.getExpiresAt().isAfter(now))
        {
            return false;
        }
        if (invitationRepository.expire(invitationId, now) != 1) return false;
        historyRepository.save(TalentInvitationHistory.builder()
            .invitationId(invitationId).action(TalentInvitationHistory.EXPIRED).actorUserId(null).build()
        );
        Map<String, Object> details = new LinkedHashMap<>();
        details.put("invitationId", invitationId);
        details.put("jobId", current.getJobId());
        details.put("status", TalentInvitation.EXPIRED);
        details.put("requestId", context == null ? null : context.getRequestId());
        auditService.log(null, "TALENT_INVITATION_EXPIRED", "TALENT_INVITATION", invitationId,
            toJson(details), requestContext(context));
        return true;
    }
    private AuditContext requestContext(AuditContext context) {
        return context == null ? null : AuditContext.builder().requestId(context.getRequestId()).build();
    }
    private String toJson(Map<String, Object> values) {
        return "{\"invitationId\":\"" + values.get("invitationId") + "\",\"jobId\":\"" + values.get("jobId")
            + "\",\"status\":\"EXPIRED\",\"requestId\":\""
            + (values.get("requestId") == null ? "null" : quote(String.valueOf(values.get("requestId"))))
+ "\"}";
    }
    private String quote(String value) {
        return "\"" + value.replace("\\", "\\\").replace("\"", "\\\"") + "\"";
    }
}
package com.mydreamjob.talentinvitation;
import jakarta.persistence.*;
import lombok.*;
import org.springframework.data.annotation.CreatedDate;
import org.springframework.data.jpa.domain.support.AuditingEntityListener;
import java.time.LocalDateTime;
@Data

```

```

@Builder
@NoArgsConstructor
@AllArgsConstructor
@Entity
@Table(name = "talent_invitation_history")
@EntityListeners(AuditingEntityListener.class)
public class TalentInvitationHistory {
    public static final String CREATED = "CREATED";
    public static final String ACCEPTED = "ACCEPTED";
    public static final String REJECTED = "REJECTED";
    public static final String EXPIRED = "EXPIRED";
    @Id @GeneratedValue(strategy = GenerationType.IDENTITY) private Long id;
    @Column(name = "invitation_id", nullable = false) private Long invitationId;
    @Column(nullable = false, length = 30) private String action;
    @Column(name = "actor_user_id") private Long actorUserId;
    @CreatedDate @Column(name = "created_at", nullable = false, updatable = false) private LocalDateTime
    createdAt;
}
package com.mydreamjob.talentinvitation;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;
@Repository
public interface TalentInvitationHistoryRepository extends JpaRepository<TalentInvitationHistory, Long> {
    long countByInvitationId(Long invitationId);
}
package com.mydreamjob.talentinvitation;
import jakarta.persistence.LockModeType;
import org.springframework.data.domain.Page;
import org.springframework.data.domain.Pageable;
import org.springframework.data.jpa.repository.*;
import org.springframework.data.repository.query.Param;
import org.springframework.stereotype.Repository;
import java.time.LocalDateTime;
import java.util.Optional;
@Repository
public interface TalentInvitationRepository extends JpaRepository<TalentInvitation, Long> {
    Optional<TalentInvitation> findByCompanyIdAndIdempotencyKey(Long companyId, String idempotencyKey);
    Optional<TalentInvitation> findByIdAndCompanyId(Long id, Long companyId);
    Optional<TalentInvitation> findByIdAndStudentUserId(Long id, Long studentUserId);
    Optional<TalentInvitation> findByCompanyIdAndJobIdAndStudentUserIdAndStatus(
        Long companyId, Long jobId, Long studentUserId, String status);
    Page<TalentInvitation> findByCompanyIdOrderByCreatedAtDescIdDesc(Long companyId, Pageable pageable);
    Page<TalentInvitation> findByStudentUserIdOrderByCreatedAtDescIdDesc(Long studentUserId, Pageable
    pageable);
    @Query("select count(t) from TalentInvitation t where t.companyId=:companyId and t.createdAt>=:from and
    t.createdAt<=:to")
    long countCreatedBetween(@Param("companyId") Long companyId, @Param("from") LocalDateTime from,
        @Param("to") LocalDateTime to);
    @Query("select count(t) from TalentInvitation t where t.companyId=:companyId and t.jobId=:jobId "
        + "and t.createdAt>=:from and t.createdAt<=:to")
    long countJobCreatedBetween(@Param("companyId") Long companyId, @Param("jobId") Long jobId,
        @Param("from") LocalDateTime from, @Param("to") LocalDateTime to);
    @Query("select case when count(t)>0 then true else false end from TalentInvitation t "
        + "where t.companyId=:companyId and t.studentUserId=:studentUserId and t.expiresAt>:since")
    boolean existsInCooldown(@Param("companyId") Long companyId, @Param("studentUserId") Long studentUserId,
        @Param("since") LocalDateTime since);
    @Modifying(clearAutomatically = true, flushAutomatically = true)
    @Query("update TalentInvitation t set t.status=:status, t.activeKey=null, t.decidedAt=:decidedAt, "
        + "t.version=t.version+1 where t.id=:id and t.studentUserId=:studentUserId "
        + "and t.status='PENDING' and t.version=:version and t.expiresAt>:now")
    int decide(@Param("id") Long id, @Param("studentUserId") Long studentUserId,
        @Param("status") String status, @Param("version") Long version,
        @Param("decidedAt") LocalDateTime decidedAt, @Param("now") LocalDateTime now);
    @Modifying(clearAutomatically = true, flushAutomatically = true)
    @Query("update TalentInvitation t set t.conversationId=:conversationId "
        + "where t.id=:id and t.studentUserId=:studentUserId and t.status='ACCEPTED' "
        + "and t.version=:version and t.conversationId is null")
    int attachConversation(@Param("id") Long id, @Param("studentUserId") Long studentUserId,
        @Param("conversationId") Long conversationId, @Param("version") Long version);
    @Modifying(clearAutomatically = true, flushAutomatically = true)
    @Query("update TalentInvitation t set t.status='EXPIRED', t.activeKey=null, t.decidedAt=:now, "
        + "t.version=t.version+1 where t.id=:id and t.status='PENDING' and t.expiresAt<=:now")

```

```

    int expire(@Param("id") Long id, @Param("now") LocalDateTime now);
}
package com.mydreamjob.talentinvitation;
import com.fasterxml.jackson.core.JsonProcessingException;
import com.fasterxml.jackson.databind.ObjectMapper;
import com.mydreamjob.audit.AuditContext;
import com.mydreamjob.audit.AuditService;
import com.mydreamjob.audit.SecurityAuditService;
import com.mydreamjob.auth.User;
import com.mydreamjob.auth.UserRepository;
import com.mydreamjob.common.exception.BusinessException;
import com.mydreamjob.common.exception.ErrorCode;
import com.mydreamjob.company.Company;
import com.mydreamjob.company.CompanyRepository;
import com.mydreamjob.job.Job;
import com.mydreamjob.job.JobRepository;
import com.mydreamjob.recommendation.JobRecommendationProfile;
import com.mydreamjob.recommendation.JobRecommendationProfileRepository;
import com.mydreamjob.review.ReviewModerationService;
import com.mydreamjob.safety.BlockRelationRepository;
import com.mydreamjob.talentinvitation.dto.CompanyTalentInvitationResponse;
import com.mydreamjob.talentinvitation.dto.CreateTalentInvitationRequest;
import com.mydreamjob.talentinvitation.dto.StudentTalentInvitationResponse;
import com.mydreamjob.talentinvitation.dto.TalentInvitationDecisionRequest;
import com.mydreamjob.talentinvitation.dto.TalentInvitationPageResponse;
import lombok.RequiredArgsConstructor;
import org.springframework.dao.DataIntegrityViolationException;
import org.springframework.orm.ObjectOptimisticLockingFailureException;
import org.springframework.data.domain.Page;
import org.springframework.data.domain.PageRequest;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;
import java.nio.charset.StandardCharsets;
import java.security.MessageDigest;
import java.time.LocalDate;
import java.time.LocalDateTime;
import java.time.ZoneId;
import java.util.*;
/** V42 邀约状态机、归属校验、幂等和接受后会话编排。 */
@Service
@RequiredArgsConstructor
public class TalentInvitationService {
    private static final ZoneId BUSINESS_ZONE = ZoneId.of("Asia/Shanghai");
    private static final int TTL_DAYS = 7;
    private static final int COOL_DOWN_DAYS = 30;
    private static final int DAILY_COMPANY_LIMIT = 100;
    private static final int DAILY_JOB_LIMIT = 20;
    private static final ObjectMapper JSON = new ObjectMapper();
    private final com.mydreamjob.security.UserRoleGuard userRoleGuard;
    private final TalentInvitationRepository invitationRepository;
    private final TalentInvitationHistoryRepository historyRepository;
    private final UserRepository userRepository;
    private final CompanyRepository companyRepository;
    private final JobRepository jobRepository;
    private final JobRecommendationProfileRepository profileRepository;
    private final TalentInvitationCandidateResolver candidateResolver;
    private final TalentInvitationConversationService conversationService;
    private final TalentInvitationExpiryService expiryService;
    private final BlockRelationRepository blockRelationRepository;
    private final ReviewModerationService moderationService;
    private final AuditService auditService;
    private final SecurityAuditService securityAuditService;
    @Transactional
    public CompanyTalentInvitationResponse create(Long userId, Long jobId, CreateTalentInvitationRequest request,
                                                AuditContext context) {
        if (jobId == null || jobId <= 0) throw new BusinessException(ErrorCode.BAD_REQUEST);
        Company company = requireVerifiedCompany(userId);
        CreateValues values = validateCreateRequest(request);
        // 公司行锁把幂等、冷却和每日限额串在同一锁顺序内; 避免并发 COUNT 后同时 INSERT。
        companyRepository.findByIdForUpdate(company.getId()).orElseThrow(() -> new
        BusinessException(ErrorCode.COMPANY_NOT_FOUND));

```

```

    Job job = requireOwnedJob(company, jobId);
    String digest = sha256(values.candidateKey());
    TalentInvitation existingByKey = invitationRepository.findByCompanyIdAndIdempotencyKey(company.getId(
), values.idempotencyKey()).orElse(null);
    if (existingByKey != null) {
        if (existingByKey.getJobId().equals(jobId)
            && digest.equals(existingByKey.getCandidateKeyDigest())
            && values.message().equals(existingByKey.getMessage())) {
            return toCompanyResponse(existingByKey, job);
        }
        throw new BusinessException(ErrorCode.TALENT_INVITATION_IDEMPOTENCY_CONFLICT);
    }
    if (!"PUBLISHED".equals(job.getStatus())) throw new BusinessException(ErrorCode.INVALID_JOB_STATUS_TRANSITION);
    JobRecommendationProfile profile = profileRepository.findByJobId(jobId).orElse(null);
    if (profile == null || !Boolean.TRUE.equals(profile.getEnabled())
        || !JobRecommendationProfile.APPROVED.equals(profile.getReviewStatus())) {
        throw new BusinessException(ErrorCode.TALENT_INVITATION_CANDIDATE_UNAVAILABLE);
    }
    LocalDateTime now = LocalDateTime.now(BUSINESS_ZONE);
    TalentInvitationCandidateResolver.Resolution resolution = candidateResolver.resolve(company.getId(),
jobId, values.candidateKey(),
now);
    if (!resolution.matched()) throw new BusinessException(ErrorCode.TALENT_INVITATION_CANDIDATE_UNAVAILABLE);
    if (invitationRepository.findByCompanyIdAndJobIdAndStudentUserIdAndStatus(
        company.getId(), jobId, resolution.studentUserId(), TalentInvitation.PENDING).isPresent()) {
        throw new BusinessException(ErrorCode.TALENT_INVITATION_ALREADY_ACTIVE);
    }
    LocalDateTime cooldownSince = now.minusDays(COOL_DOWN_DAYS);
    if (invitationRepository.existsInCooldown(company.getId(), resolution.studentUserId(),
cooldownSince))
    {
        throw new BusinessException(ErrorCode.TALENT_INVITATION_ALREADY_ACTIVE);
    }
    LocalDateTime dayStart = now.toLocalDate().atStartOfDay();
    LocalDateTime nextDay = dayStart.plusDays(1);
    if (invitationRepository.countJobCreatedBetween(company.getId(), jobId, dayStart, nextDay) >=
DAILY_JOB_LIMIT
        || invitationRepository.countCreatedBetween(company.getId(), dayStart, nextDay) >=
DAILY_COMPANY_LIMIT)
    {
        throw new BusinessException(ErrorCode.TALENT_INVITATION_RATE_LIMITED);
    }
    ReviewModerationService.ModerationResult moderation = moderationService.moderateCompanyReply(values.m
essage());
    if (!moderation.getReasonCodes().isEmpty()) {
        auditRejected(userId, company.getId(), jobId, context);
        throw new BusinessException(ErrorCode.TALENT_INVITATION_CONTENT_INVALID);
    }
    TalentInvitation invitation = TalentInvitation.builder()
        .companyId(company.getId()).jobId(jobId).studentUserId(resolution.studentUserId())
        .candidateKeyDigest(digest).message(values.message()).idempotencyKey(values.idempotencyKey())
        .activeKey(company.getId() + ":" + jobId + ":" + resolution.studentUserId())
        .status(TalentInvitation.PENDING).expiresAt(now.plusDays(TTL_DAYS)).version(0L).build();
    try {
        invitation = invitationRepository.saveAndFlush(invitation);
    } catch (DataIntegrityViolationException ex) {
        // 公司行锁正常情况下已串行；唯一约束是异常路径的最终防线，统一转换为业务冲突。
        throw new BusinessException(ErrorCode.TALENT_INVITATION_ALREADY_ACTIVE);
    }
    historyRepository.save(TalentInvitationHistory.builder().invitationId(invitation.getId())
        .action(TalentInvitationHistory.CREATED).actorUserId(userId).build());
    audit(userId, "TALENT_INVITATION_CREATED", invitation, context);
    return toCompanyResponse(invitation, job);
}
@Transactional(readOnly = true)
public TalentInvitationPageResponse<CompanyTalentInvitationResponse> listCompany(Long userId, int page,
int size)
{
    validatePage(page, size);
    Company company = requireVerifiedCompany(userId);

```

```

Page<TalentInvitation> invitations = invitationRepository.findByCompanyIdOrderByCreatedAtDescIdDesc(
    company.getId(), PageRequest.of(page, size));
List<Long> jobIds = invitations.getContent().stream().map(TalentInvitation::getJobId).distinct().toLi
st();
Map<Long, Job> jobs = new HashMap<>();
jobRepository.findAllById(jobIds).forEach(job -> jobs.put(job.getId(), job));
List<CompanyTalentInvitationResponse> items = invitations.getContent().stream()
    .map(inv -> toCompanyResponse(inv, jobs.get(inv.getJobId()))).toList();
return page(items, invitations);
}
@Transactional(readOnly = true)
public CompanyTalentInvitationResponse getCompany(Long userId, Long invitationId, AuditContext context) {
    Company company = requireVerifiedCompany(userId);
    TalentInvitation invitation = invitationRepository.findById(invitationId)
        .orElseThrow(() -> new BusinessException(ErrorCode.TALENT_INVITATION_NOT_FOUND));
    if (!company.getId().equals(invitation.getCompanyId())) {
        // 跨企业邀约统一按越权处理，不返回岗位/候选等任何业务字段。
        throw new BusinessException(ErrorCode.FORBIDDEN);
    }
    if (expireIfDue(invitation, context)) {
        throw new BusinessException(ErrorCode.TALENT_INVITATION_EXPIRED);
    }
    Job job = jobRepository.findById(invitation.getJobId())
        .orElseThrow(() -> new BusinessException(ErrorCode.TALENT_INVITATION_NOT_FOUND));
    return toCompanyResponse(invitationRepository.findById(invitationId).orElse(invitation), job);
}
@Transactional(readOnly = true)
public TalentInvitationPageResponse<StudentTalentInvitationResponse> listStudent(Long userId, int page,
int size)
{
    userRoleGuard.requireStudent(userId);
    validatePage(page, size);
    Page<TalentInvitation> invitations = invitationRepository.findByStudentUserIdOrderByCreatedAtDescIdDe
sc(
        userId, PageRequest.of(page, size));
    List<Long> jobIds = invitations.getContent().stream().map(TalentInvitation::getJobId).distinct().toLi
st();
    List<Long> companyIds = invitations.getContent().stream().map(TalentInvitation::getCompanyId).disting
t().toList();
    Map<Long, Job> jobs = new HashMap<>();
    Map<Long, Company> companies = new HashMap<>();
    jobRepository.findAllById(jobIds).forEach(job -> jobs.put(job.getId(), job));
    companyRepository.findAllById(companyIds).forEach(company -> companies.put(company.getId(),
company));
    List<StudentTalentInvitationResponse> items = invitations.getContent().stream()
        .map(inv -> toStudentResponse(inv, jobs.get(inv.getJobId()),
companies.get(inv.getCompanyId()))).toList();
    return pageStudent(items, invitations);
}
@Transactional
public StudentTalentInvitationResponse getStudent(Long userId, Long invitationId, AuditContext context) {
    userRoleGuard.requireStudent(userId);
    TalentInvitation invitation = invitationRepository.findByIdAndStudentUserId(invitationId, userId)
        .orElseThrow(() -> new BusinessException(ErrorCode.TALENT_INVITATION_NOT_FOUND));
    if (expireIfDue(invitation, context)) {
        throw new BusinessException(ErrorCode.TALENT_INVITATION_EXPIRED);
    }
    TalentInvitation current = invitationRepository.findById(invitationId).orElse(invitation);
    Job job = jobRepository.findById(current.getJobId())
        .orElseThrow(() -> new BusinessException(ErrorCode.TALENT_INVITATION_NOT_FOUND));
    Company company = companyRepository.findById(current.getCompanyId())
        .orElseThrow(() -> new BusinessException(ErrorCode.TALENT_INVITATION_NOT_FOUND));
    return toStudentResponse(current, job, company);
}
@Transactional
public StudentTalentInvitationResponse decide(Long userId, Long invitationId,
TalentInvitationDecisionRequest
request,
        AuditContext context) {
    userRoleGuard.requireStudent(userId);
    if (request == null || request.getDecision() == null || request.getExpectedVersion() == null
        || !("ACCEPT".equals(request.getDecision()) || "REJECT".equals(request.getDecision())) {

```

```

        throw new BusinessException(ErrorCode.TALENT_INVITATION_CONTENT_INVALID);
    }
    TalentInvitation invitation = invitationRepository.findByIdAndStudentUserId(invitationId, userId)
        .orElseThrow(() -> new BusinessException(ErrorCode.TALENT_INVITATION_NOT_FOUND));
    LocalDateTime now = LocalDateTime.now(BUSINESS_ZONE);
    if (TalentInvitation.PENDING.equals(invitation.getStatus()) &&
!invitation.getExpiresAt().isAfter(now))
    {
        if (expiryService.expireIfDue(invitationId, now, context)) {
            throw new BusinessException(ErrorCode.TALENT_INVITATION_EXPIRED);
        }
    }
    TalentInvitation current = invitationRepository.findByIdAndStudentUserId(invitationId, userId)
        .orElseThrow(() -> new BusinessException(ErrorCode.TALENT_INVITATION_NOT_FOUND));
    String target = "ACCEPT".equals(request.getDecision()) ? TalentInvitation.ACCEPTED :
TalentInvitation.REJECTED;
    if (!TalentInvitation.PENDING.equals(current.getStatus())) {
        if (target.equals(current.getStatus())) return studentResponse(current);
        if (TalentInvitation.EXPIRED.equals(current.getStatus())) throw new
BusinessException(ErrorCode.TALENT_INVITATION_EXPIRED);
        throw new BusinessException(ErrorCode.TALENT_INVITATION_INVALID_STATE);
    }
    if (TalentInvitation.ACCEPTED.equals(target) && isBlocked(current)) {
        throw new BusinessException(ErrorCode.FORBIDDEN);
    }
    int updated;
    try {
        updated = invitationRepository.decide(invitationId, userId, target,
request.getExpectedVersion(), now,
now);
    } catch (ObjectOptimisticLockingFailureException ex) {
        throw new BusinessException(ErrorCode.TALENT_INVITATION_CONCURRENT_UPDATE);
    }
    if (updated != 1) {
        TalentInvitation after = invitationRepository.findByIdAndStudentUserId(invitationId,
userId).orElse(null);
        if (after != null && target.equals(after.getStatus())) return studentResponse(after);
        if (after != null && TalentInvitation.EXPIRED.equals(after.getStatus())) throw new
BusinessException(ErrorCode.TALENT_INVITATION_EXPIRED);
        throw new BusinessException(ErrorCode.TALENT_INVITATION_CONCURRENT_UPDATE);
    }
    TalentInvitation saved = invitationRepository.findByIdAndStudentUserId(invitationId, userId)
        .orElseThrow(() -> new BusinessException(ErrorCode.TALENT_INVITATION_NOT_FOUND));
    Job job = jobRepository.findById(saved.getJobId()).orElseThrow(() -> new
BusinessException(ErrorCode.TALENT_INVITATION_NOT_FOUND));
    Company company = companyRepository.findById(saved.getCompanyId()).orElseThrow(() -> new
BusinessException(ErrorCode.TALENT_INVITATION_NOT_FOUND));
    if (TalentInvitation.ACCEPTED.equals(target)) {
        Long conversationId = conversationService.createForAcceptedInvitation(saved,
company.getOwnerUserId(),
job);
        if (invitationRepository.attachConversation(saved.getId(), userId, conversationId,
saved.getVersion()) != 1)
        {
            throw new BusinessException(ErrorCode.TALENT_INVITATION_CONCURRENT_UPDATE);
        }
        saved = invitationRepository.findByIdAndStudentUserId(saved.getId(), userId)
            .orElseThrow(() -> new BusinessException(ErrorCode.TALENT_INVITATION_NOT_FOUND));
    }
    historyRepository.save(TalentInvitationHistory.builder().invitationId(saved.getId()).action(
        TalentInvitation.ACCEPTED.equals(target) ? TalentInvitationHistory.ACCEPTED :
TalentInvitationHistory.REJECTED)
        .actorUserId(userId).build());
    audit(userId, TalentInvitation.ACCEPTED.equals(target) ? "TALENT_INVITATION_ACCEPTED" :
"TALENT_INVITATION_REJECTED", saved,
context);
    return toStudentResponse(saved, job, company);
}
private boolean expireIfDue(TalentInvitation invitation, AuditContext context) {
    LocalDateTime now = LocalDateTime.now(BUSINESS_ZONE);
    return TalentInvitation.PENDING.equals(invitation.getStatus()) &&
!invitation.getExpiresAt().isAfter(now)

```

```

        && expiryService.expireIfDue(invitation.getId(), now, context);
    }
    private boolean isBlocked(TalentInvitation invitation) {
        return companyRepository.findById(invitation.getCompanyId()).map(company ->
            blockRelationRepository.existsByBlockerUserIdAndBlockedUserId(company.getOwnerUserId(),
            invitation.getStudentUserId())
            || blockRelationRepository.existsByBlockerUserIdAndBlockedUserId(invitation.getStudentUserId(), company.getOwnerUserId())
            ).orElse(false);
    }
    private TalentInvitationPageResponse<CompanyTalentInvitationResponse>
    page(List<CompanyTalentInvitationResponse> items, Page<?> p)
    {
        return TalentInvitationPageResponse.<CompanyTalentInvitationResponse>builder().items(items).total(p.getTotalElements())
            .page(p.getNumber()).size(p.getSize()).totalPages(p.getTotalPages()).build();
    }
    private TalentInvitationPageResponse<StudentTalentInvitationResponse>
    pageStudent(List<StudentTalentInvitationResponse> items, Page<?> p)
    {
        return TalentInvitationPageResponse.<StudentTalentInvitationResponse>builder().items(items).total(p.getTotalElements())
            .page(p.getNumber()).size(p.getSize()).totalPages(p.getTotalPages()).build();
    }
    private CompanyTalentInvitationResponse toCompanyResponse(TalentInvitation inv, Job job) {
        return CompanyTalentInvitationResponse.builder().invitationId(inv.getId()).jobId(inv.getJobId())
            .jobTitle(job == null ? null : job.getTitle()).candidateLabel("开放人才候选人")
            .message(inv.getMessage()).status(inv.getStatus()).createdAt(inv.getCreatedAt())
            .expiresAt(inv.getExpiresAt()).decidedAt(inv.getDecidedAt())
            .conversationId(TalentInvitation.ACCEPTED.equals(inv.getStatus()) ? inv.getConversationId()
: null).build();
    }
    private StudentTalentInvitationResponse toStudentResponse(TalentInvitation inv, Job job, Company
company)
    {
        return StudentTalentInvitationResponse.builder().invitationId(inv.getId()).companyId(inv.getCompanyId()
())
            .companyName(company == null ? null : company.getName()).jobId(inv.getJobId())
            .jobTitle(job == null ? null : job.getTitle()).jobCity(job == null ? null : job.getCity())
            .message(inv.getMessage()).status(inv.getStatus()).version(inv.getVersion())
            .createdAt(inv.getCreatedAt()).expiresAt(inv.getExpiresAt()).decidedAt(inv.getDecidedAt())
            .conversationId(TalentInvitation.ACCEPTED.equals(inv.getStatus()) ? inv.getConversationId()
: null).build();
    }
    private StudentTalentInvitationResponse studentResponse(TalentInvitation inv) {
        Job job = jobRepository.findById(inv.getJobId()).orElse(null);
        Company company = companyRepository.findById(inv.getCompanyId()).orElse(null);
        return toStudentResponse(inv, job, company);
    }
    private Company requireVerifiedCompany(Long userId) {
        User user = requireUser(userId);
        if (!hasRole(user, User.Role.COMPANY)) throw new BusinessException(ErrorCode.FORBIDDEN);
        Company company = companyRepository.findByIdByOwnerUserId(userId).orElseThrow(() -> new
BusinessException(ErrorCode.COMPANY_NOT_FOUND));
        if (!Boolean.TRUE.equals(company.getVerified())) throw new BusinessException(ErrorCode.FORBIDDEN);
        return company;
    }
    private Job requireOwnedJob(Company company, Long jobId) {
        Job job = jobRepository.findById(jobId).orElseThrow(() -> new
BusinessException(ErrorCode.JOB_NOT_FOUND));
        if (!company.getId().equals(job.getCompanyId())) throw new BusinessException(ErrorCode.FORBIDDEN);
        return job;
    }
    private User requireUser(Long userId) {
        if (userId == null) throw new BusinessException(ErrorCode.UNAUTHORIZED);
        return userRepository.findById(userId).orElseThrow(() -> new
BusinessException(ErrorCode.USER_NOT_FOUND));
    }
    private boolean hasRole(User user, User.Role role) {
        return user != null && ((user.getRoles() != null && user.getRoles().contains(role)) ||
user.getRole() ==
role);
    }

```

```

    }
    private void validatePage(int page, int size) {
        if (page < 0 || size < 1 || size > 50) throw new BusinessException(ErrorCode.BAD_REQUEST);
    }
    private CreateValues validateCreateRequest(CreateTalentInvitationRequest request) {
        String key = request == null ? null : trim(request.getCandidateKey());
        String message = request == null ? null : trim(request.getMessage());
        String idempotency = request == null ? null : trim(request.getIdempotencyKey());
        if (key == null || key.length() > 128 || message == null || message.length() < 1 || message.length()
> 300
            || idempotency == null || idempotency.length() > 64) throw new
BusinessException(ErrorCode.TALENT_INVITATION_CONTENT_INVALID);
        try { UUID.fromString(idempotency); } catch (IllegalArgumentException ex) { throw new
BusinessException(ErrorCode.TALENT_INVITATION_CONTENT_INVALID);
    }
        return new CreateValues(key, message, idempotency);
    }
    private String trim(String value) { if (value == null) return null; String v = value.trim(); return
v.isEmpty() ? null : v;
    }
    private String sha256(String value) {
        try {
            byte[] bytes = MessageDigest.getInstance("SHA-256").digest(value.getBytes(StandardCharsets.UTF_8)
);
            StringBuilder out = new StringBuilder(64);
            for (byte b : bytes) out.append(String.format("%02x", b));
            return out.toString();
        } catch (Exception ex) { throw new IllegalStateException("SHA-256 unavailable", ex); }
    }
    private void audit(Long userId, String action, TalentInvitation invitation, AuditContext context) {
        auditService.log(userId, action, "TALENT_INVITATION", invitation.getId(), auditDetails(invitation,
context), requestOnly(context));
    }
    private void auditRejected(Long userId, Long companyId, Long jobId, AuditContext context) {
        securityAuditService.logRejected(userId, "TALENT_INVITATION_CREATE_REJECTED", "TALENT_INVITATION",
null,
            "{\"jobId\":\"" + jobId + "\",\"status\":\"REJECTED\",\"requestId\":\""
+ (context == null || context.getRequestId() == null ? "null" :
quote(context.getRequestId())) +
            "\"",
            requestOnly(context));
    }
    private String auditDetails(TalentInvitation invitation, AuditContext context) {
        return "{\"invitationId\":\"" + invitation.getId() + "\",\"jobId\":\"" + invitation.getJobId()
+ "\",\"status\":\"" + invitation.getStatus() + "\",\"requestId\":\""
+ (context == null || context.getRequestId() == null ? "null" :
quote(context.getRequestId())) +
        "\"";
    }
    private AuditContext requestOnly(AuditContext context) {
        return context == null ? null : AuditContext.builder().requestId(context.getRequestId()).build();
    }
    private String quote(String value) { return "\"" + value.replace("\\", "\\\\").replace("\"", "\\\"") +
        "\"";
    }
    private record CreateValues(String candidateKey, String message, String idempotencyKey) { }
}
package com.mydreamjob.talentinvitation.dto;
import lombok.Builder;
import lombok.Value;
import java.time.LocalDateTime;
@Value
@Builder
public class CompanyTalentInvitationResponse {
    Long invitationId;
    Long jobId;
    String jobTitle;
    String candidateLabel;
    String message;
    String status;
    LocalDateTime createdAt;
    LocalDateTime expiresAt;
}

```

```

        LocalDateTime decidedAt;
        Long conversationId;
    }
    package com.mydreamjob.talentinvitation.dto;
    import jakarta.validation.constraints.NotBlank;
    import jakarta.validation.constraints.Size;
    import lombok.Data;
    @Data
    public class CreateTalentInvitationRequest {
        @NotBlank @Size(max = 128) private String candidateKey;
        @NotBlank @Size(max = 300) private String message;
        @NotBlank @Size(max = 64) private String idempotencyKey;
    }
    package com.mydreamjob.talentinvitation.dto;
    import lombok.Builder;
    import lombok.Value;
    import java.time.LocalDateTime;
    @Value
    @Builder
    public class StudentTalentInvitationResponse {
        Long invitationId;
        Long companyId;
        String companyName;
        Long jobId;
        String jobTitle;
        String jobCity;
        String message;
        String status;
        Long version;
        LocalDateTime createdAt;
        LocalDateTime expiresAt;
        LocalDateTime decidedAt;
        Long conversationId;
    }
    package com.mydreamjob.talentinvitation.dto;
    import jakarta.validation.constraints.NotBlank;
    import jakarta.validation.constraints.NotNull;
    import jakarta.validation.constraints.Pattern;
    import lombok.Data;
    @Data
    public class TalentInvitationDecisionRequest {
        @NotBlank @Pattern(regexp = "^(ACCEPT|REJECT)$") private String decision;
        @NotNull private Long expectedVersion;
    }
    package com.mydreamjob.talentinvitation.dto;
    import lombok.Builder;
    import lombok.Value;
    import java.util.List;
    @Value
    @Builder
    public class TalentInvitationPageResponse<T> {
        List<T> items;
        long total;
        int page;
        int size;
        int totalPages;
    }
    package com.mydreamjob.verification;
    import jakarta.persistence.*;
    import lombok.AllArgsConstructor;
    import lombok.Builder;
    import lombok.Data;
    import lombok.NoArgsConstructor;
    import org.springframework.data.annotation.CreatedDate;
    import org.springframework.data.annotation.LastModifiedDate;
    import org.springframework.data.jpa.domain.support.AuditingEntityListener;
    import java.time.LocalDateTime;
    /**
     * 企业认证记录实体
     */
    @Data
    @Builder

```

```

@NoArgsConstructor
@AllArgsConstructor
@Entity
@Table(name = "company_verifications")
@EntityListeners(AuditingEntityListener.class)
public class CompanyVerification {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    @Column(name = "company_id", nullable = false)
    private Long companyId;
    @Column(name = "status", nullable = false, length = 20)
    @Builder.Default
    private String status = "PENDING";
    @Column(name = "submitted_by", nullable = false)
    private Long submittedBy;
    @Column(name = "materials_json", columnDefinition = "text")
    private String materialsJson;
    @Column(name = "reviewer_id")
    private Long reviewerId;
    @Column(name = "decision_reason", length = 500)
    private String decisionReason;
    @Column(name = "submitted_at", nullable = false)
    @Builder.Default
    private LocalDateTime submittedAt = LocalDateTime.now();
    @Column(name = "decided_at")
    private LocalDateTime decidedAt;
    @Version
    @Column(name = "version", nullable = false)
    @Builder.Default
    private Long version = 0L;
    @CreatedDate
    @Column(name = "created_at", nullable = false, updatable = false)
    private LocalDateTime createdAt;
    @LastModifiedDate
    @Column(name = "updated_at", nullable = false)
    private LocalDateTime updatedAt;
}

package com.mydreamjob.verification;
import com.mydreamjob.audit.AuditContext;
import com.mydreamjob.common.api.ApiResponse;
import com.mydreamjob.common.exception.BusinessException;
import com.mydreamjob.common.exception.ErrorCode;
import com.mydreamjob.verification.dto.VerificationDecisionRequest;
import com.mydreamjob.verification.dto.VerificationSubmitRequest;
import io.swagger.v3.oas.annotations.Operation;
import io.swagger.v3.oas.annotations.tags.Tag;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.validation.Valid;
import lombok.RequiredArgsConstructor;
import org.springframework.data.domain.Page;
import org.springframework.format.annotation.DateTimeFormat;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;
import java.time.LocalDateTime;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.stream.Collectors;
/**
 * 企业认证接口
 * 企业端：提交认证、查看认证状态
 * 管理端：审核列表、审核详情、审核决定、审核历史
 */
@Tag(name = "企业认证", description = "企业认证提交、审核管理")
@RestController
@RequiredArgsConstructor
public class CompanyVerificationController {
    /** R01-A3: 身份只来自安全上下文，不接受请求参数传入的 userId。 */
    private final com.mydreamjob.security.CurrentUserAccessor currentUserAccessor;
    /** R01-A3: requestId 只来自过滤器规范化后的唯一产出点。 */
    private final com.mydreamjob.common.logging.RequestIdAccessor requestIdAccessor;

```

```

private final CompanyVerificationService verificationService;
// — 企业: 提交认证 —————
@Operation(summary = "提交企业认证", description = "企业提交认证材料（营业执照等），只能提交自己的认证")
@PostMapping("/companies/me/verification")
public ResponseEntity<ApiResponse<Map<String, Object>>> submit(
    @Valid @RequestBody VerificationSubmitRequest req,
    HttpServletRequest request) {
    Long userId = currentUserAccessor.requireUserId();
    CompanyVerification verification = verificationService.submit(userId, req,
        AuditContext.from(request));
    return ResponseEntity.ok(ApiResponse.ok(verificationService.toView(verification),
requestIdAccessor.get(request)));
}
// — 企业: 查询认证状态 —————
@Operation(summary = "查询企业认证状态", description = "企业查询自己的认证状态和历史")
@GetMapping("/companies/me/verification")
public ResponseEntity<ApiResponse<Map<String, Object>>> getMyVerification(
    HttpServletRequest request) {
    Long userId = currentUserAccessor.requireUserId();
    CompanyVerification verification = verificationService.getMyVerification(userId);
    return ResponseEntity.ok(ApiResponse.ok(verificationService.toView(verification),
requestIdAccessor.get(request)));
}
// — 管理员: 待审核列表 —————
@Operation(summary = "待审核认证列表", description = "管理员查看认证申请列表，支持 status/from/to
筛选和分页")
@GetMapping("/admin/companies/verifications")
public ResponseEntity<ApiResponse<Map<String, Object>>> listVerifications(
    @RequestParam(required = false) String status,
    @RequestParam(required = false) @DateTimeFormat(iso = DateTimeFormat.ISO.DATE_TIME)
LocalDateTime
from,
    @RequestParam(required = false) @DateTimeFormat(iso = DateTimeFormat.ISO.DATE_TIME)
LocalDateTime
to,
    @RequestParam(defaultValue = "0") int page,
    @RequestParam(defaultValue = "20") int size,
    HttpServletRequest request) {
    Long userId = currentUserAccessor.requireUserId();
    verificationService.assertAdminRole(userId);
    Page<CompanyVerification> result = verificationService.listPending(status, from, to, page, size);
    Map<String, Object> data = new HashMap<>();
    data.put("items", result.getContent().stream().map(verificationService::toView).collect(Collectors.to
List()));
    data.put("total", result.getTotalElements());
    data.put("page", result.getNumber());
    data.put("size", result.getSize());
    data.put("totalPages", result.getTotalPages());
    return ResponseEntity.ok(ApiResponse.ok(data, requestIdAccessor.get(request)));
}
// — 管理员: 认证详情 —————
@Operation(summary = "认证详情", description = "管理员查看认证申请详情")
@GetMapping("/admin/companies/verifications/{id}")
public ResponseEntity<ApiResponse<Map<String, Object>>> getDetail(
    @PathVariable Long id,
    HttpServletRequest request) {
    Long userId = currentUserAccessor.requireUserId();
    verificationService.assertAdminRole(userId);
    CompanyVerification verification = verificationService.getDetail(id);
    return ResponseEntity.ok(ApiResponse.ok(verificationService.toView(verification),
requestIdAccessor.get(request)));
}
// — 管理员: 审核决定 —————
@Operation(summary = "审核决定", description = "管理员通过或拒绝认证申请，拒绝时必须填写原因")
@PostMapping("/admin/companies/verifications/{id}/decision")
public ResponseEntity<ApiResponse<Map<String, Object>>> decide(
    @PathVariable Long id,
    @Valid @RequestBody VerificationDecisionRequest req,
    HttpServletRequest request) {
    Long userId = currentUserAccessor.requireUserId();
    verificationService.assertAdminRole(userId);
    CompanyVerification verification = verificationService.decide(userId, id, req,

```

```

        AuditContext.from(request));
        return ResponseEntity.ok(ApiResponse.ok(verificationService.toView(verification),
requestIdAccessor.get(request)));
    }
    // — 管理员: 审核历史 —————
    @Operation(summary = "认证审核历史", description = "管理员查看某个认证申请的审核历史")
    @GetMapping("/admin/companies/verifications/history")
    public ResponseEntity<ApiResponse<List<Map<String, Object>>>> listHistory(
        @RequestParam Long verificationId,
        HttpServletRequest request) {
        Long userId = currentUserAccessor.requireUserId();
        verificationService.assertAdminRole(userId);
        List<Map<String, Object>> data = verificationService.getHistory(verificationId);
        return ResponseEntity.ok(ApiResponse.ok(data, requestIdAccessor.get(request)));
    }
    // — 辅助 —————
}

package com.mydreamjob.verification;
import jakarta.persistence.*;
import lombok.AllArgsConstructor;
import lombok.Builder;
import lombok.Data;
import lombok.NoArgsConstructor;
import java.time.LocalDateTime;
/**
 * 企业认证审核历史
 */
@Data
@Builder
@NoArgsConstructor
@AllArgsConstructor
@Entity
@Table(name = "company_verification_history")
public class CompanyVerificationHistory {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    @Column(name = "verification_id", nullable = false)
    private Long verificationId;
    @Column(name = "from_status", length = 20)
    private String fromStatus;
    @Column(name = "to_status", nullable = false, length = 20)
    private String toStatus;
    @Column(name = "actor_user_id", nullable = false)
    private Long actorUserId;
    @Column(name = "reason", length = 500)
    private String reason;
    @Column(name = "created_at", nullable = false, updatable = false)
    private LocalDateTime createdAt;
    @PrePersist
    void prePersist() {
        if (createdAt == null) createdAt = LocalDateTime.now();
    }
}

package com.mydreamjob.verification;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;
import java.util.List;
@Repository
public interface CompanyVerificationHistoryRepository extends JpaRepository<CompanyVerificationHistory,
Long> {
    List<CompanyVerificationHistory> findByVerificationIdOrderByCreatedAtAsc(Long verificationId);
}

package com.mydreamjob.verification;
import org.springframework.data.domain.Page;
import org.springframework.data.domain.Pageable;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;
import java.time.LocalDateTime;
import java.util.Optional;

```

```

@Repository
public interface CompanyVerificationRepository extends JpaRepository<CompanyVerification, Long> {
    Optional<CompanyVerification> findTopByCompanyIdOrderByIdDesc(Long companyId);
    Page<CompanyVerification> findByStatusOrderByCreatedAtDesc(String status, Pageable pageable);
    Page<CompanyVerification> findByStatusAndSubmittedAtBetweenOrderByCreatedAtDesc(
        String status, LocalDateTime from, LocalDateTime to, Pageable pageable);
    Page<CompanyVerification> findBySubmittedAtBetweenOrderByCreatedAtDesc(
        LocalDateTime from, LocalDateTime to, Pageable pageable);
    Page<CompanyVerification> findAllByOrderByCreatedAtDesc(Pageable pageable);
}

package com.mydreamjob.verification;
import com.mydreamjob.audit.AuditContext;
import com.mydreamjob.audit.AuditService;
import com.mydreamjob.auth.User;
import com.mydreamjob.auth.UserRepository;
import com.mydreamjob.common.exception.BusinessException;
import com.mydreamjob.common.exception.ErrorCode;
import com.mydreamjob.company.Company;
import com.mydreamjob.company.CompanyRepository;
import com.mydreamjob.notification.NotificationService;
import com.mydreamjob.verification.dto.VerificationDecisionRequest;
import com.mydreamjob.verification.dto.VerificationSubmitRequest;
import jakarta.persistence.EntityManager;
import jakarta.persistence.PersistenceContext;
import com.mydreamjob.security.UserRoleGuard;
import lombok.RequiredArgsConstructor;
import lombok.extern.slf4j.Slf4j;
import org.springframework.dao.OptimisticLockingFailureException;
import org.springframework.data.domain.Page;
import org.springframework.data.domain.PageRequest;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;
import java.time.LocalDateTime;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.stream.Collectors;

@Slf4j
@Service
@RequiredArgsConstructor
public class CompanyVerificationService {
    private static final int MAX_PAGE_SIZE = 100;
    private final com.mydreamjob.security.UserRoleGuard userRoleGuard;
    private final CompanyVerificationRepository verificationRepository;
    private final CompanyVerificationHistoryRepository historyRepository;
    private final CompanyRepository companyRepository;
    private final UserRepository userRepository;
    private final NotificationService notificationService;
    private final AuditService auditService;
    @PersistenceContext
    private EntityManager entityManager;
    // — 企业提交认证 —————
    @Transactional
    public CompanyVerification submit(Long companyUserId, VerificationSubmitRequest req, AuditContext ctx) {
        Company company = requireCompany(companyUserId);
        // 检查是否已有审批中的认证
        CompanyVerification existing = verificationRepository
            .findTopByCompanyIdOrderByIdDesc(company.getId()).orElse(null);
        if (existing != null && (existing.getStatus().equals("PENDING")
            || existing.getStatus().equals("RESUBMITTED"))) {
            throw new BusinessException(ErrorCode.VERIFICATION_ALREADY_PENDING);
        }
        if (existing != null && existing.getStatus().equals("APPROVED")) {
            throw new BusinessException(ErrorCode.VERIFICATION_ALREADY_APPROVED);
        }
        String targetStatus = (existing != null && existing.getStatus().equals("REJECTED"))
            ? VerificationStatus.RESUBMITTED.name()
            : VerificationStatus.PENDING.name();
        CompanyVerification verification = CompanyVerification.builder()
            .companyId(company.getId())
            .status(targetStatus)
            .submittedBy(companyUserId)

```

```

        .materialsJson(req.getMaterialsJson())
        .submittedAt(LocalDate.now())
        .build();
verification = verificationRepository.save(verification);
recordHistory(verification.getId(), null, targetStatus, companyUserId,
    "企业提交认证材料");
auditService.log(companyUserId, "SUBMIT_VERIFICATION", "VERIFICATION", verification.getId(),
    "Company verification submitted", ctx);
return verification;
}
// — 企业查询自己的认证状态 —————
public CompanyVerification getMyVerification(Long companyUserId) {
    Company company = requireCompany(companyUserId);
    return verificationRepository.findTopByCompanyIdOrderByIdDesc(company.getId())
        .orElseThrow(() -> new BusinessException(ErrorCode.VERIFICATION_NOT_FOUND));
}
// — 管理员: 待审核列表 —————
public Page<CompanyVerification> listPending(String status, LocalDateTime from, LocalDateTime to,
    int page, int size) {
    if (size > MAX_PAGE_SIZE) size = MAX_PAGE_SIZE;
    PageRequest pr = PageRequest.of(page, size);
    boolean hasStatus = status != null && !status.isBlank();
    boolean hasRange = from != null && to != null;
    if (hasStatus && hasRange)
        return verificationRepository.findByStatusAndSubmittedAtBetweenOrderByCreatedAtDesc(
            status, from, to, pr);
    if (hasStatus)
        return verificationRepository.findByStatusOrderByCreatedAtDesc(status, pr);
    if (hasRange)
        return verificationRepository.findBySubmittedAtBetweenOrderByCreatedAtDesc(from, to, pr);
    return verificationRepository.findAllByOrderByCreatedAtDesc(pr);
}
// — 管理员: 查看认证详情 —————
public CompanyVerification getDetail(Long verificationId) {
    return verificationRepository.findById(verificationId)
        .orElseThrow(() -> new BusinessException(ErrorCode.VERIFICATION_NOT_FOUND));
}
// — 管理员: 审核决定 —————
@Transactional
public CompanyVerification decide(Long adminUserId, Long verificationId,
    VerificationDecisionRequest req, AuditContext ctx) {
    CompanyVerification verification = verificationRepository.findById(verificationId)
        .orElseThrow(() -> new BusinessException(ErrorCode.VERIFICATION_NOT_FOUND));
    String decision = req.getDecision().toUpperCase();
    VerificationStatus targetStatus;
    try {
        targetStatus = VerificationStatus.valueOf(decision);
    } catch (IllegalArgumentException e) {
        throw new BusinessException(ErrorCode.BAD_REQUEST, "invalid decision: " + decision);
    }
    if (targetStatus != VerificationStatus.APPROVED && targetStatus != VerificationStatus.REJECTED) {
        throw new BusinessException(ErrorCode.BAD_REQUEST, "decision must be APPROVED or REJECTED");
    }
    VerificationStatus current = VerificationStatus.valueOf(verification.getStatus());
    if (current == VerificationStatus.APPROVED) {
        throw new BusinessException(ErrorCode.VERIFICATION_ALREADY_APPROVED);
    }
    // 幂等: 已相同状态无需重复
    if (current == targetStatus) {
        return verification;
    }
    if (current != VerificationStatus.PENDING && current != VerificationStatus.RESUBMITTED) {
        throw new BusinessException(ErrorCode.INVALID_VERIFICATION_STATUS_TRANSITION);
    }
    if (targetStatus == VerificationStatus.REJECTED
        && (req.getReason() == null || req.getReason().isBlank())) {
        throw new BusinessException(ErrorCode.VERIFICATION_REJECT_REASON_REQUIRED);
    }
    String fromStatus = verification.getStatus();
    try {
        verification.setStatus(targetStatus.name());
        verification.setReviewerId(adminUserId);

```

```

verification.setDecisionReason(req.getReason());
verification.setDecidedAt(LocalDate.now());
CompanyVerification saved = verificationRepository.saveAndFlush(verification);
recordHistory(saved.getId(), fromStatus, targetStatus.name(), adminUserId,
    req.getReason());
// 更新企业认证状态
if (targetStatus == VerificationStatus.APPROVED) {
    Company company = companyRepository.findById(saved.getCompanyId())
        .orElse(null);
    if (company != null && !Boolean.TRUE.equals(company.getVerified())) {
        company.setVerified(true);
        companyRepository.save(company);
    }
}
auditService.log(adminUserId, "DECIDE_VERIFICATION", "VERIFICATION", saved.getId(),
    "Verification " + targetStatus.name() + " for company " + saved.getCompanyId(), ctx);
notificationService.publishEvent(saved.getSubmittedBy(),
    "VERIFICATION_" + targetStatus.name(),
    "企业认证审核结果",
    "你的企业认证申请已被" + (targetStatus == VerificationStatus.APPROVED ? "通过" : "拒绝")
        + (req.getReason() != null ? ": " + req.getReason() : ""),
    "VERIFICATION", saved.getId(),
    "verification-decision-" + saved.getId());
return saved;
} catch (OptimisticLockingFailureException e) {
    entityManager.clear();
    CompanyVerification fresh = verificationRepository.findById(verificationId)
        .orElseThrow(() -> new BusinessException(ErrorCode.VERIFICATION_NOT_FOUND));
    if (fresh.getStatus().equals(targetStatus.name())) return fresh;
    throw new BusinessException(ErrorCode.INVALID_VERIFICATION_STATUS_TRANSITION,
        "verification was modified concurrently");
}
}
// — 管理员: 审核历史 —————
public List<Map<String, Object>> getHistory(Long verificationId) {
    return historyRepository.findByIdOrderByCreatedAtAsc(verificationId)
        .stream().map(this::toHistoryView).collect(Collectors.toList());
}
// — 视图 —————
-
public Map<String, Object> toView(CompanyVerification v) {
    Map<String, Object> view = new HashMap<>();
    view.put("id", v.getId());
    view.put("companyId", v.getCompanyId());
    view.put("status", v.getStatus());
    view.put("submittedBy", v.getSubmittedBy());
    view.put("materialsJson", v.getMaterialsJson());
    view.put("reviewerId", v.getReviewerId());
    view.put("decisionReason", v.getDecisionReason());
    view.put("submittedAt", v.getSubmittedAt());
    view.put("decidedAt", v.getDecidedAt());
    view.put("version", v.getVersion());
    view.put("createdAt", v.getCreatedAt());
    return view;
}
private Map<String, Object> toHistoryView(CompanyVerificationHistory h) {
    Map<String, Object> v = new HashMap<>();
    v.put("id", h.getId());
    v.put("fromStatus", h.getFromStatus());
    v.put("toStatus", h.getToStatus());
    v.put("actorUserId", h.getActorUserId());
    v.put("reason", h.getReason());
    v.put("createdAt", h.getCreatedAt());
    return v;
}
// — 私有辅助 —————
private Company requireCompany(Long userId) {
    User user = userRepository.findById(userId)
        .orElseThrow(() -> new BusinessException(ErrorCode.USER_NOT_FOUND));
    if (!hasRole(user, User.Role.COMPANY)) {
        throw new BusinessException(ErrorCode.FORBIDDEN);
    }
}

```

```

        return companyRepository.findByOwnerUserId(userId)
            .orElseThrow(() -> new BusinessException(ErrorCode.COMPANY_NOT_FOUND));
    }
    private boolean hasRole(User user, User.Role role) {
        return user.getRoles() != null && user.getRoles().stream().anyMatch(r -> r == role);
    }
    public void assertAdminRole(Long userId) {
        userRoleGuard.requireAdmin(userId);
    }
    private void recordHistory(Long verificationId, String fromStatus, String toStatus,
        Long actorUserId, String reason) {
        historyRepository.save(CompanyVerificationHistory.builder()
            .verificationId(verificationId)
            .fromStatus(fromStatus)
            .toStatus(toStatus)
            .actorUserId(actorUserId)
            .reason(reason)
            .build());
    }
}
package com.mydreamjob.verification;
/**
 * 企业认证状态枚举
 *
 * <p>状态流转:
 * <pre>
 * PENDING —> APPROVED (终态)
 * PENDING —> REJECTED
 * REJECTED —> RESUBMITTED —> PENDING (重新进入审核)
 * </pre>
 */
public enum VerificationStatus {
    PENDING,
    APPROVED,
    REJECTED,
    RESUBMITTED;
    public boolean canTransitionTo(VerificationStatus target) {
        if (target == null) return false;
        return switch (this) {
            case PENDING -> target == APPROVED || target == REJECTED;
            case REJECTED -> target == RESUBMITTED;
            case RESUBMITTED -> target == APPROVED || target == REJECTED;
            case APPROVED -> false;
        };
    }
    public boolean isTerminal() {
        return this == APPROVED;
    }
}
package com.mydreamjob.verification.dto;
import jakarta.validation.constraints.NotBlank;
import lombok.Data;
/**
 * 管理员审核决定请求
 */
@Data
public class VerificationDecisionRequest {
    @NotBlank(message = "decision is required")
    private String decision;
    private String reason;
}
package com.mydreamjob.verification.dto;
import jakarta.validation.constraints.NotBlank;
import lombok.Data;
/**
 * 企业提交认证材料请求
 */
@Data
public class VerificationSubmitRequest {
    @NotBlank(message = "materials is required")
    private String materialsJson;
}

```