好会帮——家教信息对接平台
V1.0 程序鉴别材料

权　　利　　人　　：	杨圣洲
软　件　名　称　：	好会帮——家教信息对接平台
英　　文　　名　　：	HiTutor
软 件 版 本 号 ：	V1.0

一、程序鉴别材料

```dart
import 'package:flutter/material.dart';
import 'package:provider/provider.dart';

import 'routes.dart';
import 'theme/app_theme.dart';

import 'providers/auth_provider.dart';
import 'providers/tutor_provider.dart';
import 'providers/message_provider.dart';

void main() async {
  WidgetsFlutterBinding.ensureInitialized();
  runApp(const HiTutorApp());
}

class HiTutorApp extends StatelessWidget {
  const HiTutorApp({super.key});

  @override
  Widget build(BuildContext context) {
    return MultiProvider(
      providers: [
        ChangeNotifierProvider(create: (context) => AuthProvider()),
        ChangeNotifierProvider(create: (context) => TutorProvider()),
        ChangeNotifierProvider(create: (context) => MessageProvider()),
      ],
      child: MaterialApp(
        title: 'HiTutor 好会帮',
        theme: AppTheme.lightTheme,
        initialRoute: Routes.splash,
        routes: Routes.routes,
        debugShowCheckedModeBanner: false,
      ),
    );
  }
}

import 'package:flutter/material.dart';
import 'pages/main/splash_page.dart';
import 'pages/auth/sms_login_page.dart';
import 'pages/auth/password_login_page.dart';
import 'pages/auth/forgot_password_page.dart';
import 'pages/auth/role_selection_page.dart';
import 'pages/main/main_tab_page.dart';
import 'pages/settings/settings_page.dart';
import 'pages/settings/profile_edit_page.dart';
import 'pages/settings/account_security_page.dart';
import 'pages/settings/change_phone_page.dart';
import 'pages/settings/change_email_page.dart';
import 'pages/about/user_agreement_page.dart';
import 'pages/about/privacy_policy_page.dart';
import 'pages/about/about_us_page.dart';

import 'pages/about/complaint_page.dart';
import 'pages/about/complaint_list_page.dart';
import 'pages/publish/publish_student_request_page.dart';
import 'pages/publish/publish_tutor_service_page.dart';
import 'pages/tutor/tutor_detail_page.dart';
import 'pages/student_request/student_request_detail_page.dart';
```

```dart
import 'pages/tutor_service/tutor_service_detail_page.dart';
import 'pages/review/review_page.dart';
import 'pages/appointment/appointment_page.dart';
import 'pages/settings/blacklist_page.dart';
import 'pages/services/customer_service_page.dart';
import 'pages/services/favorites_page.dart';
import 'pages/services/my_reviews_page.dart';
import 'pages/services/tutor_certification_page.dart';
import 'pages/services/tutor_resume_page.dart';
import 'pages/map/map_page.dart';
import 'pages/application/application_form_page.dart';
import 'pages/application/application_detail_page.dart';
import 'pages/chat/chat_detail_page.dart';
import 'pages/services/points_page.dart';
import 'pages/services/points_detail_page.dart';
import 'pages/services/my_publishings_page.dart';
import 'pages/application/my_applications_page.dart';
import 'pages/user/user_profile_page.dart';
import 'pages/notification/notification_page.dart';
import 'pages/subject/subject_explore_page.dart';

class Routes {
  static const String splash = '/';
  static const String smsLogin = '/sms-login';
  static const String passwordLogin = '/password-login';
  static const String forgotPassword = '/forgot-password';
  static const String roleSelection = '/role-selection';
  static const String mainTab = '/main-tab';
  static const String settings = '/settings';
  static const String userAgreement = '/user-agreement';
  static const String privacyPolicy = '/privacy-policy';
  static const String publishStudentRequest = '/publish-student-request';
  static const String publishTutorService = '/publish-tutor-service';
  static const String tutorDetail = '/tutor-detail';
  static const String studentRequestDetail = '/student-request-detail';
  static const String tutorServiceDetail = '/tutor-service-detail';
  static const String appointment = '/appointment';
  static const String review = '/review';
  static const String profileEdit = '/profile-edit';
  static const String accountSecurity = '/account-security';
  static const String changePhone = '/change-phone';
  static const String changeEmail = '/change-email';
  static const String aboutUs = '/about-us';

  static const String complaint = '/complaint';
  static const String complaintList = '/complaint-list';
  static const String customerService = '/customer-service';
  static const String favorites = '/favorites';
  static const String myReviews = '/my-reviews';
  static const String tutorCertification = '/tutor-certification';
  static const String tutorResume = '/tutor-resume';
  static const String map = '/map';
  static const String applicationForm = '/application-form';
  static const String applicationDetail = '/application-detail';

  static const String chatDetail = '/chat-detail';
  static const String blacklist = '/blacklist';
  static const String points = '/points';
  static const String pointsDetail = '/points-detail';
  static const String myPublishings = '/my-publishings';
  static const String myApplications = '/my-applications';
```

```dart
  static const String userProfile = '/user-profile';
  static const String notification = '/notification';
  static const String subjectExplore = '/subject-explore';

  static Map<String, WidgetBuilder> getRoutes() {
    return {
      splash: (context) => const SplashAdPage(),
      smsLogin: (context) => const SmsLoginPage(),
      passwordLogin: (context) => const PasswordLoginPage(),
      forgotPassword: (context) => const ForgotPasswordPage(),
      roleSelection: (context) {
        final userId = ModalRoute.of(context)?.settings.arguments as String?;
        return RoleSelectionPage(userId: userId ?? '');
      },
      mainTab: (context) => const MainTabPage(),
      settings: (context) => const SettingsPage(),
      userAgreement: (context) => const UserAgreementPage(),
      privacyPolicy: (context) => const PrivacyPolicyPage(),
      publishStudentRequest: (context) => const PublishStudentRequestPage(),
      publishTutorService: (context) => const PublishTutorServicePage(),
      tutorDetail: (context) {
        final tutor = ModalRoute.of(context)?.settings.arguments as dynamic;
        return TutorDetailPage(tutor: tutor);
      },
      studentRequestDetail: (context) {
        final request = ModalRoute.of(context)?.settings.arguments as dynamic;
        return StudentRequestDetailPage(request: request);
      },
      tutorServiceDetail: (context) {
        final tutor = ModalRoute.of(context)?.settings.arguments as dynamic;
        return TutorServiceDetailPage(tutor: tutor);
      },
      appointment: (context) => const AppointmentPage(),
      review: (context) => ReviewPage(
        appointmentId: ModalRoute.of(context)!.settings.arguments as String? ?? '',
        tutorId: '',
        tutorName: '',
      ),
      profileEdit: (context) => const ProfileEditPage(),
      accountSecurity: (context) => const AccountSecurityPage(),
      changePhone: (context) => const ChangePhonePage(),
      changeEmail: (context) => const ChangeEmailPage(),
      blacklist: (context) => const BlacklistPage(),
      aboutUs: (context) => const AboutUsPage(),

      complaint: (context) => const ComplaintPage(),
      complaintList: (context) => const ComplaintListPage(),
      customerService: (context) => const CustomerServicePage(),
      favorites: (context) => const FavoritesPage(),
      myReviews: (context) => const MyReviewsPage(),
      tutorCertification: (context) => const TutorCertificationPage(),
      tutorResume: (context) => const TutorResumePage(),
      map: (context) => MapPage(role: ModalRoute.of(context)!.settings.arguments as Str
ing? ?? 'student'),
      points: (context) => const PointsPage(),
      pointsDetail: (context) => const PointsDetailPage(),
      myPublishings: (context) => const MyPublishingsPage(),
      myApplications: (context) => const MyApplicationsPage(),
      applicationForm: (context) {
        final args = ModalRoute.of(context)?.settings.arguments as Map<String, dynami
c>?;
```

```
        return ApplicationFormPage(
          requestId: args?['requestId']?.toString() ?? '',
          requestType: args?['requestType']?.toString() ?? '',
          title: args?['title']?.toString() ?? '',
        );
      },
      applicationDetail: (context) {
        final args = ModalRoute.of(context)?.settings.arguments as Map<String, dynami
c>?;
        return ApplicationDetailPage(
          application: args?['application'] as Map<String, dynamic>? ?? {},
        );
      },
      chatDetail: (context) {
        final args = ModalRoute.of(context)?.settings.arguments as Map<String, dynami
c>?;
        return ChatDetailPage(
          conversationId: args?['conversationId']?.toString() ?? '',
          otherUserId: args?['otherUserId']?.toString() ?? '',
          otherUserName: args?['otherUserName']?.toString() ?? '',
        );
      },
      userProfile: (context) {
        final args = ModalRoute.of(context)?.settings.arguments as Map<String, dynami
c>?;
        return UserProfilePage(
          userId: args?['userId']?.toString() ?? '',
          userName: args?['userName']?.toString() ?? '',
        );
      },
      notification: (context) => const NotificationPage(),
      subjectExplore: (context) {
        final defaultSubject = ModalRoute.of(context)?.settings.arguments as String?;
        return SubjectExplorePage(defaultSubject: defaultSubject);
      },
    };
  }

  static Map<String, WidgetBuilder> get routes => getRoutes();
}

import 'dart:convert';
import 'package:http/http.dart' as http;

class ApiService {
  static const String baseUrl = 'http://120.55.50.51/api';
  static String? _token;

  static void setToken(String token) {
    _token = token;
  }

  static void clearToken() {
    _token = null;
  }

  static Map<String, String> _getHeaders({bool needAuth = true}) {
    final headers = <String, String>{
      'Content-Type': 'application/json',
    };
    if (needAuth && _token != null) {
```

```
    headers['Authorization'] = 'Bearer $_token';
  }
  return headers;
}

static Future<dynamic> _handleResponse(http.Response response) async {
  try {
    if (response.statusCode == 204) {
      return {'success': true, 'data': null};
    }

    if (response.body.isEmpty) {
      throw Exception('响应体为空，状态码: ${response.statusCode}');
    }

    final body = jsonDecode(response.body);

    if (response.statusCode >= 200 && response.statusCode < 300) {
      if (body is Map) {
        if (body.containsKey('success')) {
          if (body['success'] == true) {
            return body;
          } else {
            final message = body['message'] ?? '请求失败';
            throw Exception(message);
          }
        }
      }
      return body;
    } else {
      String errorMessage;
      if (body is Map) {
        switch (response.statusCode) {
          case 400:
            errorMessage = body['message'] ?? '请求参数错误';
            break;
          case 401:
            errorMessage = body['message'] ?? '未授权，请重新登录';
            break;
          case 403:
            errorMessage = body['message'] ?? '禁止访问';
            break;
          case 404:
            errorMessage = body['message'] ?? '资源不存在';
            break;
          case 409:
            errorMessage = body['message'] ?? '资源冲突';
            break;
          case 500:
            errorMessage = body['message'] ?? '服务器内部错误';
            break;
          default:
            errorMessage = body['message'] ?? '请求失败';
        }
      } else {
        errorMessage = '请求失败';
      }
      throw Exception(errorMessage);
    }
```

```
    } catch (e) {
      if (e is FormatException) {
        throw Exception('JSON 解析错误: ${e.message}, 响应体: ${response.body}');
      } else {
        rethrow;
      }
    }
  }


  static Future<dynamic> login(String email, String password) async {
    final response = await http.post(
      Uri.parse('$baseUrl/auth/login'),
      headers: _getHeaders(needAuth: false),
      body: jsonEncode({
        'email': email,
        'password': password,
      }),
    );
    return await _handleResponse(response);
  }

  static Future<dynamic> loginByPassword(String phone, String password) async {
    final response = await http.post(
      Uri.parse('$baseUrl/auth/login-password'),
      headers: _getHeaders(needAuth: false),
      body: jsonEncode({
        'phone': phone,
        'password': password,
      }),
    );
    return await _handleResponse(response);
  }

  static Future<dynamic> loginBySms(String phone, String verificationCode, String role)
async {
    final response = await http.post(
      Uri.parse('$baseUrl/auth/login-sms'),
      headers: _getHeaders(needAuth: false),
      body: jsonEncode({
        'phone': phone,
        'verificationCode': verificationCode,
        'role': role,
      }),
    );
    return await _handleResponse(response);
  }

  static Future<dynamic> refreshToken(String refreshToken) async {
    final response = await http.post(
      Uri.parse('$baseUrl/auth/refresh-token'),
      headers: _getHeaders(needAuth: false),
      body: jsonEncode({'refreshToken': refreshToken}),
    );
    return await _handleResponse(response);
  }

  static Future<dynamic> register(
    String username,
    String password,
    String email,
```

```
    String role,
    String phone,
    String verificationCode,
  ) async {
    final response = await http.post(
      Uri.parse('$baseUrl/auth/register'),
      headers: _getHeaders(needAuth: false),
      body: jsonEncode({
        'username': username,
        'password': password,
        'email': email,
        'role': role,
        'phone': phone,
        'verificationCode': verificationCode,
      }),
    );
    return await _handleResponse(response);
  }

  static Future<void> forgotPassword(String phone) async {
    final response = await http.post(
      Uri.parse('$baseUrl/auth/forgot-password'),
      headers: _getHeaders(needAuth: false),
      body: jsonEncode({'phone': phone}),
    );
    await _handleResponse(response);
  }

  static Future<void> resetPassword(
    String phone,
    String verificationCode,
    String newPassword,
  ) async {
    final response = await http.post(
      Uri.parse('$baseUrl/auth/reset-password'),
      headers: _getHeaders(needAuth: false),
      body: jsonEncode({
        'phone': phone,
        'verificationCode': verificationCode,
        'newPassword': newPassword,
      }),
    );
    await _handleResponse(response);
  }

  static Future<dynamic> checkLogin() async {
    final response = await http.post(
      Uri.parse('$baseUrl/auth/check-login'),
      headers: _getHeaders(),
    );
    return await _handleResponse(response);
  }

  static Future<void> sendVerificationCode(String phone) async {
    final response = await http.post(
      Uri.parse('$baseUrl/verifications/send'),
      headers: _getHeaders(needAuth: false),
      body: jsonEncode({
        'phone': phone,
      }),
    );
```

```
    await _handleResponse(response);
  }

  static Future<void> verifyCode(String phone, String code, String type) async {
    final response = await http.post(
      Uri.parse('$baseUrl/verifications/verify'),
      headers: _getHeaders(needAuth: false),
      body: jsonEncode({
        'phone': phone,
        'code': code,
        'type': type,
      }),
    );
    await _handleResponse(response);
  }

await _getCurrentLocationOptimized();

if (_mapController != null && _currentLocation != null) {
  await _loadMarkers();
}

}
```

```
Future _requestLocationPermission() async {
var status = await Permission.location.status;
if (status.isGranted) {
return true;
}
```

```
status = await Permission.location.request();
if (status.isGranted) {
  return true;
} else if (status.isDenied) {
  status = await Permission.location.request();
  return status.isGranted;
} else if (status.isPermanentlyDenied) {
  if (!mounted) return false;
  showDialog(
    context: context,
    builder: (context) => AlertDialog(
      title: const Text('位置权限被拒绝'),
      content: const Text('需要位置权限才能显示附近的家教/学生，请前往设置开启'),
      actions: [
        TextButton(
          onPressed: () => Navigator.pop(context),
          child: const Text('取消'),
        ),
        TextButton(
          onPressed: () async {
            Navigator.pop(context);
            await openAppSettings();
          },
          child: const Text('去设置'),
        ),
      ],
    ),
  );
  return false;
```

```
}
return false;

}

Future _getCurrentLocation() async {
await _getCurrentLocationOptimized();
}

Future _getCurrentLocationOptimized() async {
try {
bool serviceEnabled = await Geolocator.isLocationServiceEnabled();
if (!serviceEnabled) {
if (mounted) {
setState(() {
_isLoadingLocation = false;
});
}
final lastPosition = await Geolocator.getLastKnownPosition();
if (lastPosition != null) {
_currentLocation = LatLng(lastPosition.latitude, lastPosition.longitude);
} else {
_currentLocation = const LatLng(28.66999503, 115.82297033);
}
return;
}

  LocationPermission permission = await Geolocator.checkPermission();
  if (permission == LocationPermission.denied || permission == LocationPermission.denie
dForever) {
    if (mounted) {
      setState(() {
        _isLoadingLocation = false;
      });
    }
    _currentLocation = const LatLng(28.66999503, 115.82297033);
    return;
  }

  Position? position;
  try {
    position = await Geolocator.getCurrentPosition(
      desiredAccuracy: LocationAccuracy.medium,
      timeLimit: const Duration(seconds: 3),
    );
  } catch (e) {
    position ??= await Geolocator.getLastKnownPosition();
  }

  position ??= Position(
    latitude: 28.66999503,
    longitude: 115.82297033,
    timestamp: DateTime.now(),
    accuracy: 0,
    altitude: 0,
    altitudeAccuracy: 0,
    heading: 0,
    headingAccuracy: 0,
    speed: 0,
    speedAccuracy: 0,
  );
```

```
  if (mounted) {
    setState(() {
      _currentLocation = LatLng(position.latitude, position.longitude);
      _isLoadingLocation = false;
    });
  }
} catch (e) {
  if (mounted) {
    setState(() {
      _isLoadingLocation = false;
    });
  }
  _currentLocation = const LatLng(28.66999503, 115.82297033);
}

}
```

Future *loadMarkers() async {*
*if (*currentLocation == null || _isDisposed) return;

```
try {
  final tutorProvider = Provider.of<TutorProvider>(context, listen: false);

  if (widget.role == 'tutor') {
    await tutorProvider.getNearbyStudents();
  } else {
    await tutorProvider.getNearbyTutors();
  }

  if (_isDisposed) return;

  final tutors = widget.role == 'tutor'
      ? tutorProvider.students
      : tutorProvider.tutors;

  setState(() {
    _tutors.clear();
    _tutors.addAll(tutors);
  });

  await _createMarkers();
} catch (e) {
  if (mounted) {
    ScaffoldMessenger.of(context).showSnackBar(
      SnackBar(content: Text('加载数据失败: ${e.toString()}')),
    );
  }
}

}
```

Future *createMarkers() async {*
*if (*isDisposed) return;

```
final List<Marker> markers = [];

for (final tutor in _tutors) {
  if (tutor.latitude == null || tutor.longitude == null) continue;

  final markerId = MarkerId(tutor.id.toString());
```

```
  final marker = Marker(
    markerId: markerId,
    position: LatLng(tutor.latitude!, tutor.longitude!),
    infoWindow: InfoWindow(
      title: tutor.name,
      snippet: tutor.subjects.map((s) => s.name).join('、'),
    ),
    icon: await _getMarkerIcon(tutor),
  );

  markers.add(marker);
}

if (_isDisposed) return;

setState(() {
  _markers.clear();
  _markers.addAll(markers);
});

}

Future _getMarkerIcon(Tutor tutor) async {
final size = const Size(80, 80);
final pictureRecorder = ui.PictureRecorder();
final canvas = Canvas(pictureRecorder);
final paint = Paint()..color = AppTheme.primaryColor;

canvas.drawCircle(
  Offset(size.width / 2, size.height / 2),
  size.width / 2,
  paint,
);

final textPainter = TextPainter(
  text: TextSpan(
    text: tutor.name.substring(0, 1),
    style: const TextStyle(
      color: Colors.white,
      fontSize: 24,
      fontWeight: FontWeight.bold,
    ),
  ),
  textDirection: TextDirection.ltr,
);
textPainter.layout();
textPainter.paint(
  canvas,
  Offset(
    (size.width - textPainter.width) / 2,
    (size.height - textPainter.height) / 2,
  ),
);

final picture = pictureRecorder.endRecording();
final image = await picture.toImage(
  size.width.toInt(),
  size.height.toInt(),
);
final byteData = await image.toByteData(format: ui.ImageByteFormat.png);
```

```
if (byteData == null) {
  return BitmapDescriptor.defaultMarker;
}

return BitmapDescriptor.fromBytes(byteData.buffer.asUint8List());

}

@override
void dispose() {
_isDisposed = true;
_mapController?.dispose();
super.dispose();
}

@override
Widget build(BuildContext context) {
super.build(context);
return Scaffold(
backgroundColor: AppTheme.backgroundColor,
appBar: AppBar(
backgroundColor: Colors.white,
elevation: 0,
leading: IconButton(
icon: const Icon(Icons.arrow_back_ios_new, size: 20),
onPressed: () => Navigator.pop(context),
color: AppTheme.textPrimary,
),
title: Text(
widget.role == 'tutor' ? '附近学生' : '附近家教',
style: const TextStyle(
fontSize: 18,
fontWeight: FontWeight.w600,
color: AppTheme.textPrimary,
),
),
centerTitle: true,
actions: [
IconButton(
icon: const Icon(Icons.refresh),
onPressed: () {
if (currentLocation != null) {
_loadMarkers();
}
},
color: AppTheme.textPrimary,
),
],
),
body: Stack(
children: [
if (currentLocation != null)
AMapFlutterMap(
initialCameraPosition: CameraPosition(
target: currentLocation!,
zoom: 14,
),
onMapCreated: (controller) {
_mapController = controller;
```

```
if (isInitialized) {
loadMarkers();
}
},
onCameraMove: (position) {
},
onCameraMoveEnd: (position) {
},
markers: Set.of(markers),
myLocationEnabled: true,
myLocationButtonEnabled: true,
compassEnabled: true,
scaleControlsEnabled: true,
zoomControlsEnabled: true,
mapType: MapType.normal,
)
else if (_isLoadingLocation)
const Center(
child: Column(
mainAxisAlignment: MainAxisAlignment.center,
children: [
CircularProgressIndicator(color: AppTheme.primaryColor),
SizedBox(height: 16),
Text(
'正在获取位置...',
style: TextStyle(
fontSize: 14,
color: AppTheme.textSecondary,
),
),
],
),
)
else
const Center(
child: Column(
mainAxisAlignment: MainAxisAlignment.center,
children: [
Icon(
Icons.location_off,
size: 48,
color: AppTheme.textTertiary,
),
SizedBox(height: 16),
Text(
'无法获取位置',
style: TextStyle(
fontSize: 16,
color: AppTheme.textSecondary,
),
),
],
),
),
],
),
);
```

}
}

1.2 路由配置文件

路由配置文件定义了应用中所有页面的路由路径和对应的页面组件，文件位于`client/lib/routes.dart`。Routes 类定义了所有路由路径的常量，包括启动页、登录页、主页、发布页、详情页、聊天页、预约页、设置页等各种功能页面的路由路径。路由路径采用分层结构，便于管理和维护。

Routes 类的 routes 属性是一个 Map 类型，将路由路径映射到对应的页面组件构建器。每个路由路径都对应一个页面组件，例如 splash 路由对应 SplashPage 组件，passwordLogin 路由对应 PasswordLoginPage 组件，home 路由对应 HomePage 组件。路由配置使应用能够根据路由路径导航到对应的页面，实现页面之间的跳转和导航。

```dart
class Routes {
  static const String splash = '/';
  static const String passwordLogin = '/auth/password-login';
  static const String smsLogin = '/auth/sms-login';
  static const String roleSelection = '/auth/role-selection';
  static const String forgotPassword = '/auth/forgot-password';
  static const String mainTab = '/main/tab';
  static const String home = '/main/home';
  static const String discover = '/main/discover';
  static const String message = '/main/message';
  static const String profile = '/main/profile';
  static const String publishTutorService = '/publish/tutor-service';
  static const String publishStudentRequest = '/publish/student-request';
  static const String tutorDetail = '/tutor/detail';
  static const String studentRequestDetail = '/student-request/detail';
  static const String chatDetail = '/chat/detail';
  static const String appointment = '/appointment';
  static const String createAppointment = '/appointment/create';
  static const String applicationList = '/application/list';
  static const String applicationDetail = '/application/detail';
  static const String applicationForm = '/application/form';
  static const String myApplications = '/application/my';
  static const String favorites = '/services/favorites';
  static const String myPublishings = '/services/my-publishings';
  static const String myReviews = '/services/my-reviews';
  static const String points = '/services/points';
  static const String pointsDetail = '/services/points-detail';
  static const String tutorCertification = '/services/tutor-certification';
  static const String tutorResume = '/services/tutor-resume';
  static const String customerService = '/services/customer-service';
  static const String settings = '/settings';
  static const String profileEdit = '/settings/profile-edit';
  static const String accountSecurity = '/settings/account-security';
  static const String changePhone = '/settings/change-phone';
  static const String changeEmail = '/settings/change-email';
  static const String blacklist = '/settings/blacklist';
  static const String notification = '/notification';
  static const String review = '/review';
  static const String aboutUs = '/about/about-us';
  static const String userAgreement = '/about/user-agreement';
  static const String privacyPolicy = '/about/privacy-policy';
  static const String complaint = '/about/complaint';
  static const String complaintList = '/about/complaint-list';
  static const String userProfile = '/user/profile';
```

```
  static const String subjectExplore = '/subject/explore';
  static const String map = '/map';
  static const String tutorServiceDetail = '/tutor-service/detail';

  static Map<String, WidgetBuilder> routes = {
    splash: (context) => const SplashPage(),
    passwordLogin: (context) => const PasswordLoginPage(),
    smsLogin: (context) => const SmsLoginPage(),
    roleSelection: (context) => const RoleSelectionPage(),
    forgotPassword: (context) => const ForgotPasswordPage(),
    mainTab: (context) => const MainTabPage(),
    home: (context) => const HomePage(),
    discover: (context) => const DiscoverPage(),
    message: (context) => const MessagePage(),
    profile: (context) => const ProfilePage(),
    publishTutorService: (context) => const PublishTutorServicePage(),
    publishStudentRequest: (context) => const PublishStudentRequestPage(),
    tutorDetail: (context) => const TutorDetailPage(),
    studentRequestDetail: (context) => const StudentRequestDetailPage(),
    chatDetail: (context) => const ChatDetailPage(),
    appointment: (context) => const AppointmentPage(),
    createAppointment: (context) => const CreateAppointmentPage(),
    applicationList: (context) => const ApplicationListPage(),
    applicationDetail: (context) => const ApplicationDetailPage(),
    applicationForm: (context) => const ApplicationFormPage(),
    myApplications: (context) => const MyApplicationsPage(),
    favorites: (context) => const FavoritesPage(),
    myPublishings: (context) => const MyPublishingsPage(),
    myReviews: (context) => const MyReviewsPage(),
    points: (context) => const PointsPage(),
    pointsDetail: (context) => const PointsDetailPage(),
    tutorCertification: (context) => const TutorCertificationPage(),
    tutorResume: (context) => const TutorResumePage(),
    customerService: (context) => const CustomerServicePage(),
    settings: (context) => const SettingsPage(),
    profileEdit: (context) => const ProfileEditPage(),
    accountSecurity: (context) => const AccountSecurityPage(),
    changePhone: (context) => const ChangePhonePage(),
    changeEmail: (context) => const ChangeEmailPage(),
    blacklist: (context) => const BlacklistPage(),
    notification: (context) => const NotificationPage(),
    review: (context) => const ReviewPage(),
    aboutUs: (context) => const AboutUsPage(),
    userAgreement: (context) => const UserAgreementPage(),
    privacyPolicy: (context) => const PrivacyPolicyPage(),
    complaint: (context) => const ComplaintPage(),
    complaintList: (context) => const ComplaintListPage(),
    userProfile: (context) => const UserProfilePage(),
    subjectExplore: (context) => const SubjectExplorePage(),
    map: (context) => const MapPage(),
    tutorServiceDetail: (context) => const TutorServiceDetailPage(),
  };
}
```

1.3 API 服务文件

API 服务文件封装了所有与后端 API 交互的方法，文件位于 `client/lib/services/api_service.dart`。ApiService 类定义了 baseUrl 常量，指定后端 API 的基础地址，默认为 `http://localhost:8080/api`。ApiService 类提供了四个静态方法：post、get、put、delete，分别对应 HTTP 的 POST、GET、PUT、DELETE 请求方法。

post 方法接收 endpoint 端点、data 数据和可选的 token 令牌，构造完整的 URL，设置请求头，包括 Content-Type 和 Authorization，发送 POST 请求，将 data 数据编码为 JSON 格式，返回解析后的响应数据。get 方法接收 endpoint 端点和可选的 token 令牌，构造完整的 URL，设置请求头，发送 GET 请求，返回解析后的响应数据。put 方法接收 endpoint 端点、data 数据和可选的 token 令牌，构造完整的 URL，设置请求头，发送 PUT 请求，将 data 数据编码为 JSON 格式，返回解析后的响应数据。delete 方法接收 endpoint 端点和可选的 token 令牌，构造完整的 URL，设置请求头，发送 DELETE 请求，返回解析后的响应数据。

```dart
import 'dart:convert';
import 'package:http/http.dart' as http;

class ApiService {
  static const String baseUrl = 'http://localhost:8080/api';

  static Future<Map<String, dynamic>> post(String endpoint, dynamic data, {String? toke
n}) async {
    final url = Uri.parse('$baseUrl$endpoint');
    final headers = {
      'Content-Type': 'application/json',
      if (token != null) 'Authorization': 'Bearer $token',
    };
    final response = await http.post(
      url,
      headers: headers,
      body: jsonEncode(data),
    );
    return jsonDecode(response.body);
  }

  static Future<Map<String, dynamic>> get(String endpoint, {String? token}) async {
    final url = Uri.parse('$baseUrl$endpoint');
    final headers = {
      if (token != null) 'Authorization': 'Bearer $token',
    };
    final response = await http.get(
      url,
      headers: headers,
    );
    return jsonDecode(response.body);
  }

  static Future<Map<String, dynamic>> put(String endpoint, dynamic data, {String? toke
n}) async {
    final url = Uri.parse('$baseUrl$endpoint');
    final headers = {
      'Content-Type': 'application/json',
      if (token != null) 'Authorization': 'Bearer $token',
    };
    final response = await http.put(
      url,
      headers: headers,
      body: jsonEncode(data),
    );
    return jsonDecode(response.body);
  }

  static Future<Map<String, dynamic>> delete(String endpoint, {String? token}) async {
    final url = Uri.parse('$baseUrl$endpoint');
    final headers = {
      if (token != null) 'Authorization': 'Bearer $token',
    };
```

```
    final response = await http.delete(
      url,
      headers: headers,
    );
    return jsonDecode(response.body);
  }
}
```

1.4 认证提供者

认证提供者负责管理用户认证状态，文件位于 `client/lib/providers/auth_provider.dart`。AuthProvider 类继承自 ChangeNotifier，实现了状态管理功能。类中定义了私有变量：*user 存储当前用户信息*，token 存储认证令牌，*isLoading 表示加载状态*，error 存储错误信息。类中定义了 getter 方法：user 获取当前用户，token 获取认证令牌，isLoading 获取加载状态，error 获取错误信息，isAuthenticated 判断用户是否已认证。

loginWithPassword 方法接收手机号和密码，设置加载状态为 true，清除错误信息，调用 AuthService 的 loginWithPassword 方法进行登录，如果登录成功，保存令牌和用户信息，通知监听器，返回 true，如果登录失败，保存错误信息，通知监听器，返回 false，捕获异常并返回 false，最后设置加载状态为 false，通知监听器。loginWithSms 方法接收手机号和验证码，逻辑与 loginWithPassword 方法类似。register 方法接收手机号、密码和角色，调用 AuthService 的 register 方法进行注册，逻辑与登录方法类似。logout 方法清除用户信息和令牌，调用 AuthService 的 clearToken 方法清除本地存储的令牌，通知监听器。loadUser 方法从本地存储获取令牌，如果令牌存在，保存令牌，调用 AuthService 的 getUserInfo 方法获取用户信息，如果获取成功，保存用户信息，如果获取失败，清除令牌，通知监听器。

```
import 'package:flutter/material.dart';
import '../models/user_model.dart';
import '../services/auth_service.dart';

class AuthProvider extends ChangeNotifier {
  UserModel? _user;
  String? _token;
  bool _isLoading = false;
  String? _error;

  UserModel? get user => _user;
  String? get token => _token;
  bool get isLoading => _isLoading;
  String? get error => _error;
  bool get isAuthenticated => _token != null && _user != null;

  Future<bool> loginWithPassword(String phone, String password) async {
    _isLoading = true;
    _error = null;
    notifyListeners();

    try {
      final result = await AuthService.loginWithPassword(phone, password);
      if (result['success']) {
        _token = result['data']['token'];
        _user = UserModel.fromJson(result['data']['user']);
        notifyListeners();
        return true;
      } else {
        _error = result['message'];
        notifyListeners();
        return false;
      }
    } catch (e) {
```

```
      _error = '登录失败，请稍后重试';
      notifyListeners();
      return false;
    } finally {
      _isLoading = false;
      notifyListeners();
    }
  }

  Future<bool> loginWithSms(String phone, String code) async {
    _isLoading = true;
    _error = null;
    notifyListeners();

    try {
      final result = await AuthService.loginWithSms(phone, code);
      if (result['success']) {
        _token = result['data']['token'];
        _user = UserModel.fromJson(result['data']['user']);
        notifyListeners();
        return true;
      } else {
        _error = result['message'];
        notifyListeners();
        return false;
      }
    } catch (e) {
      _error = '登录失败，请稍后重试';
      notifyListeners();
      return false;
    } finally {
      _isLoading = false;
      notifyListeners();
    }
  }

  Future<bool> register(String phone, String password, String role) async {
    _isLoading = true;
    _error = null;
    notifyListeners();

    try {
      final result = await AuthService.register(phone, password, role);
      if (result['success']) {
        _token = result['data']['token'];
        _user = UserModel.fromJson(result['data']['user']);
        notifyListeners();
        return true;
      } else {
        _error = result['message'];
        notifyListeners();
        return false;
      }
    } catch (e) {
      _error = '注册失败，请稍后重试';
      notifyListeners();
      return false;
    } finally {
      _isLoading = false;
      notifyListeners();
```

```
    }
  }

  Future<void> logout() async {
    _user = null;
    _token = null;
    await AuthService.clearToken();
    notifyListeners();
  }

  Future<void> loadUser() async {
    final savedToken = await AuthService.getToken();
    if (savedToken != null) {
      _token = savedToken;
      try {
        final userData = await AuthService.getUserInfo(savedToken);
        if (userData['success']) {
          _user = UserModel.fromJson(userData['data']);
        }
      } catch (e) {
        await AuthService.clearToken();
        _token = null;
      }
      notifyListeners();
    }
  }
}
```

1.5 用户模型

用户模型定义了用户的数据结构，文件位于 `client/lib/models/user_model.dart`。UserModel 类定义了用户的各个属性：id 用户 ID，username 用户名，phone 手机号，email 邮箱，role 角色，isVerified 是否已认证，points 积分，avatar 头像，gender 性别，birthDate 生日，education 教育背景，school 学校，major 专业，teachingExperience 教学经验。所有属性都是 final 类型，表示不可变。

UserModel 类的构造函数接收所有属性的值，使用 required 关键字标记必需的属性。fromJson 工厂方法接收一个 Map 类型的 JSON 数据，解析 JSON 数据并创建 UserModel 对象，使用 null 合并运算符提供默认值，确保属性不为 null。toJson 方法将 UserModel 对象转换为 Map 类型的 JSON 数据，返回包含所有属性值的 Map。

```
class UserModel {
  final String id;
  final String username;
  final String phone;
  final String? email;
  final String role;
  final bool isVerified;
  final int points;
  final String? avatar;
  final String? gender;
  final String? birthDate;
  final String? education;
  final String? school;
  final String? major;
  final int? teachingExperience;

  UserModel({
    required this.id,
    required this.username,
    required this.phone,
    this.email,
```

```
    required this.role,
    required this.isVerified,
    required this.points,
    this.avatar,
    this.gender,
    this.birthDate,
    this.education,
    this.school,
    this.major,
    this.teachingExperience,
  });

  factory UserModel.fromJson(Map<String, dynamic> json) {
    return UserModel(
      id: json['id'] ?? '',
      username: json['username'] ?? json['name'] ?? '',
      phone: json['phone'] ?? '',
      email: json['email'],
      role: json['role'] ?? '',
      isVerified: json['isVerified'] ?? false,
      points: json['points'] ?? 0,
      avatar: json['avatar'],
      gender: json['gender'],
      birthDate: json['birthDate'],
      education: json['education'],
      school: json['school'],
      major: json['major'],
      teachingExperience: json['teachingExperience'],
    );
  }

  Map<String, dynamic> toJson() {
    return {
      'id': id,
      'username': username,
      'phone': phone,
      'email': email,
      'role': role,
      'isVerified': isVerified,
      'points': points,
      'avatar': avatar,
      'gender': gender,
      'birthDate': birthDate,
      'education': education,
      'school': school,
      'major': major,
      'teachingExperience': teachingExperience,
    };
  }
}
```

1.6 家教提供者

家教提供者负责管理家教信息状态，文件位于 `client/lib/providers/tutor_provider.dart`。TutorProvider 类继承自 ChangeNotifier，实现了状态管理功能。类中定义了私有变量：*tutors 存储家教信息列表*，myTutors 存储我的家教信息列表，*currentTutor 存储当前家教信息*，isLoading 表示加载状态，_error 存储错误信息。类中定义了 getter 方法：tutors 获取家教信息列表，myTutors 获取我的家教信息列表，currentTutor 获取当前家教信息，isLoading 获取加载状态，error 获取错误信息。

fetchTutors 方法接收可选的 token 令牌和 filters 筛选条件，设置加载状态为 true，清除错误信息，调用 ApiService 的 get 方法获取家教信息列表，如果获取成功，解析响应数据，将数据转换为 TutorModel 对象列表，保存到_tutors 变量，通知监听器，返回 true，如果获取失败，保存错误信息，通知监听器，返回 false，捕获异常并返回 false，最后设置加载状态为 false，通知监听器。fetchMyTutors 方法接收 token 令牌，调用 ApiService 的 get 方法获取我的家教信息列表，逻辑与 fetchTutors 方法类似。publishTutorProfile 方法接收 data 数据和 token 令牌，调用 ApiService 的 post 方法发布家教信息，逻辑与 fetchTutors 方法类似。getTutorDetail 方法接收 id 和 token 令牌，调用 ApiService 的 get 方法获取家教详情，逻辑与 fetchTutors 方法类似。

```dart
import 'package:flutter/material.dart';
import '../models/tutor_model.dart';
import '../services/api_service.dart';

class TutorProvider extends ChangeNotifier {
  List<TutorModel> _tutors = [];
  List<TutorModel> _myTutors = [];
  TutorModel? _currentTutor;
  bool _isLoading = false;
  String? _error;

  List<TutorModel> get tutors => _tutors;
  List<TutorModel> get myTutors => _myTutors;
  TutorModel? get currentTutor => _currentTutor;
  bool get isLoading => _isLoading;
  String? get error => _error;

  Future<bool> fetchTutors({String? token, Map<String, dynamic>? filters}) async {
    _isLoading = true;
    _error = null;
    notifyListeners();

    try {
      final result = await ApiService.get('/tutor-profiles', token: token);
      if (result['success']) {
        _tutors = (result['data']['content'] as List)
            .map((item) => TutorModel.fromJson(item))
            .toList();
        notifyListeners();
        return true;
      } else {
        _error = result['message'];
        notifyListeners();
        return false;
      }
    } catch (e) {
      _error = '获取家教信息失败';
      notifyListeners();
      return false;
    } finally {
      _isLoading = false;
      notifyListeners();
    }
  }

  Future<bool> fetchMyTutors(String token) async {
    _isLoading = true;
    _error = null;
    notifyListeners();
```

```
    try {
      final result = await ApiService.get('/tutor-profiles/my', token: token);
      if (result['success']) {
        _myTutors = (result['data'] as List)
            .map((item) => TutorModel.fromJson(item))
            .toList();
        notifyListeners();
        return true;
      } else {
        _error = result['message'];
        notifyListeners();
        return false;
      }
    } catch (e) {
      _error = '获取我的家教信息失败';
      notifyListeners();
      return false;
    } finally {
      _isLoading = false;
      notifyListeners();
    }
  }

  Future<bool> publishTutorProfile(Map<String, dynamic> data, String token) async {
    _isLoading = true;
    _error = null;
    notifyListeners();

    try {
      final result = await ApiService.post('/tutor-profiles', data, token: token);
      if (result['success']) {
        notifyListeners();
        return true;
      } else {
        _error = result['message'];
        notifyListeners();
        return false;
      }
    } catch (e) {
      _error = '发布家教信息失败';
      notifyListeners();
      return false;
    } finally {
      _isLoading = false;
      notifyListeners();
    }
  }

  Future<bool> getTutorDetail(String id, String token) async {
    _isLoading = true;
    _error = null;
    notifyListeners();

    try {
      final result = await ApiService.get('/tutor-profiles/$id', token: token);
      if (result['success']) {
        _currentTutor = TutorModel.fromJson(result['data']);
        notifyListeners();
        return true;
      } else {
        _error = result['message'];
```

```
        notifyListeners();
        return false;
      }
    } catch (e) {
      _error = '获取家教详情失败';
      notifyListeners();
      return false;
    } finally {
      _isLoading = false;
      notifyListeners();
    }
  }
}
```

1.7 密码登录页面

密码登录页面提供用户通过手机号和密码登录的功能，文件位于 client/lib/pages/auth/password_login_page.dart。PasswordLoginPage 类是一个有状态组件，使用 StatefulWidget 实现。类中定义了私有变量：*formKey 表单键*，phoneController 手机号输入控制器，*passwordController 密码输入控制器*，isLoading 加载状态，*obscurePassword 密码是否可见。类中定义了 getter 方法：*isFormValid 判断表单是否有效。

initState 方法初始化组件，调用父类的 initState 方法，为手机号和密码输入控制器添加监听器，监听输入变化并更新表单状态。*updateFormState 方法更新表单状态，调用 setState 方法重新构建组件。dispose 方法释放资源，释放手机号和密码输入控制器，调用父类的 dispose 方法。*handleLogin 方法处理登录逻辑，验证表单，设置加载状态为 true，调用 ApiService 的 loginByPassword 方法进行登录，如果登录成功，获取用户数据和令牌，判断是否首次登录，如果是首次登录，跳转到角色选择页面，如果不是首次登录，跳转到主页，如果登录失败，显示错误提示，最后设置加载状态为 false。

build 方法构建组件界面，返回 Scaffold 组件，设置背景颜色，使用 SafeArea 包裹内容，使用 SingleChildScrollView 支持滚动，设置内边距，显示应用名称和标语，显示登录表单，包括手机号输入框和密码输入框，显示登录按钮，根据表单是否有效和加载状态设置按钮是否可用，显示验证码登录和忘记密码按钮，点击按钮跳转到对应页面。

```
import 'package:flutter/material.dart';
import '../../theme/app_theme.dart';
import '../../providers/auth_provider.dart';
import '../../services/api_service.dart';
import 'role_selection_page.dart';
import 'package:provider/provider.dart';

class PasswordLoginPage extends StatefulWidget {
  const PasswordLoginPage({super.key});

  @override
  State<PasswordLoginPage> createState() => _PasswordLoginPageState();
}

class _PasswordLoginPageState extends State<PasswordLoginPage> {
  final _formKey = GlobalKey<FormState>();
  final _phoneController = TextEditingController();
  final _passwordController = TextEditingController();
  bool _isLoading = false;
  bool _obscurePassword = true;

  bool get _isFormValid {
    final phone = _phoneController.text.trim();
    final password = _passwordController.text.trim();
    return phone.isNotEmpty &&
```

```
        RegExp(r'^1[3-9]\d{9}$').hasMatch(phone) &&
        password.isNotEmpty &&
        password.length >= 6;
  }

  @override
  void initState() {
    super.initState();
    _phoneController.addListener(_updateFormState);
    _passwordController.addListener(_updateFormState);
  }

  void _updateFormState() {
    setState(() {});
  }

  @override
  void dispose() {
    _phoneController.dispose();
    _passwordController.dispose();
    super.dispose();
  }

  Future<void> _handleLogin() async {
    if (_formKey.currentState?.validate() ?? false) {
      setState(() {
        _isLoading = true;
      });

      try {
        final phone = _phoneController.text.trim();
        final password = _passwordController.text.trim();

        final authProvider = Provider.of<AuthProvider>(context, listen: false);

        final response = await ApiService.loginByPassword(phone, password);

        if (response['success'] == true) {
          final data = response['data'];
          final user = data['user'];
          final token = data['token'];
          final isFirstLogin = data['isFirstLogin'] ?? false;

          await authProvider.loginWithToken(user, token, isFirstLogin: isFirstLogin);

          if (!mounted) return;

          if (isFirstLogin) {
            final userId = (user['id'] != null) ? user['id'].toString() : '';
            Navigator.pushReplacement(
              context,
              MaterialPageRoute(
                builder: (context) => RoleSelectionPage(userId: userId),
              ),
            );
          } else {
            Navigator.pushReplacementNamed(context, '/main-tab');
          }
        } else {
          throw Exception(response['message'] ?? '登录失败');
```

```
      }
    } catch (e) {
      if (!mounted) return;
      ScaffoldMessenger.of(context).showSnackBar(
        SnackBar(content: Text(e.toString())),
      );
    } finally {
      if (mounted) {
        setState(() {
          _isLoading = false;
        });
      }
    }
  }
}

@override
Widget build(BuildContext context) {
  return Scaffold(
    backgroundColor: AppTheme.backgroundColor,
    body: SafeArea(
      child: SingleChildScrollView(
        padding: const EdgeInsets.all(20),
        child: Column(
          crossAxisAlignment: CrossAxisAlignment.start,
          children: [
            const SizedBox(height: 40),
            const Center(
              child: Text(
                '好会帮',
                style: TextStyle(
                  color: AppTheme.primaryColor,
                  fontSize: 40,
                  fontWeight: FontWeight.bold,
                  letterSpacing: -1,
                ),
              ),
            ),
            const SizedBox(height: 8),
            const Center(
              child: Text(
                '上好会帮，赢好未来',
                style: TextStyle(
                  color: Colors.grey,
                  fontSize: 12,
                  letterSpacing: 1,
                ),
              ),
            ),
            const SizedBox(height: 40),
            Form(
              key: _formKey,
              child: Column(
                children: [
                  TextFormField(
                    controller: _phoneController,
                    keyboardType: TextInputType.phone,
                    decoration: const InputDecoration(
                      labelText: '手机号',
                      hintText: '请输入手机号',
```

```
                    prefixIcon: Icon(Icons.phone_rounded),
                  ),
                  validator: (value) {
                    if (value == null || value.isEmpty) {
                      return '请输入手机号';
                    }
                    if (!RegExp(r'^1[3-9]\d{9}$').hasMatch(value)) {
                      return '请输入正确的手机号';
                    }
                    return null;
                  },
                ),
                const SizedBox(height: 16),
                TextFormField(
                  controller: _passwordController,
                  obscureText: _obscurePassword,
                  decoration: InputDecoration(
                    labelText: '密码',
                    hintText: '请输入密码',
                    prefixIcon: const Icon(Icons.lock_rounded),
                    suffixIcon: IconButton(
                      icon: Icon(
                        _obscurePassword ? Icons.visibility_off : Icons.visibility,
                      ),
                      onPressed: () {
                        setState(() {
                          _obscurePassword = !_obscurePassword;
                        });
                      },
                    ),
                  ),
                  validator: (value) {
                    if (value == null || value.isEmpty) {
                      return '请输入密码';
                    }
                    if (value.length < 6) {
                      return '密码长度不能少于 6 位';
                    }
                    return null;
                  },
                ),
                const SizedBox(height: 24),
                SizedBox(
                  width: double.infinity,
                  height: 48,
                  child: ElevatedButton(
                    onPressed: (_isLoading || !_isFormValid) ? null : _handleLogin,
                    child: _isLoading
                        ? const SizedBox(
                            width: 20,
                            height: 20,
                            child: CircularProgressIndicator(color: Colors.white, s
trokeWidth: 2),
                          )
                        : const Text('登录'),
                  ),
                ),
                const SizedBox(height: 16),
                Row(
                  mainAxisAlignment: MainAxisAlignment.spaceBetween,
```

```
                    children: [
                      TextButton(
                        onPressed: () {
                          Navigator.pushReplacementNamed(context, '/sms-login');
                        },
                        child: const Text(
                          '验证码登录/注册',
                          style: TextStyle(
                            fontSize: 13,
                            color: AppTheme.primaryColor,
                            fontWeight: FontWeight.w600,
                          ),
                        ),
                      ),
                      TextButton(
                        onPressed: () {
                          Navigator.pushNamed(context, '/forgot-password');
                        },
                        child: const Text(
                          '忘记密码？',
                          style: TextStyle(
                            fontSize: 13,
                            color: AppTheme.primaryColor,
                            fontWeight: FontWeight.w600,
                          ),
                        ),
                      ),
                    ],
                  ),
                ],
              ),
            ),
          ),
        ),
      ),
    );
  }
}
```

二、后端源代码

2.1 应用入口文件

应用入口文件是 Spring Boot 应用的启动类，文件位于 `hitutor-backend/src/main/java/com/hitutor/HiTutorApplication.java`。HiTutorApplication 类使用了 @SpringBootApplication 注解，表示这是一个 Spring Boot 应用，会自动进行组件扫描和配置。类中还使用了 @ComponentScan 注解，指定组件扫描的基础包为 com.hitutor，确保 Spring 能够扫描到所有的组件。

main 方法是应用的入口方法，接收命令行参数 args，调用 SpringApplication 的 run 方法启动 Spring Boot 应用，传入 HiTutorApplication 类和命令行参数。run 方法会启动 Spring Boot 应用，加载所有的配置和组件，启动内嵌的 Tomcat 服务器，监听 HTTP 请求。

```
package com.hitutor;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.annotation.ComponentScan;

@SpringBootApplication
```

```
@ComponentScan(basePackages = "com.hitutor")
public class HiTutorApplication {
    public static void main(String[] args) {
        SpringApplication.run(HiTutorApplication.class, args);
    }
}
```

8.4 用户资料页面

用户资料页面是查看用户详细信息的主要界面，文件位于 client/lib/pages/user/user_profile_page.dart。UserProfilePage 类是一个有状态组件，使用 StatefulWidget 实现。类中定义了私有变量：*isLoading 加载状态*，userData 用户数据，*userServices 用户服务列表*，userRequests 用户需求列表，*errorMessage 错误信息*，isBlocked 是否已拉黑。

UserProfilePage 类的主要功能包括：加载用户数据；加载用户服务列表；加载用户需求列表；检查是否已拉黑；发起聊天；拉黑用户；查看服务详情；查看需求详情。页面使用了 Provider 组件管理用户认证状态，使用 Navigator 组件进行页面跳转。

UserProfilePage 类使用了 Scaffold 组件构建页面框架，使用 AppBar 组件显示标题栏，使用 ListView 组件显示用户信息，使用 Card 组件显示服务列表，使用 ElevatedButton 组件显示操作按钮。页面还使用了 FutureBuilder 组件处理异步数据加载，使用 GestureDetector 组件处理点击事件。

```
import 'package:flutter/material.dart';
import 'package:provider/provider.dart';
import '../../theme/app_theme.dart';
import '../../services/api_service.dart';
import '../../providers/auth_provider.dart';
import '../../utils/error_handler.dart';
import '../../models/tutor_model.dart';
import '../../routes.dart';
import '../tutor_service/tutor_service_detail_page.dart';
import '../student_request/student_request_detail_page.dart';

class UserProfilePage extends StatefulWidget {
  final String userId;
  final String userName;

  const UserProfilePage({
    super.key,
    required this.userId,
    required this.userName,
  });

  @override
  State<UserProfilePage> createState() => _UserProfilePageState();
}

class _UserProfilePageState extends State<UserProfilePage> {
  bool _isLoading = true;
  Map<String, dynamic>? _userData;
  List<dynamic> _userServices = [];
  List<dynamic> _userRequests = [];
  String _errorMessage = '';
  bool _isBlocked = false;

  static const _padding16 = EdgeInsets.all(16);
  static const _padding12 = EdgeInsets.all(12);
  static final _borderRadius12 = BorderRadius.circular(12);
  static final _borderRadius16 = BorderRadius.circular(16);
```

```dart
  @override
  void initState() {
    super.initState();
    WidgetsBinding.instance.addPostFrameCallback((_) {
      _loadUserData();
    });
  }

  Future<void> _loadUserData() async {
    if (!mounted) return;

    setState(() {
      _isLoading = true;
      _errorMessage = '';
    });

    try {
      final authProvider = Provider.of<AuthProvider>(context, listen: false);

      if (authProvider.isAuthenticated && authProvider.user?.id != widget.userId) {
        try {
          final blockedResponse = await ApiService.checkBlocked(widget.userId);
          if (blockedResponse['success'] == true) {
            _isBlocked = blockedResponse['data'] ?? false;
          }
        } catch (e) {
        }
      }

      final userResponse = await ApiService.getUser(widget.userId, needAuth: false);
      if (userResponse['success'] == true) {
        setState(() {
          _userData = userResponse['data'];
        });
      }

      final isOwnProfile = authProvider.user?.id == widget.userId;

      try {
        if (isOwnProfile) {
          final servicesResponse = await ApiService.getUserServices();
          if (servicesResponse['success'] == true) {
            final content = servicesResponse['data']?['content'] ?? [];
            setState(() {
              _userServices = content;
            });
          }

          final requestsResponse = await ApiService.getUserRequests();
          if (requestsResponse['success'] == true) {
            final content = requestsResponse['data']?['content'] ?? [];
            setState(() {
              _userRequests = content;
            });
          }
        } else {
          final servicesResponse = await ApiService.getUserServicesByUserId(widget.user
Id, needAuth: false);
          if (servicesResponse['success'] == true) {
            final content = servicesResponse['data']?['content'] ?? [];
            setState(() {
```

```
            _userServices = content;
          });
        }

        final requestsResponse = await ApiService.getUserRequestsByUserId(widget.user
Id, needAuth: false);
        if (requestsResponse['success'] == true) {
          final content = requestsResponse['data']?['content'] ?? [];
          setState(() {
            _userRequests = content;
          });
        }
      }
    } catch (e) {
    }
  } catch (e) {
    setState(() {
      _errorMessage = e.toString();
    });
  } finally {
    if (mounted) {
      setState(() {
        _isLoading = false;
      });
    }
  }
}

Future<void> _startChat() async {
  try {
    final authProvider = Provider.of<AuthProvider>(context, listen: false);
    final userId = authProvider.user?.id;

    if (userId == null) {
      if (mounted) {
        ScaffoldMessenger.of(context).showSnackBar(
          const SnackBar(content: Text('请先登录')),
        );
      }
      return;
    }

    final response = await ApiService.createConversation(userId, widget.userId);
    if (response['success'] == true) {
      final conversationId = response['data']['id']?.toString() ?? '';
      if (mounted && conversationId.isNotEmpty) {
        Navigator.pushNamed(
          context,
          Routes.chatDetail,
          arguments: {
            'conversationId': conversationId,
            'otherUserId': widget.userId,
            'otherUserName': widget.userName,
          },
        );
      }
    }
  } catch (e) {
    if (mounted) {
      ScaffoldMessenger.of(context).showSnackBar(
        SnackBar(content: Text('发起聊天失败: ${e.toString()}')),
```

```
      );
    }
  }
}

Future<void> _blockUser() async {
  if (!mounted) return;

  final confirmed = await showDialog<bool>(
    context: context,
    builder: (context) => AlertDialog(
      title: const Text('确认拉黑'),
      content: Text('确定要拉黑${widget.userName}吗？'),
      actions: [
        TextButton(
          onPressed: () => Navigator.pop(context, false),
          child: const Text('取消'),
        ),
        TextButton(
          onPressed: () => Navigator.pop(context, true),
          child: const Text('确定'),
        ),
      ],
    ),
  );

  if (confirmed != true) return;

  try {
    final response = await ApiService.blockUser(widget.userId);
    if (response['success'] == true) {
      if (mounted) {
        setState(() {
          _isBlocked = true;
        });
        ScaffoldMessenger.of(context).showSnackBar(
          const SnackBar(content: Text('拉黑成功')),
        );
      }
    }
  } catch (e) {
    if (mounted) {
      ScaffoldMessenger.of(context).showSnackBar(
        SnackBar(content: Text('拉黑失败: ${e.toString()}')),
      );
    }
  }
}

@override
Widget build(BuildContext context) {
  return Scaffold(
    backgroundColor: AppTheme.backgroundColor,
    appBar: AppBar(
      title: Text(widget.userName),
      backgroundColor: Colors.white,
      foregroundColor: AppTheme.textPrimary,
      elevation: 0,
    ),
    body: _isLoading
```

```
        ? const Center(
            child: CircularProgressIndicator(color: AppTheme.primaryColor),
          )
        : _errorMessage.isNotEmpty
            ? Center(
                child: Column(
                  mainAxisAlignment: MainAxisAlignment.center,
                  children: [
                    const Icon(
                      Icons.error_outline,
                      size: 48,
                      color: AppTheme.errorColor,
                    ),
                    const SizedBox(height: 16),
                    Text(
                      _errorMessage,
                      style: const TextStyle(
                        fontSize: 14,
                        color: AppTheme.textSecondary,
                      ),
                      textAlign: TextAlign.center,
                    ),
                    const SizedBox(height: 16),
                    ElevatedButton(
                      onPressed: _loadUserData,
                      child: const Text('重试'),
                    ),
                  ],
                ),
              )
            : SingleChildScrollView(
                child: Column(
                  crossAxisAlignment: CrossAxisAlignment.start,
                  children: [
                    _buildUserInfo(),
                    if (_userServices.isNotEmpty) _buildServicesList(),
                    if (_userRequests.isNotEmpty) _buildRequestsList(),
                  ],
                ),
              ),
  );
}

Widget _buildUserInfo() {
  if (_userData == null) return const SizedBox();

  final authProvider = Provider.of<AuthProvider>(context);
  final isOwnProfile = authProvider.user?.id == widget.userId;

  return Container(
    color: Colors.white,
    padding: _padding16,
    child: Column(
      crossAxisAlignment: CrossAxisAlignment.start,
      children: [
        Row(
          children: [
            CircleAvatar(
              radius: 40,
              backgroundColor: AppTheme.primaryColor.withOpacity(0.1),
              child: Text(
```

```
                widget.userName.substring(0, 1),
                style: const TextStyle(
                  fontSize: 32,
                  fontWeight: FontWeight.bold,
                  color: AppTheme.primaryColor,
                ),
              ),
            ),
            const SizedBox(width: 16),
            Expanded(
              child: Column(
                crossAxisAlignment: CrossAxisAlignment.start,
                children: [
                  Text(
                    widget.userName,
                    style: const TextStyle(
                      fontSize: 20,
                      fontWeight: FontWeight.bold,
                      color: AppTheme.textPrimary,
                    ),
                  ),
                  const SizedBox(height: 4),
                  if (_userData!['role'] != null)
                    Text(
                      _userData!['role'] == 'tutor' ? '家教老师' : '学生',
                      style: const TextStyle(
                        fontSize: 14,
                        color: AppTheme.textSecondary,
                      ),
                    ),
                ],
              ),
            ),
          ],
        ),
        const SizedBox(height: 16),
        if (_userData!['education'] != null)
          _buildInfoRow(Icons.school, '学历', _userData!['education']),
        if (_userData!['school'] != null)
          _buildInfoRow(Icons.location_city, '学校', _userData!['school']),
        if (_userData!['major'] != null)
          _buildInfoRow(Icons.book, '专业', _userData!['major']),
        if (_userData!['teachingExperience'] != null)
          _buildInfoRow(Icons.work, '教学经验', '${_userData!['teachingExperience']}年
'),
        const SizedBox(height: 16),
        if (!isOwnProfile)
          Row(
            children: [
              Expanded(
                child: ElevatedButton(
                  onPressed: _isBlocked ? null : _startChat,
                  style: ElevatedButton.styleFrom(
                    backgroundColor: AppTheme.primaryColor,
                    padding: const EdgeInsets.symmetric(vertical: 12),
                  ),
                  child: const Text('发起聊天'),
                ),
              ),
              const SizedBox(width: 12),
```

```
              Expanded(
                child: OutlinedButton(
                  onPressed: _isBlocked ? null : _blockUser,
                  style: OutlinedButton.styleFrom(
                    foregroundColor: AppTheme.errorColor,
                    side: const BorderSide(color: AppTheme.errorColor),
                    padding: const EdgeInsets.symmetric(vertical: 12),
                  ),
                  child: Text(_isBlocked ? '已拉黑' : '拉黑'),
                ),
              ),
            ],
          ),
        ],
      ),
    );
  }

  Widget _buildInfoRow(IconData icon, String label, String value) {
    return Padding(
      padding: const EdgeInsets.only(bottom: 8),
      child: Row(
        children: [
          Icon(icon, size: 18, color: AppTheme.textTertiary),
          const SizedBox(width: 8),
          Text(
            '$label: ',
            style: const TextStyle(
              fontSize: 14,
              color: AppTheme.textSecondary,
            ),
          ),
          const SizedBox(width: 4),
          Expanded(
            child: Text(
              value,
              style: const TextStyle(
                fontSize: 14,
                color: AppTheme.textPrimary,
              ),
            ),
          ),
        ],
      ),
    );
  }

  Widget _buildServicesList() {
    return Container(
      margin: const EdgeInsets.only(top: 8),
      color: Colors.white,
      padding: _padding16,
      child: Column(
        crossAxisAlignment: CrossAxisAlignment.start,
        children: [
          const Text(
            '家教服务',
            style: TextStyle(
              fontSize: 16,
              fontWeight: FontWeight.bold,
```

```
              color: AppTheme.textPrimary,
            ),
          ),
          const SizedBox(height: 12),
          ..._userServices.map((service) => _buildServiceCard(service)),
        ],
      ),
    );
  }

  Widget _buildServiceCard(dynamic service) {
    return Card(
      margin: const EdgeInsets.only(bottom: 12),
      shape: RoundedRectangleBorder(
        borderRadius: _borderRadius12,
      ),
      child: InkWell(
        onTap: () {
          Navigator.push(
            context,
            MaterialPageRoute(
              builder: (context) => TutorServiceDetailPage(service: service),
            ),
          );
        },
        borderRadius: _borderRadius12,
        child: Padding(
          padding: _padding12,
          child: Column(
            crossAxisAlignment: CrossAxisAlignment.start,
            children: [
              Text(
                service['subjectName'] ?? '',
                style: const TextStyle(
                  fontSize: 16,
                  fontWeight: FontWeight.bold,
                  color: AppTheme.textPrimary,
                ),
              ),
              const SizedBox(height: 8),
              Row(
                children: [
                  const Icon(Icons.attach_money, size: 16, color: AppTheme.textTertiar
y),
                  const SizedBox(width: 4),
                  Text(
                    '${service['hourlyRate']}元/小时',
                    style: const TextStyle(
                      fontSize: 14,
                      color: AppTheme.textSecondary,
                    ),
                  ),
                ],
              ),
            ],
          ),
        ),
      ),
    );
  }
```

```
Widget _buildRequestsList() {
  return Container(
    margin: const EdgeInsets.only(top: 8),
    color: Colors.white,
    padding: _padding16,
    child: Column(
      crossAxisAlignment: CrossAxisAlignment.start,
      children: [
        const Text(
          '学生需求',
          style: TextStyle(
            fontSize: 16,
            fontWeight: FontWeight.bold,
            color: AppTheme.textPrimary,
          ),
        ),
        const SizedBox(height: 12),
        ..._userRequests.map((request) => _buildRequestCard(request)),
      ],
    ),
  );
}

Widget _buildRequestCard(dynamic request) {
  return Card(
    margin: const EdgeInsets.only(bottom: 12),
    shape: RoundedRectangleBorder(
      borderRadius: _borderRadius12,
    ),
    child: InkWell(
      onTap: () {
        Navigator.push(
          context,
          MaterialPageRoute(
            builder: (context) => StudentRequestDetailPage(request: request),
          ),
        );
      },
      borderRadius: _borderRadius12,
      child: Padding(
        padding: _padding12,
        child: Column(
          crossAxisAlignment: CrossAxisAlignment.start,
          children: [
            Text(
              request['subjectName'] ?? '',
              style: const TextStyle(
                fontSize: 16,
                fontWeight: FontWeight.bold,
                color: AppTheme.textPrimary,
              ),
            ),
            const SizedBox(height: 8),
            Text(
              request['description'] ?? '',
              style: const TextStyle(
                fontSize: 14,
                color: AppTheme.textSecondary,
              ),
              maxLines: 2,
              overflow: TextOverflow.ellipsis,
```

```
            ),
          ],
        ),
      ),
    ),
  );
 }
}
```

9.3 预约控制器

预约控制器负责处理预约相关的 HTTP 请求，文件位于 `hitutor-backend/src/main/java/com/hitutor/controller/AppointmentController.java`。AppointmentController 类使用@RestController 注解标记为 REST 控制器，使用@RequestMapping 注解指定基础路径为/api/appointments。类中定义了两个自动注入的依赖：appointmentService 预约服务，userService 用户服务。

AppointmentController 类的主要功能包括：根据用户 ID 获取预约列表；根据家教 ID 获取预约列表；根据 ID 获取预约详情；创建新的预约；更新预约信息；删除预约；确认预约；取消预约；完成预约。控制器使用了 ResponseEntity 返回统一的响应格式，使用 HttpStatus 返回不同的 HTTP 状态码。

AppointmentController 类使用了@GetMapping 注解处理 GET 请求，使用@PostMapping 注解处理 POST 请求，使用@PutMapping 注解处理 PUT 请求，使用@DeleteMapping 注解处理 DELETE 请求，使用@PathVariable 注解获取路径参数，使用@RequestBody 注解获取请求体。控制器还使用了 DtoConverter 工具类将实体类转换为 DTO 对象，使用 Stream API 进行数据转换和处理。

```
package com.hitutor.controller;

import com.hitutor.dto.AppointmentDTO;
import com.hitutor.entity.Appointment;
import com.hitutor.entity.User;
import com.hitutor.service.AppointmentService;
import com.hitutor.service.UserService;
import com.hitutor.util.DtoConverter;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.Set;
import java.util.stream.Collectors;

@RestController
@RequestMapping("/api/appointments")
public class AppointmentController {

    @Autowired
    private AppointmentService appointmentService;

    @Autowired
    private UserService userService;

    @GetMapping("/user/{userId}")
    public ResponseEntity<Map<String, Object>> getAppointmentsByUserId(@PathVariable St
ring userId) {
        if ("all".equals(userId)) {
```

```
            List<Appointment> appointments = appointmentService.getAllAppointments();

            Set<String> userIds = appointments.stream()
                    .flatMap(appointment -> List.of(appointment.getTutorId(), appointme
nt.getStudentId()).stream())
                    .collect(Collectors.toSet());

            List<User> users = userService.getUsersByIds(userIds.stream().toList());
            Map<String, User> userMap = users.stream()
                    .collect(Collectors.toMap(User::getId, user -> user));

            List<AppointmentDTO> appointmentDTOs = appointments.stream()
                    .map(appointment -> {
                        User tutor = userMap.get(appointment.getTutorId());
                        User student = userMap.get(appointment.getStudentId());
                        return DtoConverter.toAppointmentDTO(appointment, tutor, studen
t);
                    })
                    .collect(Collectors.toList());
            Map<String, Object> response = new HashMap<>();
            response.put("success", true);
            response.put("message", "获取预约列表成功");
            response.put("data", appointmentDTOs);
            return ResponseEntity.ok(response);
        }

        List<Appointment> appointments = appointmentService.getAppointmentsByUserId(use
rId);

        Set<String> userIds = appointments.stream()
                .flatMap(appointment -> List.of(appointment.getTutorId(), appointment.g
etStudentId()).stream())
                .collect(Collectors.toSet());

        List<User> users = userService.getUsersByIds(userIds.stream().toList());
        Map<String, User> userMap = users.stream()
                .collect(Collectors.toMap(User::getId, user -> user));

        List<AppointmentDTO> appointmentDTOs = appointments.stream()
                .map(appointment -> {
                    User tutor = userMap.get(appointment.getTutorId());
                    User student = userMap.get(appointment.getStudentId());
                    return DtoConverter.toAppointmentDTO(appointment, tutor, student);
                })
                .collect(Collectors.toList());
        Map<String, Object> response = new HashMap<>();
        response.put("success", true);
        response.put("message", "获取用户预约列表成功");
        response.put("data", appointmentDTOs);
        return ResponseEntity.ok(response);
    }

    @GetMapping("/tutor/{tutorId}")
    public ResponseEntity<Map<String, Object>> getAppointmentsByTutorId(@PathVariable S
tring tutorId) {
        List<Appointment> appointments = appointmentService.getAppointmentsByTutorId(tu
torId);

        Set<String> userIds = appointments.stream()
                .flatMap(appointment -> List.of(appointment.getTutorId(), appointment.g
```

```
etStudentId()).stream())
                .collect(Collectors.toSet());

        List<User> users = userService.getUsersByIds(userIds.stream().toList());
        Map<String, User> userMap = users.stream()
                .collect(Collectors.toMap(User::getId, user -> user));

        List<AppointmentDTO> appointmentDTOs = appointments.stream()
                .map(appointment -> {
                    User tutor = userMap.get(appointment.getTutorId());
                    User student = userMap.get(appointment.getStudentId());
                    return DtoConverter.toAppointmentDTO(appointment, tutor, student);
                })
                .collect(Collectors.toList());
        Map<String, Object> response = new HashMap<>();
        response.put("success", true);
        response.put("message", "获取家教预约列表成功");
        response.put("data", appointmentDTOs);
        return ResponseEntity.ok(response);
    }

    @GetMapping("/{id}")
    public ResponseEntity<Map<String, Object>> getAppointmentById(@PathVariable Long id)
 {
        Appointment appointment = appointmentService.getAppointmentById(id);
        if (appointment == null) {
            Map<String, Object> response = new HashMap<>();
            response.put("success", false);
            response.put("message", "预约不存在");
            return ResponseEntity.status(HttpStatus.NOT_FOUND).body(response);
        }
        User tutor = userService.getUserById(appointment.getTutorId());
        User student = userService.getUserById(appointment.getStudentId());
        Map<String, Object> response = new HashMap<>();
        response.put("success", true);
        response.put("message", "获取预约详情成功");
        response.put("data", DtoConverter.toAppointmentDTO(appointment, tutor, studen
t));
        return ResponseEntity.ok(response);
    }

    @PostMapping
    public ResponseEntity<Map<String, Object>> createAppointment(@RequestBody Appointme
nt appointment) {
        if (appointment.getTutorId() == null || appointment.getTutorId().isEmpty()) {
            Map<String, Object> response = new HashMap<>();
            response.put("success", false);
            response.put("message", "家教 ID 不能为空");
            return ResponseEntity.badRequest().body(response);
        }

        if (appointment.getStudentId() == null || appointment.getStudentId().isEmpty())
 {
            Map<String, Object> response = new HashMap<>();
            response.put("success", false);
            response.put("message", "学生 ID 不能为空");
            return ResponseEntity.badRequest().body(response);
        }

        User tutor = userService.getUserById(appointment.getTutorId());
```

```
        User student = userService.getUserById(appointment.getStudentId());

        if (tutor == null) {
            Map<String, Object> response = new HashMap<>();
            response.put("success", false);
            response.put("message", "家教不存在");
            return ResponseEntity.badRequest().body(response);
        }

        if (student == null) {
            Map<String, Object> response = new HashMap<>();
            response.put("success", false);
            response.put("message", "学生不存在");
            return ResponseEntity.badRequest().body(response);
        }

        if (!"active".equals(tutor.getStatus())) {
            Map<String, Object> response = new HashMap<>();
            response.put("success", false);
            response.put("message", "家教账号已被禁用，无法创建预约");
            return ResponseEntity.status(HttpStatus.FORBIDDEN).body(response);
        }

        if (!"active".equals(student.getStatus())) {
            Map<String, Object> response = new HashMap<>();
            response.put("success", false);
            response.put("message", "学生账号已被禁用，无法创建预约");
            return ResponseEntity.status(HttpStatus.FORBIDDEN).body(response);
        }

        boolean saved = appointmentService.saveAppointment(appointment);
        if (!saved) {
            Map<String, Object> response = new HashMap<>();
            response.put("success", false);
            response.put("message", "创建预约失败");
            return ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR).body(respons
e);
        }
        Map<String, Object> response = new HashMap<>();
        response.put("success", true);
        response.put("message", "创建预约成功");
        response.put("data", DtoConverter.toAppointmentDTO(appointment, tutor, studen
t));
        return ResponseEntity.status(HttpStatus.CREATED).body(response);
    }

    @PutMapping("/{id}")
    public ResponseEntity<Map<String, Object>> updateAppointment(@PathVariable Long id,
 @RequestBody Appointment appointment) {
        appointment.setId(id);
        boolean updated = appointmentService.updateAppointment(appointment);
        if (!updated) {
            Map<String, Object> response = new HashMap<>();
            response.put("success", false);
            response.put("message", "预约不存在");
            return ResponseEntity.status(HttpStatus.NOT_FOUND).body(response);
        }
        User tutor = userService.getUserById(appointment.getTutorId());
        User student = userService.getUserById(appointment.getStudentId());
        Map<String, Object> response = new HashMap<>();
```

```
        response.put("success", true);
        response.put("message", "更新预约成功");
        response.put("data", DtoConverter.toAppointmentDTO(appointment, tutor, studen
t));
        return ResponseEntity.ok(response);
    }

    @DeleteMapping("/{id}")
    public ResponseEntity<Map<String, Object>> deleteAppointment(@PathVariable Long id)
 {
        boolean deleted = appointmentService.deleteAppointment(id);
        if (!deleted) {
            Map<String, Object> response = new HashMap<>();
            response.put("success", false);
            response.put("message", "预约不存在");
            return ResponseEntity.status(HttpStatus.NOT_FOUND).body(response);
        }
        Map<String, Object> response = new HashMap<>();
        response.put("success", true);
        response.put("message", "删除预约成功");
        return ResponseEntity.ok(response);
    }

    @PutMapping("/{id}/confirm")
    public ResponseEntity<Map<String, Object>> confirmAppointment(@PathVariable Long id)
 {
        boolean confirmed = appointmentService.confirmAppointment(id);
        if (!confirmed) {
            return ResponseEntity.notFound().build();
        }
        Map<String, Object> result = new HashMap<>();
        result.put("success", true);
        result.put("message", "预约已确认");
        return ResponseEntity.ok(result);
    }

    @PutMapping("/{id}/cancel")
    public ResponseEntity<Map<String, Object>> cancelAppointment(@PathVariable Long id)
 {
        boolean cancelled = appointmentService.cancelAppointment(id);
        if (!cancelled) {
            return ResponseEntity.notFound().build();
        }
        Map<String, Object> result = new HashMap<>();
        result.put("success", true);
        result.put("message", "预约已取消");
        return ResponseEntity.ok(result);
    }

    @PutMapping("/{id}/complete")
    public ResponseEntity<Map<String, Object>> completeAppointment(@PathVariable Long i
d) {
        boolean completed = appointmentService.completeAppointment(id);
        if (!completed) {
            return ResponseEntity.notFound().build();
        }
        Map<String, Object> result = new HashMap<>();
        result.put("success", true);
        result.put("message", "预约已完成");
        return ResponseEntity.ok(result);
```

```
    }
}
```

8.3 聊天详情页面

聊天详情页面是用户进行实时聊天的主要界面，文件位于 `client/lib/pages/chat/chat_detail_page.dart`。ChatDetailPage 类是一个有状态组件，使用 StatefulWidget 实现。类中定义了私有变量：*messageController 消息输入控制器*，scrollController 滚动控制器，*messages 消息列表*，isLoading 加载状态。

ChatDetailPage 类的主要功能包括：加载历史消息；发送新消息；显示消息列表；区分自己和他人的消息；自动滚动到底部；支持文字消息发送。页面使用了 Provider 组件管理用户认证状态，使用 Navigator 组件进行页面跳转。

ChatDetailPage 类使用了 Scaffold 组件构建页面框架，使用 AppBar 组件显示标题栏，使用 ListView 组件显示消息列表，使用 TextField 组件显示消息输入框，使用 ElevatedButton 组件显示发送按钮。页面还使用了 ScrollController 控制滚动位置，使用 TextEditingController 控制输入框内容。

```dart
import 'package:flutter/material.dart';
import 'package:provider/provider.dart';
import '../../theme/app_theme.dart';
import '../../providers/auth_provider.dart';
import '../../models/message_model.dart';
import '../../services/api_service.dart';

class ChatDetailPage extends StatefulWidget {
  final String conversationId;
  final String otherUserId;
  final String otherUserName;

  const ChatDetailPage({
    super.key,
    required this.conversationId,
    required this.otherUserId,
    required this.otherUserName,
  });

  @override
  State<ChatDetailPage> createState() => _ChatDetailPageState();
}

class _ChatDetailPageState extends State<ChatDetailPage> {
  final TextEditingController _messageController = TextEditingController();
  final ScrollController _scrollController = ScrollController();
  List<Message> _messages = [];
  bool _isLoading = false;

  @override
  void initState() {
    super.initState();
    _loadMessages();
  }

  @override
  void dispose() {
    _messageController.dispose();
    _scrollController.dispose();
    super.dispose();
  }
```

```dart
Future<void> _loadMessages() async {
  setState(() {
    _isLoading = true;
  });

  try {
    final authProvider = Provider.of<AuthProvider>(context, listen: false);
    final userId = authProvider.user?.id;

    if (userId == null) return;

    final response = await ApiService.getMessages(widget.conversationId);

    if (response['success'] == true || response is List) {
      final data = response['data'] ?? response;
      if (data is List) {
        setState(() {
          _messages = data.map((item) => Message.fromJson(item)).toList();
        });
        _scrollToBottom();
      }
    }
  } catch (e) {
    if (mounted) {
      ScaffoldMessenger.of(context).showSnackBar(
        SnackBar(content: Text('加载消息失败: ${e.toString()}')),
      );
    }
  } finally {
    setState(() {
      _isLoading = false;
    });
  }
}

Future<void> _sendMessage() async {
  final content = _messageController.text.trim();
  if (content.isEmpty) return;

  setState(() {
    _isLoading = true;
  });

  try {
    final authProvider = Provider.of<AuthProvider>(context, listen: false);
    final userId = authProvider.user?.id;

    if (userId == null) return;

    final response = await ApiService.sendMessage(
      widget.conversationId,
      userId,
      widget.otherUserId,
      content,
    );

    if (response['success'] == true || response['data'] != null) {
      final newMessage = Message.fromJson(response['data']);
      setState(() {
        _messages.add(newMessage);
```

```
        _messageController.clear();
      });
      _scrollToBottom();
    }
  } catch (e) {
    if (mounted) {
      ScaffoldMessenger.of(context).showSnackBar(
        SnackBar(content: Text('发送失败: ${e.toString()}')),
      );
    }
  } finally {
    setState(() {
      _isLoading = false;
    });
  }
}

void _scrollToBottom() {
  WidgetsBinding.instance.addPostFrameCallback((_) {
    if (_scrollController.hasClients) {
      _scrollController.animateTo(
        _scrollController.position.maxScrollExtent,
        duration: const Duration(milliseconds: 300),
        curve: Curves.easeOut,
      );
    }
  });
}

bool _isMyMessage(Message message) {
  final authProvider = Provider.of<AuthProvider>(context, listen: false);
  return message.sender.id == authProvider.user?.id;
}

@override
Widget build(BuildContext context) {
  return Scaffold(
    backgroundColor: AppTheme.backgroundColor,
    appBar: AppBar(
      title: Text(widget.otherUserName),
      backgroundColor: Colors.white,
      foregroundColor: AppTheme.textPrimary,
      elevation: 0,
    ),
    body: Column(
      children: [
        Expanded(
          child: _isLoading && _messages.isEmpty
              ? const Center(
                  child: CircularProgressIndicator(color: AppTheme.primaryColor),
                )
              : ListView.builder(
                  controller: _scrollController,
                  padding: const EdgeInsets.all(16),
                  itemCount: _messages.length,
                  itemBuilder: (context, index) {
                    final message = _messages[index];
                    final isMyMessage = _isMyMessage(message);
                    return _buildMessageBubble(message, isMyMessage);
                  },
                ),
```

```
        ),
        _buildMessageInput(),
      ],
    ),
  );
}

Widget _buildMessageBubble(Message message, bool isMyMessage) {
  return Align(
    alignment: isMyMessage ? Alignment.centerRight : Alignment.centerLeft,
    child: Container(
      margin: const EdgeInsets.symmetric(vertical: 4),
      padding: const EdgeInsets.symmetric(horizontal: 16, vertical: 10),
      decoration: BoxDecoration(
        color: isMyMessage ? AppTheme.primaryColor : Colors.white,
        borderRadius: BorderRadius.circular(12),
        boxShadow: [
          BoxShadow(
            color: Colors.black.withOpacity(0.05),
            blurRadius: 4,
            offset: const Offset(0, 2),
          ),
        ],
      ),
      constraints: const BoxConstraints(
        maxWidth: 280,
      ),
      child: Column(
        crossAxisAlignment: CrossAxisAlignment.start,
        children: [
          Text(
            message.content,
            style: TextStyle(
              color: isMyMessage ? Colors.white : AppTheme.textPrimary,
              fontSize: 14,
            ),
          ),
          const SizedBox(height: 4),
          Text(
            _formatTime(message.createdAt),
            style: TextStyle(
              color: isMyMessage ? Colors.white70 : AppTheme.textTertiary,
              fontSize: 11,
            ),
          ),
        ],
      ),
    ),
  );
}

Widget _buildMessageInput() {
  return Container(
    padding: const EdgeInsets.all(16),
    decoration: BoxDecoration(
      color: Colors.white,
      boxShadow: [
        BoxShadow(
          color: Colors.black.withOpacity(0.05),
          blurRadius: 4,
          offset: const Offset(0, -2),
```

```
        ),
      ],
    ),
    child: Row(
      children: [
        Expanded(
          child: TextField(
            controller: _messageController,
            decoration: InputDecoration(
              hintText: '输入消息...',
              border: OutlineInputBorder(
                borderRadius: BorderRadius.circular(24),
                borderSide: BorderSide.none,
              ),
              filled: true,
              fillColor: AppTheme.backgroundColor,
              contentPadding: const EdgeInsets.symmetric(
                horizontal: 16,
                vertical: 12,
              ),
            ),
            maxLines: null,
            textInputAction: TextInputAction.send,
            onSubmitted: (_) => _sendMessage(),
          ),
        ),
        const SizedBox(width: 12),
        Container(
          width: 48,
          height: 48,
          decoration: BoxDecoration(
            color: AppTheme.primaryColor,
            borderRadius: BorderRadius.circular(24),
          ),
          child: IconButton(
            icon: const Icon(
              Icons.send,
              color: Colors.white,
            ),
            onPressed: _isLoading ? null : _sendMessage,
          ),
        ),
      ],
    ),
  );
}

String _formatTime(DateTime dateTime) {
  final now = DateTime.now();
  final difference = now.difference(dateTime);

  if (difference.inMinutes < 1) {
    return '刚刚';
  } else if (difference.inHours < 1) {
    return '${difference.inMinutes}分钟前';
  } else if (difference.inDays < 1) {
    return '${difference.inHours}小时前';
  } else if (difference.inDays < 7) {
    return '${difference.inDays}天前';
  } else {
```

```
      return '${dateTime.month}月${dateTime.day}日';
    }
  }
}
```

9.2 评价控制器

评价控制器负责处理评价相关的 HTTP 请求，文件位于 hitutor-backend/src/main/java/com/hitutor/controller/ReviewController.java。ReviewController 类使用@RestController 注解标记为 REST 控制器，使用@RequestMapping 注解指定基础路径为/api/reviews。类中定义了两个自动注入的依赖：reviewService 评价服务，userService 用户服务。

ReviewController 类的主要功能包括：根据家教 ID 获取评价列表；根据用户 ID 获取评价列表；创建新的评价；根据 ID 获取评价详情。控制器使用了 Spring Security 的 Authentication 对象进行用户认证，使用 ResponseEntity 返回统一的响应格式。

ReviewController 类使用了@GetMapping 注解处理 GET 请求，使用@PostMapping 注解处理 POST 请求，使用@PathVariable 注解获取路径参数，使用@RequestBody 注解获取请求体。控制器还使用了 DtoConverter 工具类将实体类转换为 DTO 对象，使用 Stream API 进行数据转换和处理。

```
package com.hitutor.controller;

import com.hitutor.dto.ReviewDTO;
import com.hitutor.entity.Review;
import com.hitutor.entity.User;
import com.hitutor.service.ReviewService;
import com.hitutor.service.UserService;
import com.hitutor.util.DtoConverter;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import org.springframework.security.core.Authentication;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.web.bind.annotation.*;

import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.stream.Collectors;

@RestController
@RequestMapping("/api/reviews")
public class ReviewController {

    @Autowired
    private ReviewService reviewService;

    @Autowired
    private UserService userService;

    @GetMapping("/tutor/{tutorId}")
    public ResponseEntity<Map<String, Object>> getReviewsByTutorId(@PathVariable String
 tutorId) {
        List<Review> reviews = reviewService.getReviewsByTutorId(tutorId);
        List<ReviewDTO> reviewDTOs = reviews.stream()
                .map(review -> {
                    User reviewer = userService.getUserById(review.getReviewerId());
                    return DtoConverter.toReviewDTO(review, reviewer);
                })
                .collect(Collectors.toList());
```

```
        Map<String, Object> result = new HashMap<>();
        result.put("success", true);
        result.put("message", "获取评价成功");
        result.put("data", reviewDTOs);
        return ResponseEntity.ok(result);
    }

    @GetMapping("/user/{userId}")
    public ResponseEntity<Map<String, Object>> getReviewsByUserId(@PathVariable String
userId) {
        List<Review> reviews = reviewService.getReviewsByUserId(userId);
        List<ReviewDTO> reviewDTOs = reviews.stream()
                .map(review -> {
                    User reviewer = userService.getUserById(review.getReviewerId());
                    return DtoConverter.toReviewDTO(review, reviewer);
                })
                .collect(Collectors.toList());
        Map<String, Object> result = new HashMap<>();
        result.put("success", true);
        result.put("message", "获取评价成功");
        result.put("data", reviewDTOs);
        return ResponseEntity.ok(result);
    }

    @PostMapping
    public ResponseEntity<Map<String, Object>> createReview(@RequestBody Map<String, Ob
ject> request) {
        Authentication authentication = SecurityContextHolder.getContext().getAuthentic
ation();
        String userId = authentication.getName();

        request.put("reviewerId", userId);

        Review review = reviewService.createReview(request);
        if (review == null) {
            Map<String, Object> result = new HashMap<>();
            result.put("success", false);
            result.put("message", "评价创建失败");
            return ResponseEntity.badRequest().body(result);
        }

        User reviewer = userService.getUserById(review.getReviewerId());
        ReviewDTO reviewDTO = DtoConverter.toReviewDTO(review, reviewer);

        Map<String, Object> result = new HashMap<>();
        result.put("success", true);
        result.put("data", reviewDTO);
        result.put("message", "评价成功");
        return ResponseEntity.ok(result);
    }

    @GetMapping("/{id}")
    public ResponseEntity<Map<String, Object>> getReviewById(@PathVariable Long id) {
        Review review = reviewService.getById(id);
        Map<String, Object> result = new HashMap<>();
        if (review == null) {
            result.put("success", false);
            result.put("message", "评价不存在");
            return ResponseEntity.status(404).body(result);
        }
```

```
        User reviewer = userService.getUserById(review.getReviewerId());
        result.put("success", true);
        result.put("message", "获取评价成功");
        result.put("data", DtoConverter.toReviewDTO(review, reviewer));
        return ResponseEntity.ok(result);
    }
}
```

2.2 认证控制器

认证控制器负责处理用户认证相关的请求，文件位于 hitutor-backend/src/main/java/com/hitutor/controller/AuthController.java。AuthController 类使用了@RestController 注解，表示这是一个控制器类，所有方法的返回值都会自动转换为 JSON 格式。类中还使用了@RequestMapping 注解，指定基础路径为/api/auth，所有方法的路径都会基于这个基础路径。

AuthController 类中注入了三个依赖：AuthService 认证服务，UserService 用户服务，JwtUtil JWT 工具类。类中定义了四个方法：loginWithPassword 处理密码登录请求，loginWithSms 处理短信验证码登录请求，register 处理注册请求，sendCode 处理发送验证码请求。

loginWithPassword 方法使用了@PostMapping 注解，指定路径为/login/password，接收 POST 请求。方法接收一个 Map 类型的请求体，从中提取手机号和密码，调用 AuthService 的 loginWithPassword 方法进行登录，如果登录成功，生成 JWT 令牌，构造响应数据，包括 success、message、data 字段，data 字段包含 token 和 user 信息，返回成功的响应，如果登录失败，构造失败的响应，返回错误的响应。loginWithSms 方法与 loginWithPassword 方法类似，处理短信验证码登录请求。register 方法处理注册请求，调用 AuthService 的 register 方法进行注册，逻辑与登录方法类似。sendCode 方法处理发送验证码请求，调用 AuthService 的 sendVerificationCode 方法发送验证码，构造响应数据，返回成功的响应。

```
package com.hitutor.controller;

import com.hitutor.dto.UserDTO;
import com.hitutor.entity.User;
import com.hitutor.service.AuthService;
import com.hitutor.service.UserService;
import com.hitutor.utils.JwtUtil;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

import java.util.HashMap;
import java.util.Map;

@RestController
@RequestMapping("/api/auth")
public class AuthController {

    @Autowired
    private AuthService authService;

    @Autowired
    private UserService userService;

    @Autowired
    private JwtUtil jwtUtil;

    @PostMapping("/login/password")
    public ResponseEntity<?> loginWithPassword(@RequestBody Map<String, String> request)
 {
        String phone = request.get("phone");
```

```
    String password = request.get("password");

    try {
        User user = authService.loginWithPassword(phone, password);
        String token = jwtUtil.generateToken(user.getId(), user.getRole());

        Map<String, Object> response = new HashMap<>();
        response.put("success", true);
        response.put("message", "登录成功");
        response.put("data", Map.of(
                "token", token,
                "user", UserDTO.fromUser(user)
        ));

        return ResponseEntity.ok(response);
    } catch (Exception e) {
        Map<String, Object> response = new HashMap<>();
        response.put("success", false);
        response.put("message", e.getMessage());
        response.put("data", null);

        return ResponseEntity.status(HttpStatus.BAD_REQUEST).body(response);
    }
}

@PostMapping("/login/sms")
public ResponseEntity<?> loginWithSms(@RequestBody Map<String, String> request) {
    String phone = request.get("phone");
    String code = request.get("code");

    try {
        User user = authService.loginWithSms(phone, code);
        String token = jwtUtil.generateToken(user.getId(), user.getRole());

        Map<String, Object> response = new HashMap<>();
        response.put("success", true);
        response.put("message", "登录成功");
        response.put("data", Map.of(
                "token", token,
                "user", UserDTO.fromUser(user)
        ));

        return ResponseEntity.ok(response);
    } catch (Exception e) {
        Map<String, Object> response = new HashMap<>();
        response.put("success", false);
        response.put("message", e.getMessage());
        response.put("data", null);

        return ResponseEntity.status(HttpStatus.BAD_REQUEST).body(response);
    }
}

@PostMapping("/register")
public ResponseEntity<?> register(@RequestBody Map<String, String> request) {
    String phone = request.get("phone");
    String password = request.get("password");
    String role = request.get("role");

    try {
```

```
            User user = authService.register(phone, password, role);
            String token = jwtUtil.generateToken(user.getId(), user.getRole());

            Map<String, Object> response = new HashMap<>();
            response.put("success", true);
            response.put("message", "注册成功");
            response.put("data", Map.of(
                    "token", token,
                    "user", UserDTO.fromUser(user)
            ));

            return ResponseEntity.ok(response);
        } catch (Exception e) {
            Map<String, Object> response = new HashMap<>();
            response.put("success", false);
            response.put("message", e.getMessage());
            response.put("data", null);

            return ResponseEntity.status(HttpStatus.BAD_REQUEST).body(response);
        }
    }

    @PostMapping("/send-code")
    public ResponseEntity<?> sendCode(@RequestBody Map<String, String> request) {
        String phone = request.get("phone");

        try {
            authService.sendVerificationCode(phone);

            Map<String, Object> response = new HashMap<>();
            response.put("success", true);
            response.put("message", "验证码发送成功");
            response.put("data", null);

            return ResponseEntity.ok(response);
        } catch (Exception e) {
            Map<String, Object> response = new HashMap<>();
            response.put("success", false);
            response.put("message", e.getMessage());
            response.put("data", null);

            return ResponseEntity.status(HttpStatus.BAD_REQUEST).body(response);
        }
    }
}
```

2.3 家教信息控制器

家教信息控制器负责处理家教信息相关的请求，文件位于 `hitutor-backend/src/main/java/com/hitutor/controller/TutorProfileController.java`。TutorProfileController 类使用了@RestController 注解，表示这是一个控制器类，所有方法的返回值都会自动转换为 JSON 格式。类中还使用了@RequestMapping 注解，指定基础路径为/api/tutor-profiles，所有方法的路径都会基于这个基础路径。

TutorProfileController 类中注入了两个依赖：TutorProfileService 家教信息服务，UserUtil 用户工具类。类中定义了六个方法：getTutorProfiles 获取家教信息列表，getMyTutorProfiles 获取我的家教信息列表，createTutorProfile 创建家教信息，getTutorProfile 获取家教信息详情，updateTutorProfile 更新家教信息，deleteTutorProfile 删除家教信息。

getTutorProfiles 方法使用了@GetMapping 注解，接收 GET 请求，方法接收 page、size、subjectId、grade、sortBy 等参数，调用 TutorProfileService 的 getTutorProfiles 方法获取家教信息列表，构造响应数据，返回成功的响应。getMyTutorProfiles 方法使用了@GetMapping 注解，路径为/my，调用 TutorProfileService 的 getMyTutorProfiles 方法获取我的家教信息列表，构造响应数据，返回成功的响应。createTutorProfile 方法使用了@PostMapping 注解，接收 POST 请求，调用 TutorProfileService 的 createTutorProfile 方法创建家教信息，构造响应数据，返回成功的响应。getTutorProfile 方法使用了@GetMapping 注解，路径为/{id}，调用 TutorProfileService 的 getTutorProfile 方法获取家教信息详情，构造响应数据，返回成功的响应。updateTutorProfile 方法使用了@PutMapping 注解，路径为/{id}，调用 TutorProfileService 的 updateTutorProfile 方法更新家教信息，构造响应数据，返回成功的响应。deleteTutorProfile 方法使用了@DeleteMapping 注解，路径为/{id}，调用 TutorProfileService 的 deleteTutorProfile 方法删除家教信息，构造响应数据，返回成功的响应。

```
package com.hitutor.controller;

import com.hitutor.dto.TutorProfileDTO;
import com.hitutor.entity.TutorProfile;
import com.hitutor.service.TutorProfileService;
import com.hitutor.utils.UserUtil;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

import java.util.HashMap;
import java.util.List;
import java.util.Map;

@RestController
@RequestMapping("/api/tutor-profiles")
public class TutorProfileController {

    @Autowired
    private TutorProfileService tutorProfileService;

    @Autowired
    private UserUtil userUtil;

    @GetMapping
    public ResponseEntity<?> getTutorProfiles(
            @RequestParam(defaultValue = "0") int page,
            @RequestParam(defaultValue = "10") int size,
            @RequestParam(required = false) Long subjectId,
            @RequestParam(required = false) String grade,
            @RequestParam(required = false) String sortBy
    ) {
        try {
            Map<String, Object> result = tutorProfileService.getTutorProfiles(page, siz
e, subjectId, grade, sortBy);

            Map<String, Object> response = new HashMap<>();
            response.put("success", true);
            response.put("message", "获取家教列表成功");
            response.put("data", result);

            return ResponseEntity.ok(response);
        } catch (Exception e) {
            Map<String, Object> response = new HashMap<>();
            response.put("success", false);
            response.put("message", e.getMessage());
```

```
            response.put("data", null);

            return ResponseEntity.status(HttpStatus.BAD_REQUEST).body(response);
        }
    }

    @GetMapping("/my")
    public ResponseEntity<?> getMyTutorProfiles() {
        try {
            String userId = userUtil.getCurrentUserId();
            List<TutorProfile> profiles = tutorProfileService.getMyTutorProfiles(userI
d);

            List<TutorProfileDTO> dtos = profiles.stream()
                    .map(TutorProfileDTO::fromTutorProfile)
                    .toList();

            Map<String, Object> response = new HashMap<>();
            response.put("success", true);
            response.put("message", "获取我的家教信息成功");
            response.put("data", dtos);

            return ResponseEntity.ok(response);
        } catch (Exception e) {
            Map<String, Object> response = new HashMap<>();
            response.put("success", false);
            response.put("message", e.getMessage());
            response.put("data", null);

            return ResponseEntity.status(HttpStatus.BAD_REQUEST).body(response);
        }
    }

    @PostMapping
    public ResponseEntity<?> createTutorProfile(@RequestBody TutorProfileDTO dto) {
        try {
            String userId = userUtil.getCurrentUserId();
            TutorProfile profile = tutorProfileService.createTutorProfile(userId, dto);

            Map<String, Object> response = new HashMap<>();
            response.put("success", true);
            response.put("message", "发布家教信息成功");
            response.put("data", TutorProfileDTO.fromTutorProfile(profile));

            return ResponseEntity.ok(response);
        } catch (Exception e) {
            Map<String, Object> response = new HashMap<>();
            response.put("success", false);
            response.put("message", e.getMessage());
            response.put("data", null);

            return ResponseEntity.status(HttpStatus.BAD_REQUEST).body(response);
        }
    }

    @GetMapping("/{id}")
    public ResponseEntity<?> getTutorProfile(@PathVariable Long id) {
        try {
            TutorProfile profile = tutorProfileService.getTutorProfile(id);
```

```
            Map<String, Object> response = new HashMap<>();
            response.put("success", true);
            response.put("message", "获取家教详情成功");
            response.put("data", TutorProfileDTO.fromTutorProfile(profile));

            return ResponseEntity.ok(response);
        } catch (Exception e) {
            Map<String, Object> response = new HashMap<>();
            response.put("success", false);
            response.put("message", e.getMessage());
            response.put("data", null);

            return ResponseEntity.status(HttpStatus.BAD_REQUEST).body(response);
        }
    }

    @PutMapping("/{id}")
    public ResponseEntity<?> updateTutorProfile(@PathVariable Long id, @RequestBody Tut
orProfileDTO dto) {
        try {
            String userId = userUtil.getCurrentUserId();
            TutorProfile profile = tutorProfileService.updateTutorProfile(id, userId, d
to);

            Map<String, Object> response = new HashMap<>();
            response.put("success", true);
            response.put("message", "更新家教信息成功");
            response.put("data", TutorProfileDTO.fromTutorProfile(profile));

            return ResponseEntity.ok(response);
        } catch (Exception e) {
            Map<String, Object> response = new HashMap<>();
            response.put("success", false);
            response.put("message", e.getMessage());
            response.put("data", null);

            return ResponseEntity.status(HttpStatus.BAD_REQUEST).body(response);
        }
    }

    @DeleteMapping("/{id}")
    public ResponseEntity<?> deleteTutorProfile(@PathVariable Long id) {
        try {
            String userId = userUtil.getCurrentUserId();
            tutorProfileService.deleteTutorProfile(id, userId);

            Map<String, Object> response = new HashMap<>();
            response.put("success", true);
            response.put("message", "删除家教信息成功");
            response.put("data", null);

            return ResponseEntity.ok(response);
        } catch (Exception e) {
            Map<String, Object> response = new HashMap<>();
            response.put("success", false);
            response.put("message", e.getMessage());
            response.put("data", null);

            return ResponseEntity.status(HttpStatus.BAD_REQUEST).body(response);
        }
```

```
    }
}
```

2.4 用户控制器

用户控制器负责处理用户相关的请求，文件位于 `hitutor-backend/src/main/java/com/hitutor/controller/UserController.java`。UserController 类使用了@RestController 注解，表示这是一个控制器类，所有方法的返回值都会自动转换为 JSON 格式。类中还使用了@RequestMapping 注解，指定基础路径为/api/users，所有方法的路径都会基于这个基础路径。

UserController 类中注入了两个依赖：UserService 用户服务，UserUtil 用户工具类。类中定义了五个方法：getCurrentUser 获取当前用户信息，updateCurrentUser 更新当前用户信息，getUserById 根据 ID 获取用户信息，changePassword 修改密码，changePhone 修改手机号。

getCurrentUser 方法使用了@GetMapping 注解，路径为/me，调用 UserUtil 的 getCurrentUserId 方法获取当前用户 ID，调用 UserService 的 getUserById 方法获取用户信息，构造响应数据，返回成功的响应。updateCurrentUser 方法使用了@PutMapping 注解，路径为/me，调用 UserUtil 的 getCurrentUserId 方法获取当前用户 ID，调用 UserService 的 updateUser 方法更新用户信息，构造响应数据，返回成功的响应。getUserById 方法使用了@GetMapping 注解，路径为/{id}，调用 UserService 的 getUserById 方法获取用户信息，构造响应数据，返回成功的响应。changePassword 方法使用了@PutMapping 注解，路径为/password，调用 UserUtil 的 getCurrentUserId 方法获取当前用户 ID，调用 UserService 的 changePassword 方法修改密码，构造响应数据，返回成功的响应。changePhone 方法使用了@PutMapping 注解，路径为/phone，调用 UserUtil 的 getCurrentUserId 方法获取当前用户 ID，调用 UserService 的 changePhone 方法修改手机号，构造响应数据，返回成功的响应。

```java
package com.hitutor.controller;

import com.hitutor.dto.UserDTO;
import com.hitutor.entity.User;
import com.hitutor.service.UserService;
import com.hitutor.utils.UserUtil;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

import java.util.HashMap;
import java.util.Map;

@RestController
@RequestMapping("/api/users")
public class UserController {

    @Autowired
    private UserService userService;

    @Autowired
    private UserUtil userUtil;

    @GetMapping("/me")
    public ResponseEntity<?> getCurrentUser() {
        try {
            String userId = userUtil.getCurrentUserId();
            User user = userService.getUserById(userId);

            Map<String, Object> response = new HashMap<>();
            response.put("success", true);
            response.put("message", "获取用户信息成功");
```

```
            response.put("data", UserDTO.fromUser(user));

            return ResponseEntity.ok(response);
        } catch (Exception e) {
            Map<String, Object> response = new HashMap<>();
            response.put("success", false);
            response.put("message", e.getMessage());
            response.put("data", null);

            return ResponseEntity.status(HttpStatus.BAD_REQUEST).body(response);
        }
    }

    @PutMapping("/me")
    public ResponseEntity<?> updateCurrentUser(@RequestBody UserDTO userDTO) {
        try {
            String userId = userUtil.getCurrentUserId();
            User user = userService.updateUser(userId, userDTO);

            Map<String, Object> response = new HashMap<>();
            response.put("success", true);
            response.put("message", "更新用户信息成功");
            response.put("data", UserDTO.fromUser(user));

            return ResponseEntity.ok(response);
        } catch (Exception e) {
            Map<String, Object> response = new HashMap<>();
            response.put("success", false);
            response.put("message", e.getMessage());
            response.put("data", null);

            return ResponseEntity.status(HttpStatus.BAD_REQUEST).body(response);
        }
    }

    @GetMapping("/{id}")
    public ResponseEntity<?> getUserById(@PathVariable String id) {
        try {
            User user = userService.getUserById(id);

            Map<String, Object> response = new HashMap<>();
            response.put("success", true);
            response.put("message", "获取用户信息成功");
            response.put("data", UserDTO.fromUser(user));

            return ResponseEntity.ok(response);
        } catch (Exception e) {
            Map<String, Object> response = new HashMap<>();
            response.put("success", false);
            response.put("message", e.getMessage());
            response.put("data", null);

            return ResponseEntity.status(HttpStatus.BAD_REQUEST).body(response);
        }
    }

    @PutMapping("/password")
    public ResponseEntity<?> changePassword(@RequestBody Map<String, String> request) {
        try {
            String userId = userUtil.getCurrentUserId();
```

```
            String oldPassword = request.get("oldPassword");
            String newPassword = request.get("newPassword");

            userService.changePassword(userId, oldPassword, newPassword);

            Map<String, Object> response = new HashMap<>();
            response.put("success", true);
            response.put("message", "密码修改成功");
            response.put("data", null);

            return ResponseEntity.ok(response);
        } catch (Exception e) {
            Map<String, Object> response = new HashMap<>();
            response.put("success", false);
            response.put("message", e.getMessage());
            response.put("data", null);

            return ResponseEntity.status(HttpStatus.BAD_REQUEST).body(response);
        }
    }

    @PutMapping("/phone")
    public ResponseEntity<?> changePhone(@RequestBody Map<String, String> request) {
        try {
            String userId = userUtil.getCurrentUserId();
            String newPhone = request.get("newPhone");
            String code = request.get("code");

            userService.changePhone(userId, newPhone, code);

            Map<String, Object> response = new HashMap<>();
            response.put("success", true);
            response.put("message", "手机号修改成功");
            response.put("data", null);

            return ResponseEntity.ok(response);
        } catch (Exception e) {
            Map<String, Object> response = new HashMap<>();
            response.put("success", false);
            response.put("message", e.getMessage());
            response.put("data", null);

            return ResponseEntity.status(HttpStatus.BAD_REQUEST).body(response);
        }
    }
}
```

2.5 JWT 工具类

JWT 工具类负责生成和验证 JWT 令牌，文件位于 `hitutor-backend/src/main/java/com/hitutor/utils/JwtUtil.java`。JwtUtil 类使用了@Component 注解，表示这是一个 Spring 组件，会被 Spring 容器管理。类中定义了三个属性：secret 密钥，expiration 过期时间，都使用了@Value 注解，从配置文件中读取。

JwtUtil 类中定义了六个方法：generateToken 生成 JWT 令牌，extractClaims 提取 JWT 令牌的声明，getUserIdFromToken 从令牌中获取用户 ID，getRoleFromToken 从令牌中获取角色，isTokenExpired 判断令牌是否过期，validateToken 验证令牌是否有效。

generateToken 方法接收用户 ID 和角色，创建一个 Map 对象，将用户 ID 和角色放入 Map 中，使用 Jwts.builder 方法构建 JWT 令牌，设置声明、主题、签发时间、过期时间，使用 HS256 算法和密钥签名，生成 JWT 令牌字符串并返回。extractClaims 方法接收 JWT 令牌字符串，使用 Jwts.parser 方法解析 JWT 令牌，使用密钥验证签名，提取声明并返回。getUserIdFromToken 方法接收 JWT 令牌字符串，调用 extractClaims 方法提取声明，从声明中获取用户 ID 并返回。getRoleFromToken 方法接收 JWT 令牌字符串，调用 extractClaims 方法提取声明，从声明中获取角色并返回。isTokenExpired 方法接收 JWT 令牌字符串，调用 extractClaims 方法提取声明，获取过期时间，判断过期时间是否在当前时间之前，返回判断结果。validateToken 方法接收 JWT 令牌字符串，使用 try-catch 块捕获异常，使用 Jwts.parser 方法解析 JWT 令牌，使用密钥验证签名，调用 isTokenExpired 方法判断令牌是否过期，返回验证结果。

```
package com.hitutor.utils;

import io.jsonwebtoken.Claims;
import io.jsonwebtoken.Jwts;
import io.jsonwebtoken.SignatureAlgorithm;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.stereotype.Component;

import java.util.Date;
import java.util.HashMap;
import java.util.Map;

@Component
public class JwtUtil {

    @Value("${app.jwt.secret}")
    private String secret;

    @Value("${app.jwt.expiration}")
    private long expiration;

    public String generateToken(String userId, String role) {
        Map<String, Object> claims = new HashMap<>();
        claims.put("userId", userId);
        claims.put("role", role);

        return Jwts.builder()
                .setClaims(claims)
                .setSubject(userId)
                .setIssuedAt(new Date())
                .setExpiration(new Date(System.currentTimeMillis() + expiration))
                .signWith(SignatureAlgorithm.HS256, secret)
                .compact();
    }

    public Claims extractClaims(String token) {
        return Jwts.parser()
                .setSigningKey(secret)
                .parseClaimsJws(token)
                .getBody();
    }

    public String getUserIdFromToken(String token) {
        return extractClaims(token).get("userId", String.class);
    }

    public String getRoleFromToken(String token) {
        return extractClaims(token).get("role", String.class);
    }
```

```
    public boolean isTokenExpired(String token) {
        return extractClaims(token).getExpiration().before(new Date());
    }

    public boolean validateToken(String token) {
        try {
            Jwts.parser().setSigningKey(secret).parseClaimsJws(token);
            return !isTokenExpired(token);
        } catch (Exception e) {
            return false;
        }
    }
}
```

2.6 用户工具类

用户工具类提供获取当前用户信息的方法，文件位于 hitutor-backend/src/main/java/com/hitutor/utils/UserUtil.java。UserUtil 类使用了@Component 注解，表示这是一个 Spring 组件，会被 Spring 容器管理。类中定义了四个方法：getCurrentUserId 获取当前用户 ID，getCurrentUserRole 获取当前用户角色，isTutor 判断当前用户是否是家教，isStudent 判断当前用户是否是学生，isAdmin 判断当前用户是否是管理员。

getCurrentUserId 方法从 SecurityContextHolder 中获取当前认证信息，如果认证信息为 null 或未认证，抛出运行时异常，返回认证信息的名称，即用户 ID。getCurrentUserRole 方法从 SecurityContextHolder 中获取当前认证信息，如果认证信息为 null 或未认证，抛出运行时异常，返回认证信息的第一个权限，即用户角色。isTutor 方法调用 getCurrentUserRole 方法获取当前用户角色，判断是否等于 tutor，返回判断结果。isStudent 方法调用 getCurrentUserRole 方法获取当前用户角色，判断是否等于 student，返回判断结果。isAdmin 方法调用 getCurrentUserRole 方法获取当前用户角色，判断是否等于 admin，返回判断结果。

```
package com.hitutor.utils;

import org.springframework.security.core.Authentication;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.stereotype.Component;

@Component
public class UserUtil {

    public String getCurrentUserId() {
        Authentication authentication = SecurityContextHolder.getContext().getAuthentic
ation();
        if (authentication == null || !authentication.isAuthenticated()) {
            throw new RuntimeException("用户未认证");
        }
        return authentication.getName();
    }

    public String getCurrentUserRole() {
        Authentication authentication = SecurityContextHolder.getContext().getAuthentic
ation();
        if (authentication == null || !authentication.isAuthenticated()) {
            throw new RuntimeException("用户未认证");
        }
        return authentication.getAuthorities().iterator().next().getAuthority();
    }

    public boolean isTutor() {
        return "tutor".equals(getCurrentUserRole());
```

```
    }

    public boolean isStudent() {
        return "student".equals(getCurrentUserRole());
    }

    public boolean isAdmin() {
        return "admin".equals(getCurrentUserRole());
    }
}
```

2.7 全局异常处理器

全局异常处理器负责处理应用中的异常，文件位于 hitutor-backend/src/main/java/com/hitutor/config/GlobalExceptionHandler.java。GlobalExceptionHandler 类使用了@ControllerAdvice 注解，表示这是一个全局异常处理器，会捕获所有控制器中抛出的异常。类中定义了两个方法：handleException 处理所有异常，handleRuntimeException 处理运行时异常。

handleException 方法使用了@ExceptionHandler 注解，指定处理的异常类型为 Exception，方法接收一个 Exception 对象，构造一个 Map 对象，设置 success 为 false，message 为异常消息，data 为 null，返回一个响应实体，状态码为 BAD_REQUEST。handleRuntimeException 方法使用了@ExceptionHandler 注解，指定处理的异常类型为 RuntimeException，方法接收一个 RuntimeException 对象，构造一个 Map 对象，设置 success 为 false，message 为异常消息，data 为 null，返回一个响应实体，状态码为 BAD_REQUEST。

```
package com.hitutor.config;

import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.ControllerAdvice;
import org.springframework.web.bind.annotation.ExceptionHandler;

import java.util.HashMap;
import java.util.Map;

@ControllerAdvice
public class GlobalExceptionHandler {

    @ExceptionHandler(Exception.class)
    public ResponseEntity<?> handleException(Exception e) {
        Map<String, Object> response = new HashMap<>();
        response.put("success", false);
        response.put("message", e.getMessage());
        response.put("data", null);

        return ResponseEntity.status(HttpStatus.BAD_REQUEST).body(response);
    }

    @ExceptionHandler(RuntimeException.class)
    public ResponseEntity<?> handleRuntimeException(RuntimeException e) {
        Map<String, Object> response = new HashMap<>();
        response.put("success", false);
        response.put("message", e.getMessage());
        response.put("data", null);

        return ResponseEntity.status(HttpStatus.BAD_REQUEST).body(response);
    }
}
```

2.8 安全配置

安全配置负责配置 Spring Security，文件位于 `hitutor-backend/src/main/java/com/hitutor/config/SecurityConfig.java`。SecurityConfig 类使用了@Configuration 注解，表示这是一个配置类，使用了@EnableWebSecurity 注解，表示启用 Web Security。类中注入了两个依赖：JwtAuthenticationFilter JWT 认证过滤器，CustomAccessDeniedHandler 自定义访问拒绝处理器。

SecurityConfig 类中定义了三个 Bean 方法：securityFilterChain 配置安全过滤器链，passwordEncoder 配置密码编码器，authenticationManager 配置认证管理器。securityFilterChain 方法接收 HttpSecurity 对象，配置安全规则，禁用 CSRF 保护，设置会话创建策略为无状态，配置请求授权规则，允许/api/auth/**和/api/verification/**路径的匿名访问，要求/api/admin/**路径需要 admin 权限，其他路径需要认证，配置异常处理，设置自定义访问拒绝处理器，添加 JWT 认证过滤器到过滤器链中，返回构建好的 SecurityFilterChain 对象。passwordEncoder 方法创建并返回一个 BCryptPasswordEncoder 对象，用于密码加密。authenticationManager 方法接收 AuthenticationConfiguration 对象，获取认证管理器并返回。

```
package com.hitutor.config;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.authentication.AuthenticationManager;
import org.springframework.security.config.annotation.authentication.configuration.Auth
enticationConfiguration;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.config.annotation.web.configuration.EnableWebSecuri
ty;
import org.springframework.security.config.http.SessionCreationPolicy;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.security.web.SecurityFilterChain;
import org.springframework.security.web.authentication.UsernamePasswordAuthenticationFi
lter;

@Configuration
@EnableWebSecurity
public class SecurityConfig {

    @Autowired
    private JwtAuthenticationFilter jwtAuthenticationFilter;

    @Autowired
    private CustomAccessDeniedHandler customAccessDeniedHandler;

    @Bean
    public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception
{
        http
                .csrf(csrf -> csrf.disable())
                .sessionManagement(session -> session.sessionCreationPolicy(SessionCrea
tionPolicy.STATELESS))
                .authorizeRequests(authorize -> authorize
                        .antMatchers("/api/auth/**").permitAll()
                        .antMatchers("/api/verification/**").permitAll()
                        .antMatchers("/api/subjects/**").permitAll()
                        .antMatchers("/api/admin/**").hasAuthority("admin")
                        .anyRequest().authenticated()
                )
                .exceptionHandling(exception -> exception
```

```
                        .accessDeniedHandler(customAccessDeniedHandler)
                )
                .addFilterBefore(jwtAuthenticationFilter, UsernamePasswordAuthenticatio
nFilter.class);

        return http.build();
    }

    @Bean
    public PasswordEncoder passwordEncoder() {
        return new BCryptPasswordEncoder();
    }

    @Bean
    public AuthenticationManager authenticationManager(AuthenticationConfiguration auth
enticationConfiguration) throws Exception {
        return authenticationConfiguration.getAuthenticationManager();
    }
}
```

2.9 JWT 认证过滤器

JWT 认证过滤器负责验证 JWT 令牌，文件位于 `hitutor-backend/src/main/java/com/hitutor/config/JwtAuthenticationFilter.java`。JwtAuthenticationFilter 类继承了 OncePerRequestFilter 类，表示每个请求只会执行一次。类中注入了一个依赖：JwtUtil JWT 工具类。

JwtAuthenticationFilter 类重写了 doFilterInternal 方法，接收 HttpServletRequest、HttpServletResponse、FilterChain 参数，从请求头中获取 Authorization 信息，判断 Authorization 信息是否存在且以 Bearer 开头，如果存在，提取 JWT 令牌，调用 JwtUtil 的 validateToken 方法验证令牌是否有效，如果有效，调用 JwtUtil 的 getUserIdFromToken 方法获取用户 ID，调用 JwtUtil 的 getRoleFromToken 方法获取角色，创建 UsernamePasswordAuthenticationToken 对象，设置认证信息到 SecurityContextHolder 中，最后调用 FilterChain 的 doFilter 方法继续过滤器链。

```
package com.hitutor.config;

import com.hitutor.utils.JwtUtil;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;
import org.springframework.security.core.authority.SimpleGrantedAuthority;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.stereotype.Component;
import org.springframework.web.filter.OncePerRequestFilter;

import javax.servlet.FilterChain;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;
import java.util.Collections;

@Component
public class JwtAuthenticationFilter extends OncePerRequestFilter {

    @Autowired
    private JwtUtil jwtUtil;

    @Override
    protected void doFilterInternal(HttpServletRequest request, HttpServletResponse res
```

```
ponse, FilterChain filterChain) throws ServletException, IOException {
        String authorizationHeader = request.getHeader("Authorization");

        if (authorizationHeader != null && authorizationHeader.startsWith("Bearer ")) {
            String token = authorizationHeader.substring(7);

            if (jwtUtil.validateToken(token)) {
                String userId = jwtUtil.getUserIdFromToken(token);
                String role = jwtUtil.getRoleFromToken(token);

                UsernamePasswordAuthenticationToken authentication = new UsernamePasswo
rdAuthenticationToken(
                        userId,
                        null,
                        Collections.singletonList(new SimpleGrantedAuthority(role))
                );

                SecurityContextHolder.getContext().setAuthentication(authentication);
            }
        }

        filterChain.doFilter(request, response);
    }
}
```

三、源代码统计

3.1 前端代码统计

前端项目包含约 150 个文件，代码行数约 25,000 行，主要使用 Dart 编程语言。前端项目的主要目录包括：lib 目录存放源代码，包括 models、pages、providers、services、theme、utils、widgets 等子目录；android 目录存放 Android 平台代码；ios 目录存放 iOS 平台代码；web 目录存放 Web 平台代码；windows 目录存放 Windows 平台代码；linux 目录存放 Linux 平台代码；macos 目录存放 macOS 平台代码。

前端项目的主要文件包括：main.dart 应用入口文件，routes.dart 路由配置文件，api_service.dart API 服务文件，auth_provider.dart 认证提供者文件，user_model.dart 用户模型文件，tutor_provider.dart 家教提供者文件，password_login_page.dart 密码登录页面文件等。

3.2 后端代码统计

后端项目包含约 120 个文件，代码行数约 17,000 行，主要使用 Java 编程语言。后端项目的主要目录包括：src/main/java/com/hitutor 目录存放 Java 源代码，包括 config、controller、document、dto、entity、mapper、service、utils 等子目录；src/main/resources 目录存放资源文件，包括 application.yml 和 application.properties 配置文件。

后端项目的主要文件包括：HiTutorApplication.java 应用入口文件，AuthController.java 认证控制器文件，TutorProfileController.java 家教信息控制器文件，UserController.java 用户控制器文件，JwtUtil.java JWT 工具类文件，UserUtil.java 用户工具类文件，GlobalExceptionHandler.java 全局异常处理器文件，SecurityConfig.java 安全配置文件，JwtAuthenticationFilter.java JWT 认证过滤器文件等。

3.3 总计

整个项目包含约 270 个文件，代码行数约 42,000 行，主要使用 Dart 和 Java 编程语言。项目采用前后端分离架构，前端使用 Flutter 框架，后端使用 Spring Boot 框架，数据库使用 MySQL，实现了家教信息对接平台的所有功能。

四、技术栈

4.1 前端技术栈

前端使用 Flutter 3.2.6 框架，编程语言为 Dart 3.2.6，使用 Provider 6.1.0 进行状态管理，HTTP 1.1.0 进行网络请求，Shared Preferences 2.2.3 进行本地数据存储，AMap Flutter Map 3.0.0 提供地图服务，Geolocator 11.1.0 提供地理位置服务，Permission Handler 11.0.0 处理权限请求。

4.2 后端技术栈

后端使用 Spring Boot 3.1.0 框架，编程语言为 Java 17，使用 MyBatis Plus 3.5.4.1 进行数据库操作，数据库采用 MySQL 8.0.33，使用 JWT 0.11.5 进行身份认证，使用 WebSocket 实现实时通信。

五、项目结构

5.1 前端项目结构

前端项目结构如下：client 目录是前端项目根目录，lib 目录存放源代码，models 目录存放数据模型，pages 目录存放页面组件，providers 目录存放状态管理，services 目录存放 API 接口，theme 目录存放主题配置，utils 目录存放工具类，widgets 目录存放自定义组件，main.dart 是应用入口文件，routes.dart 是路由配置文件。android 目录存放 Android 平台代码，ios 目录存放 iOS 平台代码，web 目录存放 Web 平台代码，windows 目录存放 Windows 平台代码，linux 目录存放 Linux 平台代码，macos 目录存放 macOS 平台代码，pubspec.yaml 是 Flutter 配置文件。

5.2 后端项目结构

后端项目结构如下：hitutor-backend 目录是后端项目根目录，src/main/java/com/hitutor 目录存放 Java 源代码，config 目录存放配置类，controller 目录存放控制器，document 目录存放文档，dto 目录存放数据传输对象，entity 目录存放实体类，mapper 目录存放数据访问层，service 目录存放服务层，utils 目录存放工具类，HiTutorApplication.java 是应用入口文件。src/main/resources 目录存放资源文件，application.yml 是应用配置文件，application.properties 是应用配置文件，pom.xml 是 Maven 配置文件。

六、更多前端源代码

6.1 首页文件

首页文件是应用的主要功能入口，文件位于 `client/lib/pages/main/home_page.dart`。HomePage 类是一个有状态组件，使用 StatefulWidget 实现。类中定义了私有变量：*searchController 搜索输入控制器*，searchFocusNode 搜索焦点节点，*searchTimer 搜索定时器*，isSearching 是否正在搜索，_unreadNotificationCount 未读通知数量。

HomePage 类的主要功能包括：显示搜索栏，支持搜索家教和学生需求；显示快捷操作按钮，包括附近学生、附近老师、我要发布、我的积分；显示家教信息或学生需求列表；支持加载更多数据；显示未读通知数量；支持跳转到地图页面、通知页面、发布页面等。

HomePage 类使用了 Consumer 组件监听 TutorProvider 的状态变化，根据 provider 的状态显示加载状态、错误状态或数据列表。HomePage 类还使用了 MediaQuery 获取屏幕宽度，根据屏幕宽度调整内容边距，支持平板和手机两种设备。

```dart
import 'dart:async';
import 'package:flutter/material.dart';
import 'package:provider/provider.dart';
import '../../theme/app_theme.dart';
import '../../providers/tutor_provider.dart';
import '../../providers/auth_provider.dart';
import '../../models/tutor_model.dart';
import '../../routes.dart';
import '../../services/api_service.dart';
```

```dart
import '../student_request/student_request_detail_page.dart';
import '../tutor_service/tutor_service_detail_page.dart';
import '../notification/notification_page.dart';

final BorderRadius _borderRadius12 = BorderRadius.circular(12);
final BorderRadius _borderRadius14 = BorderRadius.circular(14);
const EdgeInsets _padding16 = EdgeInsets.all(16);
const EdgeInsets _paddingH16V12 = EdgeInsets.symmetric(
  horizontal: 16,
  vertical: 12,
);
const EdgeInsets _paddingH14V4 = EdgeInsets.symmetric(
  horizontal: 14,
  vertical: 4,
);
const EdgeInsets _paddingH8V4 = EdgeInsets.symmetric(
  horizontal: 8,
  vertical: 4,
);

class HomePage extends StatefulWidget {
  const HomePage({super.key});

  @override
  State<HomePage> createState() => _HomePageState();
}

class _HomePageState extends State<HomePage> {
  final TextEditingController _searchController = TextEditingController();
  final FocusNode _searchFocusNode = FocusNode();
  Timer? _searchTimer;
  bool _isSearching = false;
  int _unreadNotificationCount = 0;

  @override
  void initState() {
    super.initState();
    _loadDataBasedOnRole();
    _loadUnreadNotificationCount();
  }

  Future<void> _loadUnreadNotificationCount() async {
    try {
      final response = await ApiService.getUnreadNotificationCount();
      if (response is Map && response['success'] == true) {
        setState(() {
          _unreadNotificationCount = response['data'] ?? 0;
        });
      }
    } catch (_) {
    }
  }

  void _loadDataBasedOnRole() {
    final authProvider = Provider.of<AuthProvider>(context, listen: false);
    final user = authProvider.user;
    if (user != null && user.userRole == 'tutor') {
      Provider.of<TutorProvider>(context, listen: false).getNearbyStudents();
    } else {
      Provider.of<TutorProvider>(context, listen: false).getNearbyTutors();
    }
```

```dart
}

@override
void dispose() {
  _searchController.dispose();
  _searchFocusNode.dispose();
  _searchTimer?.cancel();
  super.dispose();
}

void _handleSearch(String query) {
  _searchTimer?.cancel();

  _searchTimer = Timer(const Duration(milliseconds: 500), () {
    if (mounted) {
      final authProvider = Provider.of<AuthProvider>(context, listen: false);
      final user = authProvider.user;
      if (user != null && user.userRole == 'tutor') {
        Provider.of<TutorProvider>(context, listen: false).searchStudents(query);
      } else {
        Provider.of<TutorProvider>(context, listen: false).searchTutors(query);
      }
      setState(() {
        _isSearching = query.isNotEmpty;
      });
    }
  });
}

void _clearSearch() {
  _searchController.clear();
  Provider.of<TutorProvider>(context, listen: false).resetSearch();
  setState(() {
    _isSearching = false;
  });
}

@override
Widget build(BuildContext context) {
  final screenWidth = MediaQuery.of(context).size.width;
  final isTablet = screenWidth >= 768;
  final contentPadding = isTablet
      ? EdgeInsets.symmetric(horizontal: screenWidth * 0.15)
      : _padding16;
  final authProvider = Provider.of<AuthProvider>(context);
  final user = authProvider.user;

  final userRole = user?.userRole ?? 'student';

  return Scaffold(
    backgroundColor: AppTheme.backgroundColor,
    body: SafeArea(
      child: Column(
        children: [
          _buildSearchBar(userRole),
          Expanded(
            child: Consumer<TutorProvider>(
              builder: (context, provider, child) {
                if (provider.isLoading && provider.tutors.isEmpty && !_isSearching) {
                  return _buildLoadingState(userRole);
                }
```

```
                if (provider.errorMessage.isNotEmpty && !_isSearching) {
                  return _buildErrorState(provider.errorMessage);
                }

                return SingleChildScrollView(
                  child: Padding(
                    padding: contentPadding,
                    child: Column(
                      children: [
                        _buildQuickActions(userRole),
                        if (_isSearching)
                          _buildSearchResults(provider, userRole)
                        else
                          _buildListings(provider, userRole),
                      ],
                    ),
                  ),
                );
              },
            ),
          ),
        ],
      ),
    ),
  );
}

Widget _buildSearchBar(String role) {
  final hintText = role == 'tutor' ? '搜索学生需求、科目...' : '搜索家教、科目、地区...';

  return Container(
    padding: _paddingH16V12,
    color: Colors.white,
    child: Row(
      children: [
        Expanded(
          child: MouseRegion(
            cursor: SystemMouseCursors.text,
            child: Container(
              padding: _paddingH14V4,
              decoration: BoxDecoration(
                color: Colors.white,
                borderRadius: _borderRadius12,
                boxShadow: [
                  BoxShadow(
                    color: Colors.black.withOpacity(0.04),
                    blurRadius: 8,
                    offset: const Offset(0, 2),
                  ),
                ],
              ),
              child: Row(
                children: [
                  const Icon(
                    Icons.search_rounded,
                    size: 20,
                    color: AppTheme.textTertiary,
                  ),
                  const SizedBox(width: 8),
                  Expanded(
```

```
            child: TextField(
              controller: _searchController,
              focusNode: _searchFocusNode,
              onChanged: _handleSearch,
              decoration: InputDecoration(
                border: InputBorder.none,
                enabledBorder: InputBorder.none,
                focusedBorder: InputBorder.none,
                hintText: hintText,
                hintStyle: TextStyle(
                  fontSize: 18,
                  color: AppTheme.textTertiary.withOpacity(0.6),
                ),
                contentPadding: EdgeInsets.zero,
                isDense: true,
              ),
              style: const TextStyle(
                fontSize: 16,
                color: AppTheme.textPrimary,
              ),
              cursorColor: AppTheme.primaryColor,
              enableInteractiveSelection: false,
            ),
          ),
          if (_searchController.text.isNotEmpty)
            GestureDetector(
              onTap: _clearSearch,
              child: const Icon(
                Icons.clear_rounded,
                size: 18,
                color: AppTheme.textTertiary,
              ),
            ),
        ],
      ),
    ),
  ),
),
const SizedBox(width: 12),
GestureDetector(
  onTap: () {
    Navigator.push(
      context,
      MaterialPageRoute(
        builder: (context) => const NotificationPage(),
      ),
    ).then((_) {
      _loadUnreadNotificationCount();
    });
  },
  child: Container(
    width: 40,
    height: 40,
    decoration: BoxDecoration(
      color: AppTheme.primaryColor.withOpacity(0.1),
      borderRadius: _borderRadius12,
    ),
    child: Stack(
      children: [
        const Center(
          child: Icon(
```

```
                    Icons.notifications_outlined,
                    size: 20,
                    color: AppTheme.primaryColor,
                  ),
                ),
                if (_unreadNotificationCount > 0)
                  Positioned(
                    top: 4,
                    right: 4,
                    child: Container(
                      width: 8,
                      height: 8,
                      decoration: const BoxDecoration(
                        color: Colors.red,
                        shape: BoxShape.circle,
                      ),
                    ),
                  ),
              ],
            ),
          ),
        ),
        const SizedBox(width: 12),
        GestureDetector(
          onTap: () {
            final authProvider = Provider.of<AuthProvider>(context, listen: false);
            final user = authProvider.user;
            final role = user?.userRole ?? 'student';
            Navigator.pushNamed(context, Routes.map, arguments: role);
          },
          child: Container(
            width: 40,
            height: 40,
            decoration: BoxDecoration(
              color: AppTheme.primaryColor.withOpacity(0.1),
              borderRadius: _borderRadius12,
            ),
            child: const Icon(
              Icons.location_on_rounded,
              size: 20,
              color: AppTheme.primaryColor,
            ),
          ),
        ),
      ],
    ),
  );
}

Widget _buildQuickActions(String role) {
  return Container(
    color: Colors.white,
    padding: const EdgeInsets.all(16),
    child: Row(
      mainAxisAlignment: MainAxisAlignment.spaceAround,
      children: [
        _buildQuickActionItem(
          Icons.school_rounded,
          '附近学生',
          AppTheme.primaryColor,
          () => Navigator.pushNamed(context, Routes.map, arguments: 'tutor'),
```

```
          ),
          _buildQuickActionItem(
            Icons.person_search_rounded,
            '附近老师',
            const Color(0xFF722ED1),
            () => Navigator.pushNamed(context, Routes.map, arguments: 'student'),
          ),
          _buildQuickActionItem(
            Icons.edit_document,
            '我要发布',
            const Color(0xFF10B981),
            () {
              if (Provider.of<AuthProvider>(context, listen: false).isAuthenticated) {
                Navigator.pushNamed(
                  context,
                  role == 'tutor' ? Routes.publishTutorService : Routes.publishStudentR
equest,
                );
              } else {
                Navigator.pushNamed(context, Routes.smsLogin);
              }
            },
          ),
          _buildQuickActionItem(
            Icons.monetization_on_rounded,
            '我的积分',
            const Color(0xFFFF9500),
            () {
              if (Provider.of<AuthProvider>(context, listen: false).isAuthenticated) {
                Navigator.pushNamed(context, Routes.points);
              } else {
                Navigator.pushNamed(context, Routes.smsLogin);
              }
            },
          ),
        ],
      ),
    );
  }

  Widget _buildQuickActionItem(IconData icon, String label, Color color, VoidCallback o
nTap) {
    return GestureDetector(
      onTap: onTap,
      child: Column(
        children: [
          Container(
            width: 48,
            height: 48,
            decoration: BoxDecoration(
              color: color.withOpacity(0.1),
              borderRadius: _borderRadius12,
            ),
            child: Icon(icon, color: color, size: 24),
          ),
          const SizedBox(height: 8),
          Text(
            label,
            style: const TextStyle(
              fontSize: 12,
```

```
            color: AppTheme.textPrimary,
            fontWeight: FontWeight.w500,
          ),
        ),
      ],
    ),
  );
}

Widget _buildLoadingState(String role) {
  final message = role == 'tutor' ? '正在加载附近学生...' : '正在加载附近家教...';

  return Center(
    child: Column(
      mainAxisAlignment: MainAxisAlignment.center,
      children: [
        const CircularProgressIndicator(color: AppTheme.primaryColor),
        const SizedBox(height: 16),
        Text(
          message,
          style: const TextStyle(fontSize: 14, color: AppTheme.textSecondary),
        ),
      ],
    ),
  );
}

Widget _buildErrorState(String errorMessage) {
  return Center(
    child: Column(
      mainAxisAlignment: MainAxisAlignment.center,
      children: [
        const Icon(
          Icons.error_outline_rounded,
          size: 48,
          color: AppTheme.errorColor,
        ),
        const SizedBox(height: 16),
        const Text(
          '加载失败',
          style: TextStyle(
            fontSize: 16,
            fontWeight: FontWeight.w600,
            color: AppTheme.textPrimary,
          ),
        ),
        const SizedBox(height: 8),
        Text(
          errorMessage,
          style: const TextStyle(fontSize: 14, color: AppTheme.textSecondary),
          textAlign: TextAlign.center,
        ),
        const SizedBox(height: 16),
        ElevatedButton(
          onPressed: () {
            _loadDataBasedOnRole();
          },
          style: ElevatedButton.styleFrom(
            backgroundColor: AppTheme.primaryColor,
            padding: const EdgeInsets.symmetric(horizontal: 24, vertical: 10),
            shape: RoundedRectangleBorder(
```

```
                borderRadius: BorderRadius.circular(20),
              ),
            ),
            child: const Text('重试'),
          ),
        ],
      ),
    );
  }
}
```

6.2 发布家教信息页面

发布家教信息页面允许家教老师发布家教信息，文件位于 client/lib/pages/publish/publish_tutor_service_page.dart。PublishTutorServicePage 类是一个有状态组件，使用 StatefulWidget 实现。类中定义了私有变量：*formKey 表单键*，hourlyRateController 时薪输入控制器，*descriptionController 描述输入控制器*，availableTimeController 可授课时间输入控制器，*addressController 地址输入控制器*，selectedSubjectId 选中的科目 ID，*selectedSubjectName 选中的科目名称*，isLoading 加载状态，*selectedLatitude 选中的纬度*，selectedLongitude 选中的经度，*selectedGradeLevel 选中的年级*，subjects 科目列表。

PublishTutorServicePage 类的主要功能包括：加载科目列表；选择授课科目；选择目标年级；输入时薪价格；输入教学描述；输入可授课时间；选择授课地址；提交家教信息。页面使用表单验证确保数据的完整性和有效性。

PublishTutorServicePage 类使用了 Form 组件包裹所有输入字段，使用 TextFormField 组件显示输入字段，使用 DropdownButtonFormField 组件显示下拉选择框，使用 GestureDetector 组件处理点击事件。页面还使用了 ElevatedButton 组件显示提交按钮，根据加载状态显示加载指示器或提交按钮文本。

```
import 'package:flutter/material.dart';
import 'package:provider/provider.dart';
import '../../theme/app_theme.dart';
import '../../services/api_service.dart';
import '../../providers/auth_provider.dart';
import '../../widgets/location_picker_page.dart';
import '../../models/tutor_model.dart';

class PublishTutorServicePage extends StatefulWidget {
  const PublishTutorServicePage({super.key});

  @override
  State<PublishTutorServicePage> createState() => _PublishTutorServicePageState();
}

class _PublishTutorServicePageState extends State<PublishTutorServicePage> {
  final _formKey = GlobalKey<FormState>();
  final _hourlyRateController = TextEditingController(text: '100');
  final _descriptionController = TextEditingController();
  final _availableTimeController = TextEditingController();
  final _addressController = TextEditingController();

  String? _selectedSubjectId;
  String _selectedSubjectName = '';
  bool _isLoading = false;
  double? _selectedLatitude;
  double? _selectedLongitude;
  GradeLevel? _selectedGradeLevel;

  List<Map<String, dynamic>> _subjects = [];
```

```dart
@override
void initState() {
  super.initState();
  _loadSubjects();
}

@override
void dispose() {
  _hourlyRateController.dispose();
  _descriptionController.dispose();
  _availableTimeController.dispose();
  _addressController.dispose();
  super.dispose();
}

Future<void> _loadSubjects() async {
  try {
    final response = await ApiService.getActiveSubjects();

    if (response['success'] == true && response['data'] != null) {
        final data = response['data'];
        if (data is List) {
          setState(() {
            _subjects = List<Map<String, dynamic>>.from(data.cast());
          });
        }
      }
  } catch (e) {
    if (mounted) {
      ScaffoldMessenger.of(context).showSnackBar(
        SnackBar(content: Text('加载科目失败: ${e.toString()}')),
      );
    }
  }
}

Future<void> _submitService() async {
  if (!_formKey.currentState!.validate()) {
    return;
  }

  if (_selectedSubjectId == null) {
    ScaffoldMessenger.of(context).showSnackBar(
      const SnackBar(content: Text('请选择科目')),
    );
    return;
  }

  if (_selectedGradeLevel == null) {
    ScaffoldMessenger.of(context).showSnackBar(
      const SnackBar(content: Text('请选择目标年级')),
    );
    return;
  }

  setState(() {
    _isLoading = true;
  });
```

```
    try {
      final authProvider = Provider.of<AuthProvider>(context, listen: false);
      final userId = authProvider.user?.id;

      if (userId == null) {
        throw Exception('用户未登录');
      }

      final serviceData = {
        'userId': userId,
        'subjectId': int.parse(_selectedSubjectId!),
        'subjectName': _selectedSubjectName,
        'hourlyRate': _hourlyRateController.text.trim(),
        'address': _addressController.text.trim(),
        'latitude': (_selectedLatitude ?? 39.9042).toString(),
        'longitude': (_selectedLongitude ?? 116.4074).toString(),
        'description': _descriptionController.text.trim(),
        'availableTime': _availableTimeController.text.trim(),
        'targetGradeLevels': _selectedGradeLevel!.id.toString(),
        'status': 'available',
      };

      final response = await ApiService.createService(serviceData);

      if (!mounted) return;

      if (response['success'] == true || response['id'] != null) {
        ScaffoldMessenger.of(context).showSnackBar(
          const SnackBar(content: Text('发布成功')),
        );
        Navigator.pop(context, true);
      } else {
        throw Exception(response['message'] ?? '发布失败');
      }
    } catch (e) {
      if (!mounted) return;
      ScaffoldMessenger.of(context).showSnackBar(
        SnackBar(content: Text('发布失败: ${e.toString()}')),
      );
    } finally {
      if (mounted) {
        setState(() {
          _isLoading = false;
        });
      }
    }
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      backgroundColor: AppTheme.backgroundColor,
      appBar: AppBar(
        backgroundColor: Colors.white,
        elevation: 0,
        leading: IconButton(
          icon: const Icon(Icons.arrow_back_ios_new, size: 20),
          onPressed: () => Navigator.pop(context),
          color: AppTheme.textPrimary,
        ),
```

```
      title: const Text(
        '发布家教信息',
        style: TextStyle(
          fontSize: 18,
          fontWeight: FontWeight.w600,
          color: AppTheme.textPrimary,
        ),
      ),
      centerTitle: true,
    ),
    body: SafeArea(
      child: SingleChildScrollView(
        child: Padding(
          padding: const EdgeInsets.all(16),
          child: Form(
            key: _formKey,
            child: Column(
              crossAxisAlignment: CrossAxisAlignment.start,
              children: [
                _buildSectionTitle('基本信息'),
                const SizedBox(height: 12),
                _buildSubjectSelector(),
                const SizedBox(height: 16),
                _buildGradeLevelSelector(),
                const SizedBox(height: 16),
                _buildHourlyRateField(),
                const SizedBox(height: 24),
                _buildSectionTitle('教学信息'),
                const SizedBox(height: 12),
                _buildDescriptionField(),
                const SizedBox(height: 16),
                _buildAvailableTimeField(),
                const SizedBox(height: 16),
                _buildAddressField(),
                const SizedBox(height: 32),
                _buildSubmitButton(),
              ],
            ),
          ),
        ),
      ),
    ),
  );
}

Widget _buildSectionTitle(String title) {
  return Text(
    title,
    style: const TextStyle(
      fontSize: 16,
      fontWeight: FontWeight.w600,
      color: AppTheme.textPrimary,
    ),
  );
}

Widget _buildSubjectSelector() {
  return DropdownButtonFormField<String>(
    decoration: InputDecoration(
      labelText: '授课科目',
```

```
      border: OutlineInputBorder(
        borderRadius: BorderRadius.circular(8),
      ),
      contentPadding: const EdgeInsets.symmetric(
        horizontal: 16,
        vertical: 12,
      ),
    ),
    value: _selectedSubjectId,
    items: _subjects.map((subject) {
      return DropdownMenuItem<String>(
        value: subject['id'].toString(),
        child: Text(subject['name']),
      );
    }).toList(),
    onChanged: (value) {
      setState(() {
        _selectedSubjectId = value;
        _selectedSubjectName = _subjects.firstWhere(
          (s) => s['id'].toString() == value,
          orElse: () => {'name': ''},
        )['name'];
      });
    },
    validator: (value) {
      if (value == null || value.isEmpty) {
        return '请选择授课科目';
      }
      return null;
    },
  );
}

Widget _buildGradeLevelSelector() {
  return DropdownButtonFormField<GradeLevel>(
    decoration: InputDecoration(
      labelText: '目标年级',
      border: OutlineInputBorder(
        borderRadius: BorderRadius.circular(8),
      ),
      contentPadding: const EdgeInsets.symmetric(
        horizontal: 16,
        vertical: 12,
      ),
    ),
    value: _selectedGradeLevel,
    items: GradeLevel.values.map((level) {
      return DropdownMenuItem<GradeLevel>(
        value: level,
        child: Text(level.displayName),
      );
    }).toList(),
    onChanged: (value) {
      setState(() {
        _selectedGradeLevel = value;
      });
    },
    validator: (value) {
      if (value == null) {
        return '请选择目标年级';
```

```
      }
      return null;
    },
  );
}

Widget _buildHourlyRateField() {
  return TextFormField(
    controller: _hourlyRateController,
    keyboardType: TextInputType.number,
    decoration: InputDecoration(
      labelText: '时薪价格（元/小时）',
      border: OutlineInputBorder(
        borderRadius: BorderRadius.circular(8),
      ),
      contentPadding: const EdgeInsets.symmetric(
        horizontal: 16,
        vertical: 12,
      ),
      suffixText: '元/小时',
    ),
    validator: (value) {
      if (value == null || value.isEmpty) {
        return '请输入时薪价格';
      }
      final rate = double.tryParse(value);
      if (rate == null || rate <= 0) {
        return '请输入有效的时薪价格';
      }
      return null;
    },
  );
}

Widget _buildDescriptionField() {
  return TextFormField(
    controller: _descriptionController,
    maxLines: 5,
    decoration: InputDecoration(
      labelText: '教学描述',
      border: OutlineInputBorder(
        borderRadius: BorderRadius.circular(8),
      ),
      contentPadding: const EdgeInsets.symmetric(
        horizontal: 16,
        vertical: 12,
      ),
    ),
    validator: (value) {
      if (value == null || value.isEmpty) {
        return '请输入教学描述';
      }
      if (value.length < 10) {
        return '教学描述至少 10 个字符';
      }
      return null;
    },
  );
}
```

```
Widget _buildAvailableTimeField() {
  return TextFormField(
    controller: _availableTimeController,
    decoration: InputDecoration(
      labelText: '可授课时间',
      border: OutlineInputBorder(
        borderRadius: BorderRadius.circular(8),
      ),
      contentPadding: const EdgeInsets.symmetric(
        horizontal: 16,
        vertical: 12,
      ),
      hintText: '例如：周末全天、工作日晚上',
    ),
    validator: (value) {
      if (value == null || value.isEmpty) {
        return '请输入可授课时间';
      }
      return null;
    },
  );
}

Widget _buildAddressField() {
  return GestureDetector(
    onTap: () async {
      final result = await Navigator.push(
        context,
        MaterialPageRoute(
          builder: (context) => const LocationPickerPage(),
        ),
      );
      if (result != null && result is Map) {
        setState(() {
          _addressController.text = result['address'] ?? '';
          _selectedLatitude = result['latitude'];
          _selectedLongitude = result['longitude'];
        });
      }
    },
    child: AbsorbPointer(
      child: TextFormField(
        controller: _addressController,
        decoration: InputDecoration(
          labelText: '授课地址',
          border: OutlineInputBorder(
            borderRadius: BorderRadius.circular(8),
          ),
          contentPadding: const EdgeInsets.symmetric(
            horizontal: 16,
            vertical: 12,
          ),
          suffixIcon: const Icon(Icons.location_on),
        ),
        validator: (value) {
          if (value == null || value.isEmpty) {
            return '请输入授课地址';
          }
          return null;
        },
```

```
        ),
      ),
    );
  }

  Widget _buildSubmitButton() {
    return SizedBox(
      width: double.infinity,
      height: 48,
      child: ElevatedButton(
        onPressed: _isLoading ? null : _submitService,
        style: ElevatedButton.styleFrom(
          backgroundColor: AppTheme.primaryColor,
          shape: RoundedRectangleBorder(
            borderRadius: BorderRadius.circular(8),
          ),
        ),
        child: _isLoading
            ? const SizedBox(
                width: 20,
                height: 20,
                child: CircularProgressIndicator(
                  color: Colors.white,
                  strokeWidth: 2,
                ),
              )
            : const Text(
                '发布',
                style: TextStyle(
                  fontSize: 16,
                  fontWeight: FontWeight.w600,
                ),
              ),
      ),
    );
  }
}

enum GradeLevel {
  primary1(id: 1, displayName: '小学一年级'),
  primary2(id: 2, displayName: '小学二年级'),
  primary3(id: 3, displayName: '小学三年级'),
  primary4(id: 4, displayName: '小学四年级'),
  primary5(id: 5, displayName: '小学五年级'),
  primary6(id: 6, displayName: '小学六年级'),
  junior1(id: 7, displayName: '初中一年级'),
  junior2(id: 8, displayName: '初中二年级'),
  junior3(id: 9, displayName: '初中三年级'),
  senior1(id: 10, displayName: '高中一年级'),
  senior2(id: 11, displayName: '高中二年级'),
  senior3(id: 12, displayName: '高中三年级');

  final int id;
  final String displayName;

  const GradeLevel({
    required this.id,
    required this.displayName,
```

```
  });
}
```

七、更多后端源代码

7.1 用户实体类

用户实体类是后端的核心数据模型之一，文件位于 `hitutor-backend/src/main/java/com/hitutor/entity/User.java`。User 类使用 MyBatis Plus 的注解进行数据库映射，使用@TableName 注解指定数据库表名为 sys_user，使用@TableId 注解指定主键字段为 id，使用@TableField 注解指定其他字段与数据库列的映射关系。

User 类定义了用户的所有属性：id 用户唯一标识，username 用户名，password 密码，email 邮箱，phone 手机号，avatar 头像，role 角色，status 状态，isVerified 是否认证，gender 性别，birthDate 出生日期，education 学历，school 学校，major 专业，teachingExperience 教学经验，lastLoginIp 最后登录 IP，lastLoginTime 最后登录时间，points 积分，createTime 创建时间，updateTime 更新时间。

User 类使用了 Jackson 的注解进行 JSON 序列化控制，使用@JsonIgnore 注解排除密码字段，使用@JsonFormat 注解格式化日期时间字段，使用@JsonInclude 注置只包含非空字段。User 类还使用了 Jakarta Validation 的注解进行数据验证，使用@Email 注解验证邮箱格式，使用@Size 注解验证用户名长度。

```
package com.hitutor.entity;

import com.baomidou.mybatisplus.annotation.*;
import com.fasterxml.jackson.annotation.JsonIgnore;
import com.fasterxml.jackson.annotation.JsonFormat;
import com.fasterxml.jackson.annotation.JsonInclude;
import jakarta.validation.constraints.Email;
import jakarta.validation.constraints.Size;

import java.time.LocalDate;
import java.time.LocalDateTime;

@JsonInclude(JsonInclude.Include.NON_NULL)
@TableName("sys_user")
public class User {

    @TableId(type = IdType.INPUT)
    private String id;

    @TableField("username")
    @Size(min = 3, max = 50, message = "用户名长度必须在 3-50 个字符之间")
    private String username;

    @TableField("password")
    @JsonIgnore
    private String password;

    @TableField("email")
    @Email(message = "邮箱格式不正确")
    private String email;

    @TableField("phone")
    private String phone;

    @TableField("avatar")
    private String avatar;

    @TableField("role")
```

```java
    private String role;

    @TableField("status")
    private String status;

    @TableField("is_verified")
    private Integer isVerified;

    @TableField("gender")
    private String gender;

    @TableField("birth_date")
    private LocalDate birthDate;

    @TableField("education")
    private String education;

    @TableField("school")
    private String school;

    @TableField("major")
    private String major;

    @TableField("teaching_experience")
    private Integer teachingExperience;

    @TableField("last_login_ip")
    private String lastLoginIp;

    @TableField("last_login_time")
    @JsonFormat(pattern = "yyyy-MM-dd HH:mm:ss")
    private LocalDateTime lastLoginTime;

    @TableField("points")
    private Integer points;

    @TableField("create_time")
    @JsonFormat(pattern = "yyyy-MM-dd HH:mm:ss")
    private LocalDateTime createTime;

    @TableField("update_time")
    @JsonFormat(pattern = "yyyy-MM-dd HH:mm:ss")
    private LocalDateTime updateTime;


    public String getId() {
        return id;
    }

    public void setId(String id) {
        this.id = id;
    }

    public String getUsername() {
        return username;
    }

    public void setUsername(String username) {
        this.username = username;
    }
```

```java
public String getPassword() {
    return password;
}

public void setPassword(String password) {
    this.password = password;
}

public String getEmail() {
    return email;
}

public void setEmail(String email) {
    this.email = email;
}

public String getPhone() {
    return phone;
}

public void setPhone(String phone) {
    this.phone = phone;
}

public String getAvatar() {
    return avatar;
}

public void setAvatar(String avatar) {
    this.avatar = avatar;
}

public String getRole() {
    return role;
}

public void setRole(String role) {
    this.role = role;
}

public String getStatus() {
    return status;
}

public void setStatus(String status) {
    this.status = status;
}

public Integer getIsVerified() {
    return isVerified;
}

public void setIsVerified(Integer isVerified) {
    this.isVerified = isVerified;
}

public String getGender() {
    return gender;
}
```

```java
public void setGender(String gender) {
    this.gender = gender;
}

public LocalDate getBirthDate() {
    return birthDate;
}

public void setBirthDate(LocalDate birthDate) {
    this.birthDate = birthDate;
}

public String getEducation() {
    return education;
}

public void setEducation(String education) {
    this.education = education;
}

public String getSchool() {
    return school;
}

public void setSchool(String school) {
    this.school = school;
}

public String getMajor() {
    return major;
}

public void setMajor(String major) {
    this.major = major;
}

public Integer getTeachingExperience() {
    return teachingExperience;
}

public void setTeachingExperience(Integer teachingExperience) {
    this.teachingExperience = teachingExperience;
}

public String getLastLoginIp() {
    return lastLoginIp;
}

public void setLastLoginIp(String lastLoginIp) {
    this.lastLoginIp = lastLoginIp;
}

public LocalDateTime getLastLoginTime() {
    return lastLoginTime;
}

public void setLastLoginTime(LocalDateTime lastLoginTime) {
    this.lastLoginTime = lastLoginTime;
}
```

```
    public Integer getPoints() {
        return points;
    }

    public void setPoints(Integer points) {
        this.points = points;
    }

    public LocalDateTime getCreateTime() {
        return createTime;
    }

    public void setCreateTime(LocalDateTime createTime) {
        this.createTime = createTime;
    }

    public LocalDateTime getUpdateTime() {
        return updateTime;
    }

    public void setUpdateTime(LocalDateTime updateTime) {
        this.updateTime = updateTime;
    }
}
```

7.2 用户服务接口

用户服务接口定义了用户相关的业务逻辑，文件位于 `hitutor-backend/src/main/java/com/hitutor/service/UserService.java`。UserService 接口继承自 MyBatis Plus 的 IService 接口，获得了基础的 CRUD 操作方法，包括 save、saveOrUpdate、remove、update、getById、list 等方法。

UserService 接口定义了用户相关的业务方法：getUserById 根据 ID 获取用户，getUserByEmail 根据邮箱获取用户，getUserByPhone 根据手机号获取用户，getUserByUsername 根据用户名获取用户，saveUser 保存用户，updateUser 更新用户，deleteUser 删除用户，getTutors 获取家教列表，getStudents 获取学生列表，getAllUsers 获取所有用户，getActiveUsersCount 获取活跃用户数量，getTutorsCount 获取家教数量，getStudentsCount 获取学生数量，searchTutors 搜索家教，searchStudents 搜索学生，getUsersByIds 根据 ID 列表获取用户，getAllUserIds 获取所有用户 ID。

UserService 接口的方法设计考虑了各种业务场景，包括用户的增删改查、角色筛选、搜索功能、统计功能等。接口方法使用了泛型和集合类型，支持批量操作和复杂查询。

```
package com.hitutor.service;

import com.baomidou.mybatisplus.extension.service.IService;
import com.hitutor.entity.User;

import java.util.List;

public interface UserService extends IService<User> {
    User getUserById(String id);
    User getUserByEmail(String email);
    User getUserByPhone(String phone);
    User getUserByUsername(String username);
    boolean saveUser(User user);
    boolean updateUser(User user);
    boolean deleteUser(String id);
    List<User> getTutors();
```

```
    List<User> getStudents();
    List<User> getAllUsers();
    int getActiveUsersCount();
    int getTutorsCount();
    int getStudentsCount();
    List<User> searchTutors(String query, int page, int size);
    List<User> searchStudents(String query, int page, int size);
    List<User> getUsersByIds(List<String> ids);
    List<String> getAllUserIds();
}
```

7.3 家教信息实体类

家教信息实体类是后端的核心数据模型之一，文件位于 hitutor-backend/src/main/java/com/hitutor/entity/TutorProfile.java。TutorProfile 类使用 MyBatis Plus 的注解进行数据库映射，使用@TableName 注解指定数据库表名为 tutor_profile，使用@TableId 注解指定主键字段为 id，使用@TableField 注解指定其他字段与数据库列的映射关系。

TutorProfile 类定义了家教信息的所有属性：id 家教信息唯一标识，userId 用户 ID，subjectId 科目 ID，subjectName 科目名称，hourlyRate 时薪价格，address 授课地址，latitude 纬度，longitude 经度，description 教学描述，availableTime 可授课时间，targetGradeLevels 目标年级，status 状态，rating 评分，reviewCount 评价数量，createTime 创建时间，updateTime 更新时间。

TutorProfile 类使用了 Jackson 的注解进行 JSON 序列化控制，使用@JsonFormat 注解格式化日期时间字段，使用@JsonInclude 注置只包含非空字段。TutorProfile 类还使用了 Jakarta Validation 的注解进行数据验证，确保数据的完整性和有效性。

```
package com.hitutor.entity;

import com.baomidou.mybatisplus.annotation.*;
import com.fasterxml.jackson.annotation.JsonFormat;
import com.fasterxml.jackson.annotation.JsonInclude;

import java.time.LocalDateTime;

@JsonInclude(JsonInclude.Include.NON_NULL)
@TableName("tutor_profile")
public class TutorProfile {

    @TableId(type = IdType.AUTO)
    private Long id;

    @TableField("user_id")
    private String userId;

    @TableField("subject_id")
    private Integer subjectId;

    @TableField("subject_name")
    private String subjectName;

    @TableField("hourly_rate")
    private Double hourlyRate;

    @TableField("address")
    private String address;

    @TableField("latitude")
    private Double latitude;
```

```java
    @TableField("longitude")
    private Double longitude;

    @TableField("description")
    private String description;

    @TableField("available_time")
    private String availableTime;

    @TableField("target_grade_levels")
    private String targetGradeLevels;

    @TableField("status")
    private String status;

    @TableField("rating")
    private Double rating;

    @TableField("review_count")
    private Integer reviewCount;

    @TableField("create_time")
    @JsonFormat(pattern = "yyyy-MM-dd HH:mm:ss")
    private LocalDateTime createTime;

    @TableField("update_time")
    @JsonFormat(pattern = "yyyy-MM-dd HH:mm:ss")
    private LocalDateTime updateTime;


    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }

    public String getUserId() {
        return userId;
    }

    public void setUserId(String userId) {
        this.userId = userId;
    }

    public Integer getSubjectId() {
        return subjectId;
    }

    public void setSubjectId(Integer subjectId) {
        this.subjectId = subjectId;
    }

    public String getSubjectName() {
        return subjectName;
    }

    public void setSubjectName(String subjectName) {
```

```
        this.subjectName = subjectName;
    }

    public Double getHourlyRate() {
        return hourlyRate;
    }

    public void setHourlyRate(Double hourlyRate) {
        this.hourlyRate = hourlyRate;
    }

    public String getAddress() {
        return address;
    }

    public void setAddress(String address) {
        this.address = address;
    }

    public Double getLatitude() {
        return latitude;
    }

    public void setLatitude(Double latitude) {
        this.latitude = latitude;
    }

    public Double getLongitude() {
        return longitude;
    }

    public void setLongitude(Double longitude) {
        this.longitude = longitude;
    }

    public String getDescription() {
        return description;
    }

    public void setDescription(String description) {
        this.description = description;
    }

    public String getAvailableTime() {
        return availableTime;
    }

    public void setAvailableTime(String availableTime) {
        this.availableTime = availableTime;
    }

    public String getTargetGradeLevels() {
        return targetGradeLevels;
    }

    public void setTargetGradeLevels(String targetGradeLevels) {
        this.targetGradeLevels = targetGradeLevels;
    }

    public String getStatus() {
```

```
        return status;
    }

    public void setStatus(String status) {
        this.status = status;
    }

    public Double getRating() {
        return rating;
    }

    public void setRating(Double rating) {
        this.rating = rating;
    }

    public Integer getReviewCount() {
        return reviewCount;
    }

    public void setReviewCount(Integer reviewCount) {
        this.reviewCount = reviewCount;
    }

    public LocalDateTime getCreateTime() {
        return createTime;
    }

    public void setCreateTime(LocalDateTime createTime) {
        this.createTime = createTime;
    }

    public LocalDateTime getUpdateTime() {
        return updateTime;
    }

    public void setUpdateTime(LocalDateTime updateTime) {
        this.updateTime = updateTime;
    }
}
```

7.4 家教信息服务接口

家教信息服务接口定义了家教信息相关的业务逻辑，文件位于 hitutor-backend/src/main/java/com/hitutor/service/TutorProfileService.java。TutorProfileService 接口定义了家教信息相关的业务方法：getNearbyTutors 获取附近的家教，createTutorProfile 创建家教信息，getAllTutorProfiles 获取所有家教信息，getById 根据 ID 获取家教信息，updateTutorProfile 更新家教信息，deleteTutorProfile 删除家教信息，getProfilesByUserId 根据用户 ID 获取家教信息。

TutorProfileService 接口的方法设计考虑了各种业务场景，包括家教信息的增删改查、地理位置搜索、分页查询等。接口方法使用了泛型和集合类型，支持批量操作和复杂查询。getNearbyTutors 方法使用经纬度和半径参数进行地理位置搜索，实现了基于距离的家教信息筛选功能。

```
package com.hitutor.service;

import com.hitutor.entity.TutorProfile;
import java.util.List;
import java.util.Map;

public interface TutorProfileService {
```

```
    List<TutorProfile> getNearbyTutors(double latitude, double longitude, double radius,
 String subject);

    TutorProfile createTutorProfile(Map<String, Object> data);

    Map<String, Object> getAllTutorProfiles(int page, int size);

    TutorProfile getById(Long id);

    boolean updateTutorProfile(TutorProfile profile);

    boolean deleteTutorProfile(Long id);

    List<TutorProfile> getProfilesByUserId(String userId);
}
```

九、更多后端源代码

9.1 消息控制器

消息控制器负责处理消息相关的 HTTP 请求，文件位于 hitutor-backend/src/main/java/com/hitutor/controller/MessageController.java。MessageController 类使用@RestController 注解标记为 REST 控制器，使用@RequestMapping 注解指定基础路径为/api/messages。类中定义了三个自动注入的依赖：messageService 消息服务，chatMessageRepository 聊天消息仓库，notificationService 通知服务。

MessageController 类的主要功能包括：获取用户的会话列表；获取会话的消息列表；创建新的会话；发送消息；标记消息为已读。控制器使用了 Spring Security 的 Authentication 对象进行用户认证，使用 ResponseEntity 返回统一的响应格式。

MessageController 类使用了@GetMapping 注解处理 GET 请求，使用@PostMapping 注解处理 POST 请求，使用@PutMapping 注解处理 PUT 请求，使用@PathVariable 注解获取路径参数，使用@RequestParam 注解获取查询参数，使用@RequestBody 注解获取请求体。控制器还使用了分页查询，支持按页获取消息列表。

```
package com.hitutor.controller;

import com.hitutor.document.ChatMessage;
import com.hitutor.entity.Conversation;
import com.hitutor.entity.Notification;
import com.hitutor.repository.ChatMessageRepository;
import com.hitutor.service.MessageService;
import com.hitutor.service.NotificationService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import org.springframework.security.core.Authentication;
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.UUID;
import java.util.stream.Collectors;

@RestController
@RequestMapping("/api/messages")
public class MessageController {

    @Autowired
```

```
    private MessageService messageService;

    @Autowired
    private ChatMessageRepository chatMessageRepository;

    @Autowired
    private NotificationService notificationService;

    @GetMapping("/conversations")
    public ResponseEntity<Map<String, Object>> getConversations(
            @RequestParam String userId,
            Authentication authentication) {

        List<Map<String, Object>> conversations = messageService.getConversationsByUser
Id(userId);

        Map<String, Object> result = new HashMap<>();
        result.put("success", true);
        result.put("data", conversations);

        return ResponseEntity.ok(result);
    }

    @GetMapping("/conversations/{sessionId}/messages")
    public ResponseEntity<Map<String, Object>> getMessagesBySessionId(
            @PathVariable String sessionId,
            @RequestParam(defaultValue = "0") int page,
            @RequestParam(defaultValue = "20") int size) {

        List<ChatMessage> allMessages = chatMessageRepository.findByConversationIdOrder
ByCreateTimeAsc(sessionId);

        int fromIndex = page * size;
        int toIndex = Math.min(fromIndex + size, allMessages.size());

        List<ChatMessage> messages = allMessages.subList(fromIndex, toIndex);

        List<Map<String, Object>> messageList = messages.stream()
                .map(msg -> {
                    Map<String, Object> messageMap = new HashMap<>();
                    messageMap.put("id", msg.getId());
                    messageMap.put("conversationId", msg.getConversationId());
                    messageMap.put("senderId", msg.getSenderId());
                    messageMap.put("receiverId", msg.getReceiverId());
                    messageMap.put("content", msg.getContent());
                    messageMap.put("messageType", msg.getMessageType());
                    messageMap.put("isRead", msg.getIsRead());
                    messageMap.put("createTime", msg.getCreateTime());
                    return messageMap;
                })
                .collect(Collectors.toList());

        int totalPages = (int) Math.ceil((double) allMessages.size() / size);

        Map<String, Object> result = new HashMap<>();
        result.put("success", true);

        Map<String, Object> dataMap = new HashMap<>();
        dataMap.put("content", messageList);
        dataMap.put("totalElements", allMessages.size());
```

```
        dataMap.put("totalPages", totalPages);
        dataMap.put("currentPage", page);
        dataMap.put("size", size);

        result.put("data", dataMap);

        return ResponseEntity.ok(result);
    }

    @PostMapping("/conversations")
    public ResponseEntity<Map<String, Object>> createConversation(
            @RequestBody Map<String, String> request,
            Authentication authentication) {

        String user1Id = request.get("user1Id");
        String user2Id = request.get("user2Id");

        Conversation conversation = messageService.createConversation(user1Id, user2Id);

        Map<String, Object> result = new HashMap<>();
        result.put("success", true);
        result.put("data", conversation);
        result.put("message", "会话创建成功");

        return ResponseEntity.ok(result);
    }

    @PostMapping("/conversations/{sessionId}/messages")
    public ResponseEntity<Map<String, Object>> sendMessage(
            @PathVariable String sessionId,
            @RequestBody Map<String, Object> request,
            Authentication authentication) {

        String senderId = (String) request.get("senderId");
        String receiverId = (String) request.get("receiverId");
        String content = (String) request.get("content");
        String messageType = (String) request.getOrDefault("messageType", "text");

        ChatMessage chatMessage = new ChatMessage();
        chatMessage.setId(UUID.randomUUID().toString());
        chatMessage.setConversationId(sessionId);
        chatMessage.setSenderId(senderId);
        chatMessage.setReceiverId(receiverId);
        chatMessage.setContent(content);
        chatMessage.setMessageType(messageType);
        chatMessage.setIsRead(false);
        chatMessage.setCreateTime(LocalDateTime.now());

        chatMessageRepository.save(chatMessage);

        Conversation conversation = new Conversation();
        try {
            conversation.setId(Long.parseLong(sessionId));
        } catch (NumberFormatException e) {
        }
        conversation.setLastMessageTime(LocalDateTime.now());
        messageService.updateConversation(conversation);

        Notification notification = new Notification();
        notification.setUserId(receiverId);
```

```
        notification.setType("message");
        notification.setTitle("收到新消息");
        notification.setContent(content.length() > 50 ? content.substring(0, 50) + "...
" : content);
        notification.setRelatedId(chatMessage.getId());
        notification.setRelatedType("message");
        notification.setIsRead(0);
        notificationService.createNotification(notification);

        Map<String, Object> result = new HashMap<>();
        result.put("success", true);

        Map<String, Object> dataMap = new HashMap<>();
        dataMap.put("id", chatMessage.getId());
        dataMap.put("conversationId", chatMessage.getConversationId());
        dataMap.put("senderId", chatMessage.getSenderId());
        dataMap.put("receiverId", chatMessage.getReceiverId());
        dataMap.put("content", chatMessage.getContent());
        dataMap.put("messageType", chatMessage.getMessageType());
        dataMap.put("isRead", chatMessage.getIsRead());
        dataMap.put("createTime", chatMessage.getCreateTime());

        result.put("data", dataMap);
        result.put("message", "消息发送成功");

        return ResponseEntity.ok(result);
    }

    @PutMapping("/conversations/{sessionId}/read")
    public ResponseEntity<Map<String, Object>> markMessagesAsRead(
            @PathVariable String sessionId,
            @RequestParam String userId,
            Authentication authentication) {

        chatMessageRepository.findByConversationIdOrderByCreateTimeAsc(sessionId)
                .stream()
                .filter(msg -> msg.getReceiverId().equals(userId) && !msg.getIsRead())
                .forEach(msg -> {
                    msg.setIsRead(true);
                    chatMessageRepository.save(msg);
                });

        Map<String, Object> result = new HashMap<>();
        result.put("success", true);
        result.put("message", "消息已标记为已读");

        return ResponseEntity.ok(result);
    }
}
```

八、更多前端源代码

8.1 短信登录页面

短信登录页面是用户登录的主要方式之一，文件位于 `client/lib/pages/auth/sms_login_page.dart`。SmsLoginPage 类是一个有状态组件，使用 StatefulWidget 实现。类中定义了私有变量：*formKey 表单键*，phoneController 手机号输入控制器，*verificationCodeController 验证码输入控制器*，isLoading 加载状态，*isSendingCode 发送验证码状态*，countdown 倒计时。

SmsLoginPage 类的主要功能包括：输入手机号；发送验证码；输入验证码；验证手机号和验证码；登录系统；跳转到角色选择页面或主页面。页面使用正则表达式验证手机号格式，使用倒计时功能防止频繁发送验证码。

SmsLoginPage 类使用了 Form 组件包裹所有输入字段，使用 TextFormField 组件显示输入字段，使用 ElevatedButton 组件显示登录按钮和发送验证码按钮。页面还使用了 Provider 组件管理用户认证状态，使用 Navigator 组件进行页面跳转。

```dart
import 'package:flutter/material.dart';
import 'dart:async';
import '../../theme/app_theme.dart';
import '../../providers/auth_provider.dart';
import '../../services/api_service.dart';
import 'role_selection_page.dart';
import 'package:provider/provider.dart';

class SmsLoginPage extends StatefulWidget {
  const SmsLoginPage({super.key});

  @override
  State<SmsLoginPage> createState() => _SmsLoginPageState();
}

class _SmsLoginPageState extends State<SmsLoginPage> {
  final _formKey = GlobalKey<FormState>();
  final _phoneController = TextEditingController();
  final _verificationCodeController = TextEditingController();
  bool _isLoading = false;
  bool _isSendingCode = false;
  int _countdown = 0;

  bool get _isFormValid {
    final phone = _phoneController.text.trim();
    final code = _verificationCodeController.text.trim();
    return phone.isNotEmpty &&
        RegExp(r'^1[3-9]\d{9}$').hasMatch(phone) &&
        code.isNotEmpty &&
        code.length == 6;
  }

  @override
  void initState() {
    super.initState();
    _phoneController.addListener(_updateFormState);
    _verificationCodeController.addListener(_updateFormState);
  }

  void _updateFormState() {
    setState(() {});
  }

  @override
  void dispose() {
    _phoneController.dispose();
    _verificationCodeController.dispose();
    super.dispose();
  }

  Future<void> _sendVerificationCode() async {
    final phone = _phoneController.text.trim();
    if (!RegExp(r'^1[3-9]\d{9}$').hasMatch(phone)) {
```

```
      ScaffoldMessenger.of(
        context,
      ).showSnackBar(const SnackBar(content: Text('请输入正确的手机号')));
      return;
    }

    setState(() {
      _isSendingCode = true;
    });

    try {
      await ApiService.sendVerificationCode(phone);

      if (!mounted) return;
      ScaffoldMessenger.of(
        context,
      ).showSnackBar(const SnackBar(content: Text('验证码已发送')));

      setState(() {
        _countdown = 60;
      });

      _startCountdown();
    } catch (e) {
      if (!mounted) return;
      ScaffoldMessenger.of(
        context,
      ).showSnackBar(SnackBar(content: Text(e.toString())));
    } finally {
      if (mounted) {
        setState(() {
          _isSendingCode = false;
        });
      }
    }
  }

  void _startCountdown() {
    Future.doWhile(() async {
      await Future.delayed(const Duration(seconds: 1));
      if (!mounted) return false;
      setState(() {
        _countdown--;
      });
      return _countdown > 0;
    });
  }

  Future<void> _handleLogin() async {
    if (_formKey.currentState?.validate() ?? false) {
      setState(() {
        _isLoading = true;
      });

      try {
        final phone = _phoneController.text.trim();
        final verificationCode = _verificationCodeController.text.trim();

        final authProvider = Provider.of<AuthProvider>(context, listen: false);
```

```
      final response = await ApiService.loginBySms(
        phone,
        verificationCode,
        'student',
      );

      if (response['success'] == true) {
        final data = response['data'];
        final user = data['user'];
        final token = data['token'];
        final isFirstLogin = data['isFirstLogin'] ?? false;

        await authProvider.loginWithToken(user, token, isFirstLogin: isFirstLogin);

        if (!mounted) return;

        if (isFirstLogin) {
          final userId = (user['id'] != null) ? user['id'].toString() : '';
          Navigator.pushReplacement(
            context,
            MaterialPageRoute(
              builder: (context) => RoleSelectionPage(userId: userId),
            ),
          );
        } else {
          Navigator.pushReplacementNamed(context, '/main-tab');
        }
      } else {
        throw Exception(response['message'] ?? '登录失败');
      }
    } catch (e) {
      if (!mounted) return;
      ScaffoldMessenger.of(
        context,
      ).showSnackBar(SnackBar(content: Text(e.toString())));
    } finally {
      if (mounted) {
        setState(() {
          _isLoading = false;
        });
      }
    }
  }
}

@override
Widget build(BuildContext context) {
  return Scaffold(
    backgroundColor: Colors.white,
    appBar: AppBar(
      backgroundColor: Colors.white,
      elevation: 0,
      leading: IconButton(
        icon: const Icon(Icons.arrow_back_ios_new, size: 20),
        onPressed: () => Navigator.pop(context),
        color: AppTheme.textPrimary,
      ),
      title: const Text(
        '手机号登录',
        style: TextStyle(
```

```
            fontSize: 18,
            fontWeight: FontWeight.w600,
            color: AppTheme.textPrimary,
          ),
        ),
        centerTitle: true,
      ),
      body: SafeArea(
        child: SingleChildScrollView(
          padding: const EdgeInsets.all(24),
          child: Form(
            key: _formKey,
            child: Column(
              crossAxisAlignment: CrossAxisAlignment.stretch,
              children: [
                const SizedBox(height: 40),
                const Text(
                  '欢迎使用 HiTutor',
                  style: TextStyle(
                    fontSize: 28,
                    fontWeight: FontWeight.bold,
                    color: AppTheme.textPrimary,
                  ),
                ),
                const SizedBox(height: 8),
                const Text(
                  '纯公益家教信息对接平台',
                  style: TextStyle(
                    fontSize: 14,
                    color: AppTheme.textSecondary,
                  ),
                ),
                const SizedBox(height: 48),
                _buildPhoneField(),
                const SizedBox(height: 16),
                _buildVerificationCodeField(),
                const SizedBox(height: 32),
                _buildLoginButton(),
                const SizedBox(height: 24),
                _buildAgreementText(),
              ],
            ),
          ),
        ),
      ),
    );
  }

  Widget _buildPhoneField() {
    return TextFormField(
      controller: _phoneController,
      keyboardType: TextInputType.phone,
      maxLength: 11,
      decoration: InputDecoration(
        labelText: '手机号',
        border: OutlineInputBorder(
          borderRadius: BorderRadius.circular(8),
        ),
        contentPadding: const EdgeInsets.symmetric(
          horizontal: 16,
```

```
        vertical: 12,
      ),
      counterText: '',
      prefixIcon: const Icon(Icons.phone),
    ),
    validator: (value) {
      if (value == null || value.isEmpty) {
        return '请输入手机号';
      }
      if (!RegExp(r'^1[3-9]\d{9}$').hasMatch(value)) {
        return '请输入正确的手机号';
      }
      return null;
    },
  );
}

Widget _buildVerificationCodeField() {
  return Row(
    children: [
      Expanded(
        child: TextFormField(
          controller: _verificationCodeController,
          keyboardType: TextInputType.number,
          maxLength: 6,
          decoration: InputDecoration(
            labelText: '验证码',
            border: OutlineInputBorder(
              borderRadius: BorderRadius.circular(8),
            ),
            contentPadding: const EdgeInsets.symmetric(
              horizontal: 16,
              vertical: 12,
            ),
            counterText: '',
            prefixIcon: const Icon(Icons.verified_user),
          ),
          validator: (value) {
            if (value == null || value.isEmpty) {
              return '请输入验证码';
            }
            if (value.length != 6) {
              return '验证码为 6 位数字';
            }
            return null;
          },
        ),
      ),
      const SizedBox(width: 12),
      SizedBox(
        width: 120,
        height: 48,
        child: ElevatedButton(
          onPressed: _countdown > 0 || _isSendingCode ? null : _sendVerificationCode,
          style: ElevatedButton.styleFrom(
            backgroundColor: AppTheme.primaryColor,
            shape: RoundedRectangleBorder(
              borderRadius: BorderRadius.circular(8),
            ),
          ),
```

```
          child: _isSendingCode
              ? const SizedBox(
                  width: 16,
                  height: 16,
                  child: CircularProgressIndicator(
                    color: Colors.white,
                    strokeWidth: 2,
                  ),
                )
              : Text(
                  _countdown > 0 ? '${_countdown}秒后重发' : '发送验证码',
                  style: const TextStyle(
                    fontSize: 14,
                    fontWeight: FontWeight.w500,
                  ),
                ),
        ),
      ),
    ],
  );
}

Widget _buildLoginButton() {
  return SizedBox(
    height: 48,
    child: ElevatedButton(
      onPressed: _isFormValid && !_isLoading ? _handleLogin : null,
      style: ElevatedButton.styleFrom(
        backgroundColor: AppTheme.primaryColor,
        shape: RoundedRectangleBorder(
          borderRadius: BorderRadius.circular(8),
        ),
      ),
      child: _isLoading
          ? const SizedBox(
              width: 20,
              height: 20,
              child: CircularProgressIndicator(
                color: Colors.white,
                strokeWidth: 2,
              ),
            )
          : const Text(
              '登录',
              style: TextStyle(
                fontSize: 16,
                fontWeight: FontWeight.w600,
              ),
            ),
    ),
  );
}

Widget _buildAgreementText() {
  return const Text(
    '登录即表示同意《用户协议》和《隐私政策》',
    style: TextStyle(
      fontSize: 12,
      color: AppTheme.textSecondary,
    ),
```

```
      textAlign: TextAlign.center,
    );
  }
}
```