ГУАП

КАФЕДРА № 41

ОТЧЕТ   
ЗАЩИЩЕН С ОЦЕНКОЙ

ПРЕПОДАВАТЕЛЬ

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Доцент, доцент |  |  |  | С. А. Чернышев |
| должность, уч. степень, звание |  | подпись, дата |  | инициалы, фамилия |

|  |
| --- |
| ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ №5 |
| «Проектирование локализованного в GUI кроссплатформенных приложений» |
| по курсу: «Методы объектно-ориентированного проектирования» |
|  |
|  |

РАБОТУ ВЫПОЛНИЛ

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| СТУДЕНТ гр. № |  |  |  |  |  |
|  |  |  | подпись, дата |  | инициалы, фамилия |

Санкт-Петербург 2024

1. **Цель работы**

Познакомиться с принципами проектирования локализованного графического пользовательского интерфейса кроссплатформенных приложений.

Заменить язык текста на дизайне экранов из первой лабораторной работы.

1. **Выполнение работы**

На рисунках 1 – (номер рисунка) представлены локализованные дизайны экранов.

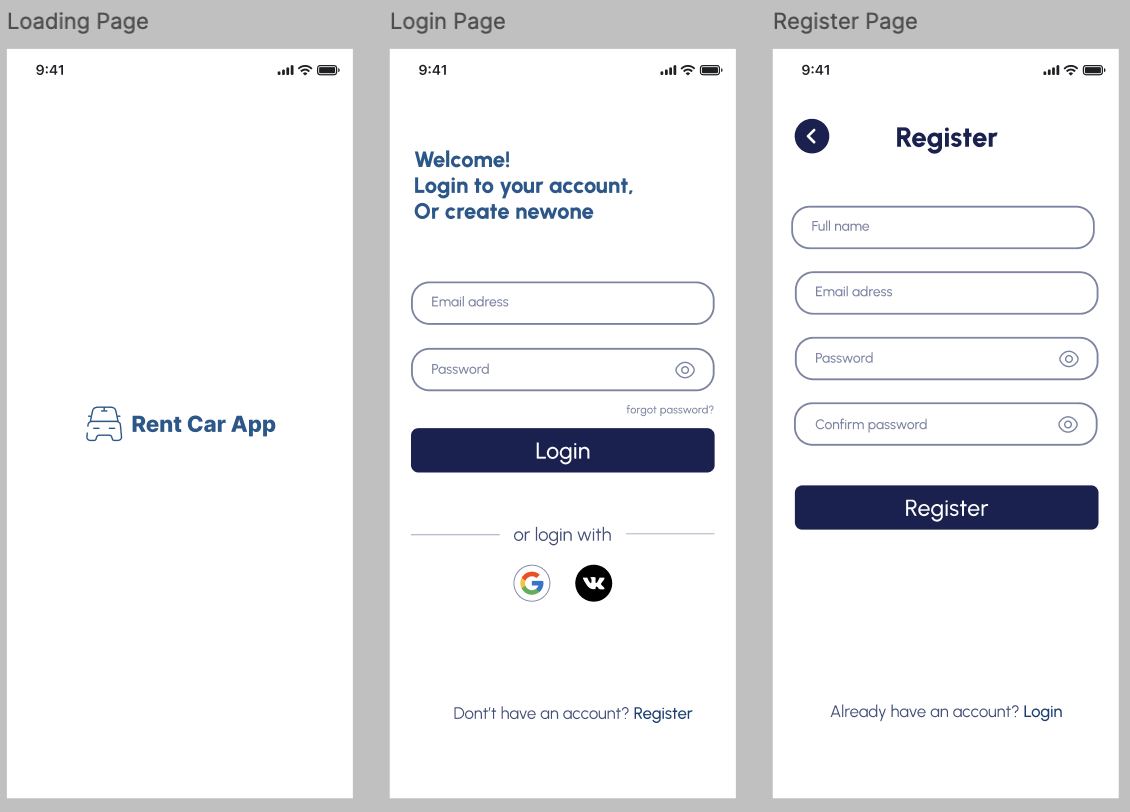


Рисунок 1 – Экраны авторизации и регистрации на английском языке

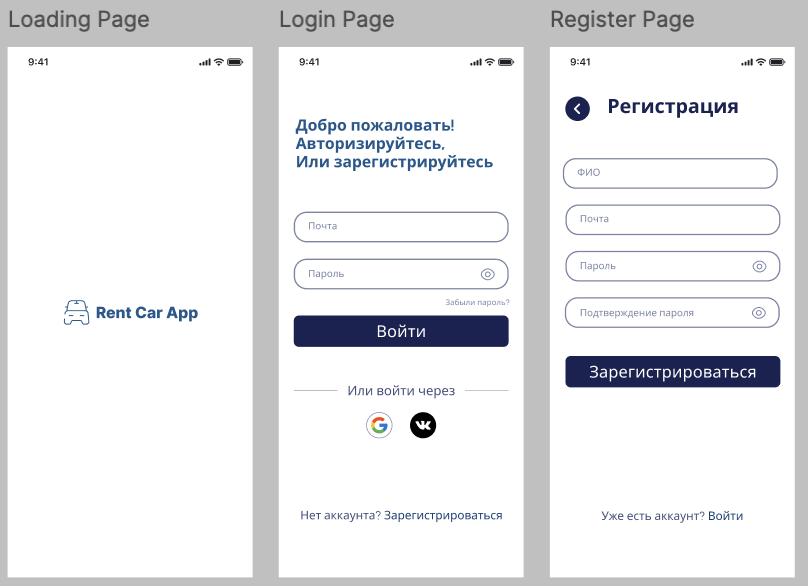


Рисунок 2 – Экраны авторизации и регистрации на русском языке



Рисунок 3 – Главный экран и экран с детальной информацией об автомобиле на английском языке



Рисунок 4 – Главный экран и экран с детальной информацией об автомобиле на русском языке

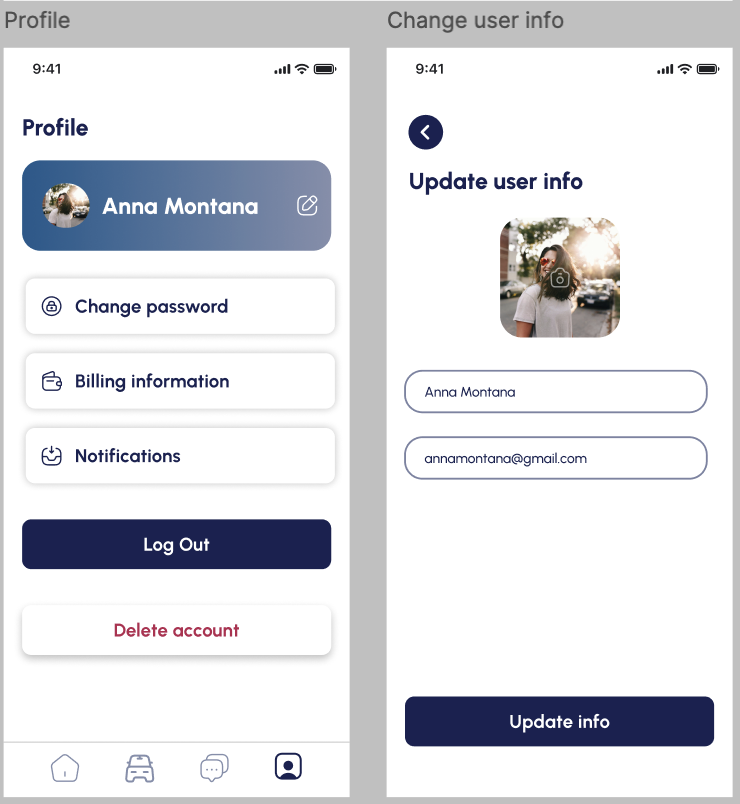


Рисунок 5 – Экран профиля и изменения данных на английском языке

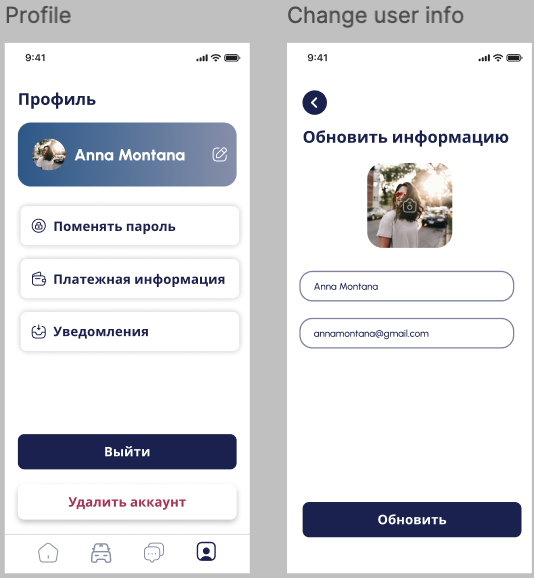


Рисунок 6 – Экран профиля и изменения данных на русском языке

**ПРИЛОЖЕНИЕ А**

**Файлы Сервера**

Листинг 1 – server.dart

|  |
| --- |
| import 'dart:convert';  import 'dart:io';  import 'package:shelf/shelf.dart';  import 'package:shelf/shelf\_io.dart' as io;  import 'package:shelf\_router/shelf\_router.dart';  import 'package:shelf\_static/shelf\_static.dart';  import 'package:server/database.dart';  import 'package:mime/mime.dart';  import 'package:shelf\_static/shelf\_static.dart';  final db = AppDatabase();  Middleware corsHeaders() {  const corsHeaders = {  'Access-Control-Allow-Origin': '\*',  'Access-Control-Allow-Methods': 'GET, POST, PUT, DELETE, OPTIONS',  'Access-Control-Allow-Headers': 'Origin, Content-Type, Authorization, Cookie',  'Access-Control-Allow-Credentials': 'true',  };  Response addCorsHeaders(Response response) =>  response.change(headers: {...response.headers, ...corsHeaders});  return (Handler handler) {  return (Request request) async {  if (request.method == 'OPTIONS') {  return Response.ok('', headers: corsHeaders);  }  final Response response = await handler(request);  return addCorsHeaders(response);  };  };  }  void main() async {  final router = Router();  final staticFilesHandler = createStaticHandler('uploads', defaultDocument: 'index.html');  router.mount('/uploads/', staticFilesHandler);    router.post('/register', (Request request) async {  final payload = await request.readAsString();  final data = Uri.splitQueryString(payload);  final fullName = data['fullName'];  final email = data['email'];  final password = data['password'];  final confirmPassword = data['confirmPassword'];  if (fullName == null || email == null || password == null || confirmPassword == null) {  return Response.badRequest(body: jsonEncode({'error': 'Missing required fields'}));  }  if (password != confirmPassword) {  return Response.badRequest(body: jsonEncode({'error': 'Passwords do not match'}));  }  final existingUser = await db.getUserByEmail(email);  if (existingUser != null) {  return Response.badRequest(body: jsonEncode({'error': 'Email already exists'}));  }  try {  final userId = await db.registerUser(fullName, email, password);    final session = await db.createSession(userId);  return Response.ok(jsonEncode({  'message': 'User registered and logged in',  'userId': userId,  'sessionId': session?.sessionId,  }));  } catch (e) {  return Response.internalServerError(body: jsonEncode({'error': 'Registration failed', 'details': e.toString()}));  }  });  router.post('/login', (Request request) async {  final payload = await request.readAsString();  final data = Uri.splitQueryString(payload);  final email = data['email'];  final password = data['password'];  if (email == null || password == null) {  return Response.badRequest(body: jsonEncode({'error': 'Missing email or password'}));  }  final user = await db.authenticateUser(email, password);  if (user != null) {  final session = await db.createSession(user.id);  return Response.ok(jsonEncode({  'message': 'Login successful',  'userId': user.id,  'fullName': user.fullName,  'sessionId': session?.sessionId,  }), headers: {  'Content-Type': 'application/json',  });  } else {  return Response.forbidden(jsonEncode({'error': 'Invalid email or password'}));  }  });  String? extractSessionId(Request request) {  final authHeader = request.headers['Authorization'];  if (authHeader != null && authHeader.startsWith('Bearer ')) {  return authHeader.substring(7);  }  return null;  }  router.post('/logout', (Request request) async {  final sessionId = extractSessionId(request);  if (sessionId == null || sessionId.isEmpty) {  return Response.badRequest(body: jsonEncode({'error': 'Не найден идентификатор сессии в заголовке Authorization'}));  }  await db.deleteSession(sessionId);  return Response.ok(jsonEncode({'message': 'Вы успешно вышли из системы'}), headers: {  'Authorization': 'Bearer ',  });  });  final protectedRouter = Router();  protectedRouter.get('/users', (Request request) async {  try {  final users = await db.getAllUsers();  final usersList = users.map((user) => {  'id': user.id,  'fullName': user.fullName,  'email': user.email,  }).toList();  return Response.ok(jsonEncode(usersList), headers: {'Content-Type': 'application/json'});  } catch (e) {  return Response.internalServerError(body: 'Failed to retrieve users: $e');  }  });  protectedRouter.get('/user', (Request request) async {  try {  final userId = request.url.queryParameters['userId'];  final id = int.tryParse(userId ?? '');  if (id == null) {  return Response.badRequest(body: jsonEncode({'error': 'Invalid or missing user ID'}));  }  final user = await db.getUserById(id);  if (user == null) {  return Response.notFound(jsonEncode({'error': 'User not found'}));  }  return Response.ok(jsonEncode({  'id': user.id,  'fullName': user.fullName,  'email': user.email,  'photo': user.photo,  }), headers: {'Content-Type': 'application/json'});  } catch (e) {  return Response.internalServerError(body: jsonEncode({'error': 'Failed to retrieve user data', 'details': e.toString()}));  }  });  protectedRouter.post('/updateUser', (Request request) async {  final boundary = request.headers['content-type']?.split('boundary=')[1];  if (boundary == null) {  return Response.badRequest(body: jsonEncode({'error': 'Invalid content type'}));  }  final transformer = MimeMultipartTransformer(boundary);  final parts = await transformer.bind(request.read()).toList();  String? fullName;  String? email;  String? photoPath;  for (final part in parts) {  final contentDisposition = part.headers['content-disposition'];  if (contentDisposition != null && contentDisposition.contains('filename=')) {  final content = await part.toList();  final fileName = contentDisposition.split('filename=')[1].replaceAll('"', '');  final fileBytes = content.expand((e) => e).toList();  final filePath = 'uploads/$fileName';  final file = File(filePath);  await file.writeAsBytes(fileBytes);  photoPath = filePath;  } else {  final field = utf8.decode(await part.expand((bytes) => bytes).toList());  if (contentDisposition!.contains('name="fullName"')) {  fullName = field;  } else if (contentDisposition.contains('name="email"')) {  email = field;  }  }  }  final sessionId = extractSessionId(request);  if (sessionId == null) {  return Response.forbidden(jsonEncode({'error': 'Not authorized'}));  }  final session = await db.getSessionById(sessionId);  if (session == null) {  return Response.forbidden(jsonEncode({'error': 'Session not found or expired'}));  }  try {  final user = await db.getUserById(session.userId);  if (user == null) {  return Response.notFound(jsonEncode({'error': 'User not found'}));  }  await db.updateUser(  id: session.userId,  fullName: fullName ?? user.fullName,  email: email ?? user.email,  photo: photoPath ?? user.photo,  );  return Response.ok(jsonEncode({'message': 'User updated successfully'}));  } catch (e) {  return Response.internalServerError(body: jsonEncode({'error': 'Failed to update user', 'details': e.toString()}));  }  });  protectedRouter.get('/cars', (Request request) async {  final languageCode = request.headers['accept-language']?.split(',').first ?? 'en';  try {  final cars = await db.getAllCars(languageCode);  return Response.ok(jsonEncode(cars), headers: {'Content-Type': 'application/json'});  } catch (e) {  return Response.internalServerError(body: 'Failed to retrieve cars: $e');  }  });  protectedRouter.get('/cars/popular', (Request request) async {  final languageCode = request.headers['accept-language']?.split(',').first ?? 'en';  try {  final cars = await db.getPopularCars(languageCode);  return Response.ok(jsonEncode(cars), headers: {'Content-Type': 'application/json'});  } catch (e) {  return Response.internalServerError(body: 'Failed to retrieve popular cars: $e');  }  });  protectedRouter.post('/addCar', (Request request) async {  final boundary = request.headers['content-type']?.split('boundary=')[1];  if (boundary == null) {  return Response.badRequest(body: jsonEncode({'error': 'Invalid content type'}));  }  final transformer = MimeMultipartTransformer(boundary);  final parts = await transformer.bind(request.read()).toList();  String? name;  double? rating;  double? reviewCount;  double? rentalPricePerDay;  bool? isPopular;  String? localizedDescription; // JSON  String? localizedFuelType; // JSON  String? localizedColor; // JSON  String? driveType;  String? engineType;  int? power;  List<String> photoPaths = [];  for (final part in parts) {  final contentDisposition = part.headers['content-disposition'];  if (contentDisposition != null && contentDisposition.contains('filename=')) {  // Сохраняем файл  final content = await part.toList();  final fileName = contentDisposition.split('filename=')[1].replaceAll('"', '');  final fileBytes = content.expand((e) => e).toList();  final filePath = 'uploads/$fileName';  final file = File(filePath);  await file.writeAsBytes(fileBytes);  photoPaths.add(filePath);  } else {  // Обрабатываем текстовые данные  final field = utf8.decode(await part.expand((bytes) => bytes).toList());  if (contentDisposition!.contains('name="name"')) {  name = field;  } else if (contentDisposition.contains('name="rating"')) {  rating = double.parse(field);  } else if (contentDisposition.contains('name="reviewCount"')) {  reviewCount = double.parse(field);  } else if (contentDisposition.contains('name="rentalPricePerDay"')) {  rentalPricePerDay = double.parse(field);  } else if (contentDisposition.contains('name="isPopular"')) {  isPopular = field == 'true';  } else if (contentDisposition.contains('name="localizedDescription"')) {  localizedDescription = field; // Ожидаем JSON  } else if (contentDisposition.contains('name="localizedFuelType"')) {  localizedFuelType = field; // Ожидаем JSON  } else if (contentDisposition.contains('name="localizedColor"')) {  localizedColor = field; // Ожидаем JSON  } else if (contentDisposition.contains('name="driveType"')) {  driveType = field;  } else if (contentDisposition.contains('name="engineType"')) {  engineType = field; // Новый параметр  } else if (contentDisposition.contains('name="power"')) {  power = int.parse(field); // Новый параметр  }  }  }  // Проверяем, что обязательные поля переданы  if (name == null ||  rating == null ||  reviewCount == null ||  rentalPricePerDay == null ||  isPopular == null ||  localizedDescription == null ||  localizedFuelType == null ||  localizedColor == null ||  driveType == null || engineType == null || power == null) {  return Response.badRequest(body: jsonEncode({'error': 'Missing required fields'}));  }  try {  // Преобразуем JSON данные в Map<String, String>  final Map<String, String> descriptions =  (jsonDecode(localizedDescription) as Map<String, dynamic>)  .map((key, value) => MapEntry(key, value.toString()));  final Map<String, String> fuelTypes =  (jsonDecode(localizedFuelType) as Map<String, dynamic>)  .map((key, value) => MapEntry(key, value.toString()));  final Map<String, String> colors =  (jsonDecode(localizedColor) as Map<String, dynamic>)  .map((key, value) => MapEntry(key, value.toString()));  // Добавляем машину в базу данных  await db.addCar(  photos: jsonEncode(photoPaths), // Сериализуем список в JSON  name: name,  rating: rating,  reviewCount: reviewCount,  rentalPricePerDay: rentalPricePerDay,  isPopular: isPopular,  localizedDescriptions: descriptions,  localizedFuelTypes: fuelTypes,  localizedColors: colors,  driveType: driveType,  engineType: engineType, // Новый параметр  power: power,  );  return Response.ok(jsonEncode({'message': 'Car added successfully'}));  } catch (e) {  return Response.internalServerError(  body: jsonEncode({'error': 'Failed to add car', 'details': e.toString()}));  }  });  protectedRouter.get('/promotions', (Request request) async {  final languageCode = request.headers['accept-language']?.split(',').first ?? 'en';  try {  final promotions = await db.getAllPromotions(languageCode);  return Response.ok(jsonEncode(promotions), headers: {'Content-Type': 'application/json'});  } catch (e) {  return Response.internalServerError(body: 'Failed to retrieve promotions: $e');  }  });  protectedRouter.post('/addPromotion', (Request request) async {  final boundary = request.headers['content-type']?.split('boundary=')[1];  if (boundary == null) {  return Response.badRequest(body: jsonEncode({'error': 'Invalid content type'}));  }  final transformer = MimeMultipartTransformer(boundary);  final parts = await transformer.bind(request.read()).toList();  String? localizedName; // JSON  String? photoPath;  for (final part in parts) {  final contentDisposition = part.headers['content-disposition'];  if (contentDisposition != null && contentDisposition.contains('filename=')) {  try {  final content = await part.toList();  final fileName = contentDisposition.split('filename=')[1].replaceAll('"', '');  final fileBytes = content.expand((e) => e).toList();  final filePath = 'uploads/$fileName';  final file = File(filePath);  await file.writeAsBytes(fileBytes);  photoPath = filePath; // Путь к загруженному файлу  } catch (e) {  return Response.internalServerError(  body: jsonEncode({'error': 'Failed to upload file', 'details': e.toString()}));  }  } else {  final field = utf8.decode(await part.expand((bytes) => bytes).toList());  if (contentDisposition!.contains('name="localizedName"')) {  localizedName = field; // JSON строка  }  }  }  // Проверяем, что обязательные поля заполнены  if (localizedName == null || photoPath == null) {  return Response.badRequest(body: jsonEncode({'error': 'Missing required fields'}));  }  try {  // Преобразуем JSON в Map<String, String>  final Map<String, String> localizedNames =  (jsonDecode(localizedName) as Map<String, dynamic>)  .map((key, value) => MapEntry(key, value.toString()));  // Добавляем промоакцию в базу  await db.addPromotion(  localizedNames: localizedNames,  photo: photoPath,  );  return Response.ok(jsonEncode({'message': 'Promotion added successfully'}));  } catch (e) {  return Response.internalServerError(  body: jsonEncode({'error': 'Failed to add promotion', 'details': e.toString()}));  }  });    Middleware sessionChecker() {  return (Handler handler) {  return (Request request) async {  final sessionId = extractSessionId(request);  if (sessionId == null || sessionId.isEmpty) {  return Response.forbidden(jsonEncode({'error': 'Not authorized'}));  }  final session = await db.getSessionById(sessionId);  if (session == null) {  return Response.forbidden(jsonEncode({'error': 'Session not found or expired'}));  }  final now = DateTime.now();  final inactiveDuration = now.difference(session.lastUsed);  if (inactiveDuration > Duration(minutes: 15)) {  await db.deleteSession(sessionId);  return Response.forbidden(jsonEncode({'error': 'Session expired due to inactivity'}));  }  await db.updateSessionLastUsed(sessionId, now);  return handler(request);  };  };  }  final protectedHandler = Pipeline()  .addMiddleware(sessionChecker())  .addHandler(protectedRouter.call);  final handler = Pipeline()  .addMiddleware(logRequests())  .addMiddleware(corsHeaders())  .addHandler((Router()  ..mount('/', router.call)  ..mount('/', protectedHandler)).call);  final server = await io.serve(handler, InternetAddress.anyIPv4, 8080);  print('Server running on http://localhost:${server.port}');  } |

Листинг 2 – database.dart

|  |
| --- |
| import 'package:drift/drift.dart';  import 'package:drift/native.dart';  import 'package:path/path.dart' as p;  import 'dart:io';  import 'package:crypto/crypto.dart';  import 'dart:convert';  part 'database.g.dart';  @DataClassName('Session')  class Sessions extends Table {  IntColumn get id => integer().autoIncrement()();  TextColumn get sessionId => text().customConstraint('UNIQUE')();  IntColumn get userId => integer().customConstraint('REFERENCES users(id)')();  DateTimeColumn get expiresAt => dateTime()();  DateTimeColumn get lastUsed => dateTime().withDefault(currentDateAndTime)();  }  @DataClassName('User')  class Users extends Table {  IntColumn get id => integer().autoIncrement()();  TextColumn get fullName => text()();  TextColumn get email => text().customConstraint('UNIQUE')();  TextColumn get passwordHash => text()();  TextColumn get photo => text().nullable()();  }  @DataClassName('Promotion')  class Promotions extends Table {  IntColumn get id => integer().autoIncrement()();  TextColumn get localizedName => text()();  TextColumn get photo => text()();  }  @DataClassName('Car')  class Cars extends Table {  IntColumn get id => integer().autoIncrement()();  TextColumn get photos => text()();  TextColumn get name => text()();  RealColumn get rating => real()();  RealColumn get reviewCount => real()();  RealColumn get rentalPricePerDay => real()();  BoolColumn get isPopular => boolean()();  TextColumn get localizedDescription => text()();  TextColumn get engineType => text()();  IntColumn get power => integer()();  TextColumn get localizedFuelType => text()();  TextColumn get localizedColor => text()();  TextColumn get driveType => text()();  }  String extractLocalizedText(String localizedTextJson, String languageCode) {  if (localizedTextJson.isEmpty) {  return 'Translation missing';  }  try {  final Map<String, dynamic> translations = jsonDecode(localizedTextJson);  return translations[languageCode] ?? translations['en'] ?? 'Translation missing';  } catch (e) {  return 'Translation missing';  }  }  @DriftDatabase(tables: [Users, Cars, Promotions, Sessions])  class AppDatabase extends \_$AppDatabase {  AppDatabase() : super(\_openConnection());  @override  int get schemaVersion => 1;  @override  MigrationStrategy get migration => MigrationStrategy(  onCreate: (Migrator m) async {  await m.createAll();  },  onUpgrade: (Migrator m, int from, int to) async {  },  );  String hashPassword(String password) {  return sha256.convert(utf8.encode(password)).toString();  }  Future<int> registerUser(String fullName, String email, String password) async {  final user = UsersCompanion(  fullName: Value(fullName),  email: Value(email),  passwordHash: Value(hashPassword(password)),  );  return into(users).insert(user);  }  Future<User?> authenticateUser(String email, String password) async {  final query = select(users)..where((tbl) => tbl.email.equals(email));  final user = await query.getSingleOrNull();  if (user != null && user.passwordHash == hashPassword(password)) {  return user;  }  return null;  }  Future<List<User>> getAllUsers() async {  return select(users).get();  }  Future<User?> getUserByEmail(String email) async {  final query = select(users)..where((tbl) => tbl.email.equals(email));  return await query.getSingleOrNull();  }  Future<User?> getUserById(int id) async {  final query = select(users)..where((tbl) => tbl.id.equals(id));  return await query.getSingleOrNull();  }  Future<void> updateUser({  required int id,  required String fullName,  required String email,  String? photo,  }) async {  final user = UsersCompanion(  fullName: Value(fullName),  email: Value(email),  photo: Value(photo ?? ''),  );  await (update(users)..where((tbl) => tbl.id.equals(id))).write(user);  }  Future<List<Map<String, dynamic>>> getAllCars(String languageCode) async {  final carList = await select(cars).get();  return carList.map((car) {  return {  'id': car.id,  'photos': jsonDecode(car.photos),  'name': car.name,  'rating': car.rating,  'reviewCount': car.reviewCount,  'rentalPricePerDay': car.rentalPricePerDay,  'isPopular': car.isPopular,  'description': extractLocalizedText(car.localizedDescription, languageCode),  'engineType': car.engineType,  'power': car.power,  'fuelType': extractLocalizedText(car.localizedFuelType, languageCode),  'color': extractLocalizedText(car.localizedColor, languageCode),  'driveType': car.driveType,  };  }).toList();  }  Future<List<Map<String, dynamic>>> getPopularCars(String languageCode) async {  final popularCars = await (select(cars)..where((tbl) => tbl.isPopular.equals(true))).get();  return popularCars.map((car) {  return {  'id': car.id,  'photos': jsonDecode(car.photos),  'name': car.name,  'rating': car.rating,  'reviewCount': car.reviewCount,  'rentalPricePerDay': car.rentalPricePerDay,  'isPopular': car.isPopular,  'description': extractLocalizedText(car.localizedDescription, languageCode),  'engineType': car.engineType,  'power': car.power,  'fuelType': extractLocalizedText(car.localizedFuelType, languageCode),  'color': extractLocalizedText(car.localizedColor, languageCode),  'driveType': car.driveType,  };  }).toList();  }  Future<int> addCar({  required String photos,  required String name,  required double rating,  required double reviewCount,  required double rentalPricePerDay,  required bool isPopular,  required Map<String, String> localizedDescriptions,  required Map<String, String> localizedFuelTypes,  required Map<String, String> localizedColors,  required String engineType,  required int power,  required String driveType,  }) async {  final car = CarsCompanion(  photos: Value(photos),  name: Value(name),  rating: Value(rating),  reviewCount: Value(reviewCount),  rentalPricePerDay: Value(rentalPricePerDay),  isPopular: Value(isPopular),  localizedDescription: Value(jsonEncode(localizedDescriptions)),  localizedFuelType: Value(jsonEncode(localizedFuelTypes)),  localizedColor: Value(jsonEncode(localizedColors)),  engineType: Value(engineType),  power: Value(power),  driveType: Value(driveType),  );  return into(cars).insert(car);  }  Future<List<Map<String, dynamic>>> getAllPromotions(String languageCode) async {  final promotionList = await select(promotions).get();  return promotionList.map((promo) {  return {  'id': promo.id,  'name': extractLocalizedText(promo.localizedName, languageCode),  'photo': promo.photo,  };  }).toList();  }  Future<int> addPromotion({  required Map<String, String> localizedNames,  required String photo,  }) async {  final promotion = PromotionsCompanion(  localizedName: Value(jsonEncode(localizedNames)),  photo: Value(photo),  );  return into(promotions).insert(promotion);  }  Future<Session?> createSession(int userId) async {  final sessionId = \_generateSessionId(userId);  final expiresAt = DateTime.now().add(Duration(minutes: 15));  final lastUsed = DateTime.now();  final session = SessionsCompanion(  sessionId: Value(sessionId),  userId: Value(userId),  expiresAt: Value(expiresAt),  lastUsed: Value(lastUsed),  );  final sessionIdInserted = await into(sessions).insert(session);  return (select(sessions)..where((tbl) => tbl.id.equals(sessionIdInserted))).getSingleOrNull();  }  String \_generateSessionId(int userId) {  final bytes = utf8.encode(DateTime.now().toString() + userId.toString());  return sha256.convert(bytes).toString();  }  Future<Session?> getSessionById(String sessionId) async {  final query = select(sessions)..where((tbl) => tbl.sessionId.equals(sessionId));  return await query.getSingleOrNull();  }  Future<void> updateSessionLastUsed(String sessionId, DateTime lastUsed) async {  await (update(sessions)  ..where((tbl) => tbl.sessionId.equals(sessionId)))  .write(SessionsCompanion(lastUsed: Value(lastUsed)));  }  Future<void> deleteSession(String sessionId) async {  await (delete(sessions)..where((tbl) => tbl.sessionId.equals(sessionId))).go();  }  }  LazyDatabase \_openConnection() {  return LazyDatabase(() async {  final dbFolder = Directory.current.path;  final file = File(p.join(dbFolder, 'app.db'));  return NativeDatabase(file);  });  } |

Листинг 3 – pubspec.yaml

|  |
| --- |
| name: server  description: A sample command-line application.  version: 1.0.0  # repository: https://github.com/my\_org/my\_repo  environment:  sdk: ^3.5.3  # Add regular dependencies here.  dependencies:  path: ^1.8.0  shelf: ^1.3.0  shelf\_router: ^1.0.0  shelf\_static: ^1.1.0  drift: ^2.7.0  sqflite\_common: ^2.3.0  dev\_dependencies:  lints: ^5.0.0  test: ^1.24.0  build\_runner: ^2.1.0  drift\_dev: ^2.3.0  shelf\_swagger\_ui: ^1.0.0+2  openapi\_generator: |

**Файлы Клиента**

Листинг 4 – main.dart

|  |
| --- |
| import 'package:flutter/material.dart';  import 'Pages/login\_page.dart';  import 'Pages/register\_page.dart';  import 'Pages/loading\_screen.dart';  import 'Pages/tab\_bar.dart';  import 'Pages/edit\_user\_page.dart';  import 'Pages/detail\_page.dart';  import 'Models/car\_model.dart';  import 'package:flutter\_localizations/flutter\_localizations.dart';  import 'generated/l10n.dart';  void main() {  runApp(const MyApp());  }  class MyApp extends StatelessWidget {  const MyApp({super.key});  @override  Widget build(BuildContext context) {  return MaterialApp(  localizationsDelegates: const [  S.delegate,  GlobalMaterialLocalizations.delegate,  GlobalWidgetsLocalizations.delegate,  GlobalCupertinoLocalizations.delegate,  ],  supportedLocales: S.delegate.supportedLocales,  title: 'Flutter Demo',  theme: ThemeData(  colorScheme: ColorScheme.fromSeed(seedColor: Colors.white),  useMaterial3: true,  ),  initialRoute: '/',  routes: {  '/': (context) => const LoadingScreen(),  '/login': (context) => const LoginPage(),  '/register': (context) => const RegisterPage(),  '/customTabBar': (context) => const CustomBottomNavigationBar(),  '/editUserPage': (context) => const EditUserPage(),  },  onGenerateRoute: (settings) {  if (settings.name == '/detailPage') {  final car = settings.arguments as Car;    return PageRouteBuilder(  pageBuilder: (context, animation, secondaryAnimation) => DetailPage(car: car),  transitionsBuilder: (context, animation, secondaryAnimation, child) {  const begin = Offset(0.0, 1.0);  const end = Offset.zero;  const curve = Curves.fastEaseInToSlowEaseOut;  var tween = Tween(begin: begin, end: end).chain(CurveTween(curve: curve));  var offsetAnimation = animation.drive(tween);  return SlideTransition(  position: offsetAnimation,  child: child,  );  },  );  }  return null;  },  );  }  } |

Листинг 5 – loading\_screen.dart

|  |
| --- |
| import 'package:flutter/material.dart';  import 'login\_page.dart';  import 'tab\_bar.dart';  import '../Services/auth\_service.dart';  class LoadingScreen extends StatefulWidget {  const LoadingScreen({super.key});  @override  \_LoadingScreenState createState() => \_LoadingScreenState();  }  class \_LoadingScreenState extends State<LoadingScreen> {  final NetworkService \_authService = NetworkService();  @override  void initState() {  super.initState();  \_attemptAutoLogin();  }  Future<void> \_attemptAutoLogin() async {  final result = await \_authService.autoLogin();  Future.delayed(const Duration(seconds: 3), () {  if (result['success']) {  Navigator.pushReplacement(  context,  MaterialPageRoute(builder: (context) => const CustomBottomNavigationBar()),  );  } else {  Navigator.pushReplacement(  context,  MaterialPageRoute(builder: (context) => const LoginPage()),  );  }  });  }  @override  Widget build(BuildContext context) {  return Scaffold(  body: Container(  color: Colors.white,  child: Center(  child: Row(  mainAxisAlignment: MainAxisAlignment.center,  children: [  Image.asset(  'assets/images/car.png',  width: 50,  height: 50,  color: const Color(0xFF1B588C),  colorBlendMode: BlendMode.srcATop,  ),  const SizedBox(width: 10),  const Text(  "Rent Car App",  style: TextStyle(  fontSize: 24,  fontWeight: FontWeight.w900,  color: Color(0xFF1B588C),  ),  )  ],  ),  ),  ),  );  }  } |

Листинг 6 – login\_page.dart

|  |
| --- |
| import 'package:flutter/material.dart';  import '../Services/auth\_service.dart';  import 'tab\_bar.dart';  import '../Components/text\_field.dart';  import 'package:flutter/foundation.dart';  import 'dart:io';  import '../generated/l10n.dart';  class LoginPage extends StatefulWidget {  const LoginPage({super.key});  @override  \_LoginPageState createState() => \_LoginPageState();  }  class \_LoginPageState extends State<LoginPage> {  final TextEditingController \_emailController = TextEditingController();  final TextEditingController \_passwordController = TextEditingController();  final NetworkService \_authService = NetworkService();  bool isButtonEnabled = false;  @override  void initState() {  super.initState();  \_emailController.addListener(\_checkFields);  \_passwordController.addListener(\_checkFields);  }  Future<void> \_login() async {  final email = \_emailController.text;  final password = \_passwordController.text;  final result = await \_authService.login(email, password);  if (result['success']) {  Navigator.pushAndRemoveUntil(  context,  MaterialPageRoute(builder: (context) => const CustomBottomNavigationBar()),  (Route<dynamic> route) => false,  );  } else {  showDialog(  context: context,  builder: (context) {  return AlertDialog(  title: Text(S.of(context).loginFailed),  content: Text(result['error'] ?? S.of(context).loginFailedPleaseTryAgain),  actions: [  TextButton(  onPressed: () {  Navigator.of(context).pop();  },  child: const Text('OK'),  ),  ],  );  },  );  }  }  void \_checkFields() {  setState(() {  isButtonEnabled =  \_emailController.text.isNotEmpty &&  \_passwordController.text.isNotEmpty;  });  }  @override  Widget build(BuildContext context) {  bool isMobile = !kIsWeb && (Platform.isAndroid || Platform.isIOS);  Widget content = Column(  mainAxisAlignment: MainAxisAlignment.center,  crossAxisAlignment: CrossAxisAlignment.start,  children: [  const SizedBox(height: 82),  Text(  S.of(context).welcomeLogin,  style: TextStyle(  fontSize: 24,  fontWeight: FontWeight.bold,  color: Color(0xFF1B588C),  ),  ),  const SizedBox(height: 62),  CustomTextField(  controller: \_emailController,  labelText: S.of(context).emailAdress  ),  const SizedBox(height: 25),  CustomTextField(  controller: \_passwordController,  labelText: S.of(context).password,  isObscure: true,  ),  Align(  alignment: Alignment.centerRight,  child: TextButton(  onPressed: () {},  child: Text(  S.of(context).forgotPassword,  style: TextStyle(  color: const Color.fromARGB(255, 178, 188, 222).withOpacity(0.7),  fontFamily: "Urbanist",  fontSize: 12,  fontWeight: FontWeight.w600,  ),  ),  ),  ),  const SizedBox(height: 9),  Center(  child: Opacity(  opacity: isButtonEnabled ? 1.0 : 0.5,  child: ElevatedButton(  onPressed: isButtonEnabled ? \_login : null,  style: ElevatedButton.styleFrom(  backgroundColor: const Color(0xFF192252),  shape: RoundedRectangleBorder(  borderRadius: BorderRadius.circular(8),  ),  minimumSize: const Size(double.infinity, 50),  ),  child: Text(  S.of(context).login,  style: TextStyle(  fontFamily: "Urbanist",  fontSize: 25,  color: Colors.white,  fontWeight: FontWeight.w600,  ),  ),  ),  ),  ),  const SizedBox(height: 53),  Row(  children: [  Expanded(  child: Divider(  color: const Color(0xFF424F7B).withOpacity(0.5),  ),  ),  Padding(  padding: EdgeInsets.symmetric(horizontal: 8.0),  child: Text(  S.of(context).orLoginWith,  style: TextStyle(  color: Color(0xFF424F7B),  fontFamily: "Urbanist",  fontSize: 20,  ),  ),  ),  Expanded(  child: Divider(  color: const Color(0xFF424F7B).withOpacity(0.5),  ),  ),  ],  ),  const SizedBox(height: 20),  Row(  mainAxisAlignment: MainAxisAlignment.center,  children: [  SizedBox(  width: 50,  height: 50,  child: IconButton(  icon: Image.asset('assets/images/google\_icon.png'),  onPressed: () {},  ),  ),  const SizedBox(width: 20),  SizedBox(  width: 50,  height: 50,  child: IconButton(  icon: Image.asset('assets/images/vk\_icon.png'),  onPressed: () {},  ),  ),  ],  ),  isMobile  ? const Spacer()  : const SizedBox(height: 100),  Center(  child: TextButton(  onPressed: () {  Navigator.pushNamed(context, '/register');  },  child: RichText(  text: TextSpan(  text: S.of(context).dontHaveAnAccount,  style: TextStyle(  fontFamily: "Urbanist",  color: Color(0xFF424F7B),  fontSize: 18,  ),  children: <TextSpan>[  TextSpan(  text: S.of(context).registerBtn,  style: TextStyle(  fontFamily: "Urbanist",  color: Color(0xFF103F74),  fontWeight: FontWeight.bold,  ),  ),  ],  ),  ),  ),  ),  ],  );  return Scaffold(  body: Container(  color: Colors.white,  child: Padding(  padding: const EdgeInsets.all(20.0),  child: isMobile  ? content  : SingleChildScrollView(  child: content,  ),  ),  ),  );  }  @override  void dispose() {  \_emailController.dispose();  \_passwordController.dispose();  super.dispose();  }  } |

Листинг 7 – register\_page.dart

|  |
| --- |
| import 'package:flutter/material.dart';  import '../Services/auth\_service.dart';  import 'tab\_bar.dart';  import '../Components/text\_field.dart';  import 'package:flutter/foundation.dart';  import 'dart:io';  import '../generated/l10n.dart';  class RegisterPage extends StatefulWidget {  const RegisterPage({super.key});  @override  \_RegisterPageState createState() => \_RegisterPageState();  }  class \_RegisterPageState extends State<RegisterPage> {  final TextEditingController \_fullNameController = TextEditingController();  final TextEditingController \_emailController = TextEditingController();  final TextEditingController \_passwordController = TextEditingController();  final TextEditingController \_confirmPasswordController = TextEditingController();  final NetworkService \_authService = NetworkService();  bool isButtonEnabled = false;  @override  void initState() {  super.initState();  \_fullNameController.addListener(\_checkFields);  \_emailController.addListener(\_checkFields);  \_passwordController.addListener(\_checkFields);  \_confirmPasswordController.addListener(\_checkFields);  }  void \_checkFields() {  setState(() {  isButtonEnabled =  \_fullNameController.text.isNotEmpty &&  \_emailController.text.isNotEmpty &&  \_passwordController.text.isNotEmpty &&  \_confirmPasswordController.text.isNotEmpty;  });  }  Future<void> \_register() async {  final fullName = \_fullNameController.text;  final email = \_emailController.text;  final password = \_passwordController.text;  final confirmPassword = \_confirmPasswordController.text;  final result = await \_authService.register(fullName, email, password, confirmPassword);  if (result['success']) {  Navigator.pushAndRemoveUntil(  context,  MaterialPageRoute(builder: (context) => const CustomBottomNavigationBar()),  (Route<dynamic> route) => false,  );  } else {  showDialog(  context: context,  builder: (context) {  return AlertDialog(  title: Text(S.of(context).registrationFailed),  content: Text(result['error'] ?? S.of(context).registrationFailedPleaseTryAgain),  actions: [  TextButton(  onPressed: () {  Navigator.of(context).pop();  },  child: const Text('OK'),  ),  ],  );  },  );  }  }  @override  Widget build(BuildContext context) {  bool isMobile = !kIsWeb && (Platform.isAndroid || Platform.isIOS);  Widget content = Column(  crossAxisAlignment: CrossAxisAlignment.start,  children: [  const SizedBox(height: 40),  CustomTextField(  controller: \_fullNameController,  labelText: S.of(context).fullName  ),  const SizedBox(height: 24),  CustomTextField(  controller: \_emailController,  labelText: S.of(context).emailAdress  ),  const SizedBox(height: 24),  CustomTextField(  controller: \_passwordController,  labelText: S.of(context).password,  isObscure: true,  ),  const SizedBox(height: 24),  CustomTextField(  controller: \_confirmPasswordController,  labelText: S.of(context).confirmPassword,  isObscure: true,  ),  const SizedBox(height: 43),  Center(  child: Opacity(  opacity: isButtonEnabled ? 1.0 : 0.5,  child: ElevatedButton(  onPressed: isButtonEnabled ? \_register : null,  style: ElevatedButton.styleFrom(  backgroundColor: const Color(0xFF192252),  shape: RoundedRectangleBorder(  borderRadius: BorderRadius.circular(8),  ),  minimumSize: const Size(double.infinity, 50),  ),  child: Text(  S.of(context).registerBtn,  style: TextStyle(  fontFamily: "Urbanist",  fontSize: 25,  color: Colors.white,  fontWeight: FontWeight.w600,  ),  ),  ),  ),  ),  isMobile  ? const Spacer()  : const SizedBox(height: 200),  Center(  child: TextButton(  onPressed: () {  Navigator.pop(context);  },  child: RichText(  text: TextSpan(  text: S.of(context).alreadyHaveAnAccount,  style: TextStyle(  fontFamily: "Urbanist",  color: Color(0xFF424F7B),  fontSize: 18,  ),  children: <TextSpan>[  TextSpan(  text: S.of(context).login,  style: TextStyle(  fontFamily: "Urbanist",  color: Color(0xFF103F74),  fontWeight: FontWeight.bold,  ),  ),  ],  ),  ),  ),  ),  ],  );  return Scaffold(  appBar: AppBar(  backgroundColor: Colors.white,  title: Text(  S.of(context).registerTitle,  style: TextStyle(  fontSize: 30,  fontWeight: FontWeight.bold,  color: Color(0xFF192252)  ),  ),  elevation: 0,  leading: SizedBox(  width: 45,  height: 45,  child: IconButton(  icon: Image.asset('assets/images/backIcon.png'),  onPressed: () {  Navigator.pop(context);  },  ),  ),  ),  body: Container(  color: Colors.white,  child: Padding(  padding: const EdgeInsets.all(20.0),  child: isMobile  ? content  : SingleChildScrollView(  child: content,  ),  ),  ),  );  }  @override  void dispose() {  \_fullNameController.dispose();  \_emailController.dispose();  \_passwordController.dispose();  \_confirmPasswordController.dispose();  super.dispose();  }  } |

Листинг 8 – home\_page.dart

|  |
| --- |
| import 'package:flutter/material.dart';  import 'package:smooth\_page\_indicator/smooth\_page\_indicator.dart';  import '../Services/auth\_service.dart';  import '../Models/car\_model.dart';  import 'package:intl/intl.dart';  import 'package:flutter/foundation.dart';  import 'dart:io';  import '../generated/l10n.dart';  class HomePage extends StatefulWidget {  const HomePage({super.key});  @override  \_HomePageState createState() => \_HomePageState();  }  class \_HomePageState extends State<HomePage> {  final PageController \_pageController = PageController();  NetworkService networkService = NetworkService();  List<Map<String, dynamic>> allPromotions = [];  List<Car> popularCars = [];  List<Car> allCars = [];  bool isLoading = true;  final String baseURL = 'http://localhost:8080/';  List<Car> convertToCarList(List<Map<String, dynamic>> carData) {  return carData.map((data) {  return Car(  name: data['name'],  imageUrl: data['photos'].isNotEmpty ? List<String>.from(data['photos'].map((photo) => baseURL + photo)) : [''],  rentalPricePerDay: (data['rentalPricePerDay'] as num).toInt(),  rating: data['rating'].toDouble(),  reviewCount: (data['reviewCount'] as num).toInt(),  description: data['description'],  engine: data['engineType'],  power: data['power'],  fuel: data['fuelType'],  color: data['color'],  drivetrain: data['driveType'],  );  }).toList();  }  @override  void initState() {  super.initState();  fetchPromotions();  fetchCarsData();  }  Future<void> fetchCarsData() async {  List<Map<String, dynamic>> popular = await networkService.getPopularCars();  List<Map<String, dynamic>> all = await networkService.getAllCars();  setState(() {  popularCars = convertToCarList(popular);  allCars = convertToCarList(all);  isLoading = false;  });  }  Future<void> fetchPromotions() async {  List<Map<String, dynamic>> promotions = await networkService.getPromotions();  setState(() {  allPromotions = promotions;  isLoading = false;  });  }  @override  void dispose() {  \_pageController.dispose();  super.dispose();  }  Future<void> \_refreshData() async {  setState(() {  isLoading = true;  });  await fetchPromotions();  await fetchCarsData();  setState(() {  isLoading = false;  });  }    @override  Widget build(BuildContext context) {  bool isMobile = !kIsWeb && (Platform.isAndroid || Platform.isIOS);  return Scaffold(  body: Stack(  children: [  Positioned.fill(  top: 117,  left: 8,  right: 8,  bottom: 50,  child: isLoading  ? const Center(child: CircularProgressIndicator())  : RefreshIndicator(  onRefresh: \_refreshData,  child: Container(  color: Colors.white,  child: SingleChildScrollView(  padding: EdgeInsets.only(top: 10),  child: Column(  crossAxisAlignment: CrossAxisAlignment.start,  children: [  LayoutBuilder(  builder: (context, constraints) {  double screenWidth = constraints.maxWidth;  double blockHeight = (screenWidth / 3).clamp(200, 400);  double imageHeight = (blockHeight - 50).clamp(150, 350);    return SizedBox(  height: blockHeight,  child: PageView.builder(  controller: \_pageController,  itemCount: allPromotions.length,  onPageChanged: (int index) {  setState(() {});  },  itemBuilder: (context, index) {  final promotion = allPromotions[index];  final imageUrl = promotion['photo'] ?? '';  return Container(  margin: const EdgeInsets.symmetric(horizontal: 8.0),  child: Column(  crossAxisAlignment: CrossAxisAlignment.start,  children: [  Container(  height: imageHeight,  decoration: BoxDecoration(  borderRadius: BorderRadius.circular(16),  color: imageUrl.isNotEmpty ? null : Colors.grey,  image: imageUrl.isNotEmpty ? DecorationImage(  image: NetworkImage('$baseURL$imageUrl'),  fit: BoxFit.cover) : null,  ),  ),  const SizedBox(height: 8),  Text(  promotion['name'] ?? 'No Name',  style: const TextStyle(  fontSize: 20,  fontWeight: FontWeight.bold,  fontFamily: "Urbanist",  ),  maxLines: 1,  overflow: TextOverflow.ellipsis,  ),  ],  ),  );  },  ),  );  },  ),  const SizedBox(height: 10),  Align(  alignment: Alignment.centerLeft,  child: Padding(  padding: const EdgeInsets.symmetric(horizontal: 8.0),  child: allPromotions.length > 1 ? SmoothPageIndicator(  controller: \_pageController,  count: allPromotions.length,  effect: const ExpandingDotsEffect(  expansionFactor: 4.0,  activeDotColor: Color(0xFF192252),  dotColor: Color(0xFFD9D9D9),  dotHeight: 8,  dotWidth: 8,  spacing: 4.0,  ),  ) : const SizedBox.shrink(),  ),  ),  const SizedBox(height: 27),  Padding(  padding: EdgeInsets.symmetric(horizontal: 8.0),  child: Row(  mainAxisAlignment: MainAxisAlignment.spaceBetween,  children: [  Text(  S.of(context).popularCars,  style: TextStyle(  fontSize: 20,  fontWeight: FontWeight.bold,  ),  ),  Text(  S.of(context).seeAll,  style: TextStyle(  fontSize: 20,  color: Color(0xFF192252),  fontFamily: "Urbanist",  ),  ),  ],  ),  ),  Padding(  padding: const EdgeInsets.symmetric(vertical: 22),  child: SizedBox(  height: 220,  child: ListView.builder(  scrollDirection: Axis.horizontal,  itemCount: popularCars.length,  shrinkWrap: true,  physics: const ClampingScrollPhysics(),  itemBuilder: (context, index) {  return InkWell(  onTap: () {  Navigator.pushNamed(  context,  '/detailPage',  arguments: popularCars[index],  );  },  child: carCard(  context: context,  imageUrl: popularCars[index].imageUrl[0],  title: popularCars[index].name,  price: popularCars[index].rentalPricePerDay,  rating: popularCars[index].rating.toString(),  numOfReviews: '(${popularCars[index].reviewCount})',  isHorizontalScroll: true,  ),  );  },  ),  ),  ),  Padding(  padding: EdgeInsets.fromLTRB(8, 0, 0, 22),  child: Text(  S.of(context).allCars,  style: TextStyle(  fontSize: 20,  fontWeight: FontWeight.bold,  color: Color(0xFF192252),  ),  ),  ),  LayoutBuilder(  builder: (context, constraints) {  double screenWidth = constraints.maxWidth;  int crossAxisCount = (screenWidth ~/ 180).clamp(2, double.infinity).toInt();    return GridView.builder(  padding: EdgeInsets.fromLTRB(0, 0, 0, 50),  physics: const NeverScrollableScrollPhysics(),  shrinkWrap: true,  gridDelegate: SliverGridDelegateWithFixedCrossAxisCount(  crossAxisCount: crossAxisCount,  mainAxisSpacing: 22,  crossAxisSpacing: 8,  childAspectRatio: 180 / 220,  ),  itemCount: allCars.length,  itemBuilder: (context, index) {  return InkWell(  onTap: () {  Navigator.pushNamed(  context,  '/detailPage',  arguments: allCars[index],  );  },  child: carCard(  context: context,  imageUrl: allCars[index].imageUrl[0],  title: allCars[index].name,  price: allCars[index].rentalPricePerDay,  rating: allCars[index].rating.toString(),  numOfReviews: '(${allCars[index].reviewCount}',  isHorizontalScroll: false  ),  );  },  );  },  ),  ],  ),  ),  ),  ),  ),  Positioned(  top: isMobile ? 0 : 59,  left: 0,  right: 0,  child: AppBar(  backgroundColor: Colors.white,  elevation: 0,  title: Column(  crossAxisAlignment: CrossAxisAlignment.start,  children: [  Row(  children: [  Text(  S.of(context).yourLocation,  style: TextStyle(  color: Color(0xFF848FAC),  fontSize: 18,  fontFamily: "Urbanist",  ),  ),  SizedBox(width: 10),  ImageIcon(AssetImage('assets/images/arrow-down.png')),  ],  ),  Text(  S.of(context).moscowRussia,  style: TextStyle(  color: Color(0xFF192252),  fontSize: 20,  fontWeight: FontWeight.bold,  ),  ),  ],  ),  actions: [  IconButton(  icon: const ImageIcon(  AssetImage('assets/images/search.png'),  color: Color(0xFF192252),  ),  onPressed: () {},  ),  IconButton(  icon: const ImageIcon(  AssetImage('assets/images/sms.png'),  color: Color(0xFF192252),  ),  onPressed: () {},  ),  ],  ),  ),  ],  ),  );  }  static Widget carCard({  required BuildContext context,  required String imageUrl,  required String title,  required int price,  required String rating,  required String numOfReviews,  bool isHorizontalScroll = false,  }) {  int numOfReviewsInt = int.parse(numOfReviews.replaceAll(RegExp(r'\D'), ''));  String formattedReviews;  if (numOfReviewsInt >= 100) {  formattedReviews = S.of(context).reviews\_100\_plus;  } else {  formattedReviews = S.of(context).reviews(numOfReviewsInt);  }  return Container(  width: 180,  height: 219,  margin: isHorizontalScroll  ? const EdgeInsets.only(right: 16, bottom: 10, left: 8)  : const EdgeInsets.only(bottom: 8),  decoration: BoxDecoration(  color: Colors.white,  borderRadius: BorderRadius.circular(16),  boxShadow: [  BoxShadow(  color: Colors.black.withOpacity(0.1),  spreadRadius: 1,  blurRadius: 5,  offset: const Offset(0, 3),  ),  ],  ),  child: Column(  crossAxisAlignment: CrossAxisAlignment.start,  children: [  Container(  height: 85,  decoration: BoxDecoration(  borderRadius: BorderRadius.circular(16),  image: DecorationImage(  image: NetworkImage(imageUrl),  fit: BoxFit.cover,  ),  ),  ),  const SizedBox(height: 8),  Padding(  padding: const EdgeInsets.symmetric(horizontal: 8),  child: Text(  title,  style: const TextStyle(  fontSize: 15,  fontWeight: FontWeight.bold,  color: Color(0xFF192252),  ),  overflow: TextOverflow.ellipsis,  maxLines: 1,  ),  ),  const SizedBox(height: 8),  Padding(  padding: const EdgeInsets.symmetric(horizontal: 8),  child: Row(  children: [  const ImageIcon(  AssetImage('assets/images/star.png'),  color: Color(0xFFFFBD16),  ),  const SizedBox(width: 3),  Text(  rating,  style: const TextStyle(  fontSize: 15,  color: Color(0xFFFFBD16),  ),  ),  const SizedBox(width: 10),  Text(  formattedReviews,  style: const TextStyle(  fontSize: 12,  color: Color(0xFF192252),  ),  ),  ],  ),  ),  const Spacer(),  Padding(  padding: const EdgeInsets.only(left: 8, bottom: 8, right: 8),  child: Text(  '${NumberFormat('#,###', 'ru\_RU').format(price)} ₽ / ${S.of(context).day}',  style: const TextStyle(  fontSize: 15,  color: Color(0xFF192252),  ),  ),  ),  ],  ),  );  }  } |

Листинг 9 – detail\_page.dart

|  |
| --- |
| import 'package:flutter/material.dart';  import 'package:smooth\_page\_indicator/smooth\_page\_indicator.dart';  import '../Models/car\_model.dart';  import 'package:intl/intl.dart';  import '../generated/l10n.dart';  class DetailPage extends StatefulWidget {    final Car car;  const DetailPage({super.key, required this.car});  @override  \_DetailPageState createState() => \_DetailPageState();  }  class \_DetailPageState extends State<DetailPage> {  final PageController \_pageController = PageController();  double \_calculateDynamicHeight(BuildContext context) {  double screenWidth = MediaQuery.of(context).size.width;    if (screenWidth < 400) {  return 200;  } else if (screenWidth > 800) {  return 800;  } else {  return 200 + (screenWidth - 400) \* (400 / 400);  }  }  @override  void dispose() {  \_pageController.dispose();  super.dispose();  }  @override  Widget build(BuildContext context) {  return Scaffold(  body: Stack(  children: [  CustomScrollView(  slivers: [  SliverAppBar(  automaticallyImplyLeading: false,  leading: Padding(  padding: const EdgeInsets.only(left: 24),  child: InkWell(  onTap: () {  Navigator.pop(context);  },  borderRadius: BorderRadius.circular(25),  child: Container(  width: 50,  height: 50,  decoration: const BoxDecoration(  shape: BoxShape.circle,  color: Colors.white,  ),  child: const Center(  child: Icon(  Icons.chevron\_left,  size: 30,  color: Color(0xFF192252),  ),  ),  ),  ),  ),  expandedHeight: \_calculateDynamicHeight(context),  floating: false,  pinned: true,  flexibleSpace: LayoutBuilder(  builder: (BuildContext context, BoxConstraints constraints) {  var top = constraints.biggest.height;  return FlexibleSpaceBar(  title: AnimatedOpacity(  opacity: top <= kToolbarHeight + 50 ? 1.0 : 0.0,  duration: const Duration(milliseconds: 300),  ),  background: Stack(  fit: StackFit.expand,  children: [  PageView(  controller: \_pageController,  children: widget.car.imageUrl.map((image) {  return Image.network(  image,  fit: BoxFit.cover,  );  }).toList(),  ),  Align(  alignment: Alignment.bottomCenter,  child: widget.car.imageUrl.length > 1  ? Padding(  padding: const EdgeInsets.only(bottom: 16),  child: SmoothPageIndicator(  controller: \_pageController,  count: widget.car.imageUrl.length,  effect: const ExpandingDotsEffect(  expansionFactor: 4.0,  activeDotColor: Color(0xFF192252),  dotColor: Color(0xFFD9D9D9),  dotHeight: 8,  dotWidth: 8,  spacing: 8.0,  ),  ),  )  : const SizedBox.shrink(),  ),  IgnorePointer(  child: Container(  color: Colors.white.withOpacity(top <= kToolbarHeight + 50 ? 1.0 : 0.0),  ),  ),  ],  ),  );  },  ),  ),    SliverToBoxAdapter(  child: Container(  color: Colors.white,  child: Column(  crossAxisAlignment: CrossAxisAlignment.start,  children: [  const SizedBox(height: 16),  Padding(  padding: const EdgeInsets.symmetric(horizontal: 16),  child: Text(  widget.car.name,  style: const TextStyle(  fontSize: 25,  fontWeight: FontWeight.w900,  color: Color(0xFF192252),  ),  ),  ),  const SizedBox(height: 16),  Padding(  padding: const EdgeInsets.symmetric(horizontal: 16),  child: Row(  children: [  const ImageIcon(  AssetImage('assets/images/star.png'),  size: 30,  color: Color(0xFFFFBD16),  ),  const SizedBox(width: 3),  Text(  '${widget.car.rating}',  style: const TextStyle(  fontSize: 20,  color: Color(0xFFFFBD16),  ),  ),  const SizedBox(width: 10),  Text(  widget.car.reviewCount >= 100  ? S.of(context).reviews\_100\_plus  : S.of(context).reviews(widget.car.reviewCount),  style: const TextStyle(  fontSize: 15,  color: Color(0xFF192252),  ),  ),  ],  ),  ),  Padding(  padding: const EdgeInsets.symmetric(horizontal: 16, vertical: 20),  child: Text(widget.car.description,  style: const TextStyle(  fontWeight: FontWeight.normal,  color: Color(0xFF848FAC),  fontSize: 15,  fontFamily: 'Urbanist'  ),  ),  ),  Padding(  padding: EdgeInsets.only(top: 5, left: 16, right: 16, bottom: 20),  child: Text(  S.of(context).carInfo,  style: TextStyle(  fontSize: 20,  fontWeight: FontWeight.bold,  color: Color(0xFF192252),  ),  )  ),  Padding(  padding: const EdgeInsets.only(bottom: 150, left: 16, right: 16),  child: Column(  children: [  InfoRow(label: S.of(context).engine, value: widget.car.engine),  const SizedBox(height: 20),  InfoRow(label: S.of(context).power, value: S.of(context).horsepower(widget.car.power.toString())),  const SizedBox(height: 20),  InfoRow(label: S.of(context).fuel, value: widget.car.fuel),  const SizedBox(height: 20),  InfoRow(label: S.of(context).color, value: widget.car.color),  const SizedBox(height: 20),  InfoRow(label: 'Drivetrain', value: widget.car.drivetrain),  ],  ),  ),  ],  ),  ),  ),  ],  ),  Positioned(  bottom: 0,  left: 0,  right: 0,  child: Container(  color: Colors.white,  padding: const EdgeInsets.symmetric(horizontal: 16, vertical: 20),  child: Padding(  padding: const EdgeInsets.only(bottom: 16),  child: Row(  mainAxisAlignment: MainAxisAlignment.spaceBetween,  children: [  Column(  crossAxisAlignment: CrossAxisAlignment.start,  children: [  Text(  widget.car.name,  style: const TextStyle(  fontSize: 20,  fontWeight: FontWeight.bold,  color: Color(0xFF192252),  ),  ),  const SizedBox(height: 8),  Text(  '${NumberFormat('#,###', 'ru\_RU').format(widget.car.rentalPricePerDay)} ₽ / ${S.of(context).day}',  style: const TextStyle(  fontSize: 15,  color: Color(0xFF192252),  ),  ),  ],  ),  ElevatedButton(  style: ElevatedButton.styleFrom(  backgroundColor: const Color(0xFF192252),  shape: RoundedRectangleBorder(  borderRadius: BorderRadius.circular(10),  ),  ),  onPressed: () {    },  child: Padding(  padding: EdgeInsets.symmetric(vertical: 20, horizontal: 14),  child: Text(  S.of(context).rentCar,  style: TextStyle(  fontSize: 12,  color: Colors.white  ),  ),  ),  ),  ],  ),  ),  ),  ),  ],  ),  );  }  }  class InfoRow extends StatelessWidget {  final String label;  final String value;  const InfoRow({super.key, required this.label, required this.value});  @override  Widget build(BuildContext context) {  return Row(  children: [  Text(  label,  style: const TextStyle(  fontSize: 18,  color: Color(0xFF848FAC),  fontWeight: FontWeight.normal,  ),  ),  const Spacer(),  Text(  value,  style: const TextStyle(  fontSize: 18,  color: Color(0xFF192252),  fontWeight: FontWeight.bold,  ),  ),  ],  );  }  } |

Листинг 10 – profile\_page.dart

|  |
| --- |
| import 'package:flutter/material.dart';  import 'package:rent\_car\_project/Pages/edit\_user\_page.dart';  import '../Components/custom\_button.dart';  import '../Components/custom\_action\_button.dart';  import 'package:shared\_preferences/shared\_preferences.dart';  import 'package:flutter/foundation.dart';  import 'dart:io';  import '../Services/auth\_service.dart';  import '../generated/l10n.dart';  class ProfilePage extends StatefulWidget {  const ProfilePage({super.key});  @override  \_ProfilePageState createState() => \_ProfilePageState();  }  class \_ProfilePageState extends State<ProfilePage> {  final NetworkService \_networkService = NetworkService();  String? fullName;  String? email;  String? profileImageUrl;  bool isLoading = true;  @override  void initState() {  super.initState();  \_loadUserData();  }  Future<void> \_loadUserData() async {  final prefs = await SharedPreferences.getInstance();  final userId = prefs.getInt('userId');  if (userId == null) {  setState(() {  isLoading = false;  });  return;  }  final response = await \_networkService.getUserData(userId);  if (response['success'] == true) {  setState(() {  fullName = response['userData']['fullName'];  email = response['userData']['email'];  profileImageUrl = response['userData']['photo'] != null && response['userData']['photo'].isNotEmpty  ? 'http://localhost:8080/${response['userData']['photo']}'  : null;  isLoading = false;  });  } else {  setState(() {  isLoading = false;  });  ScaffoldMessenger.of(context).showSnackBar(  SnackBar(content: Text(response['error'] ?? 'Failed to load user data')),  );  }  }  Future<void> \_logout() async {  final response = await \_networkService.logout();  if (response['success'] == true) {  Navigator.pushNamedAndRemoveUntil(  context,  '/login',  (Route<dynamic> route) => false,  );  } else {  ScaffoldMessenger.of(context).showSnackBar(  SnackBar(content: Text(response['error'] ?? 'Logout failed')),  );  }  }  @override  Widget build(BuildContext context) {  bool isMobile = !kIsWeb && (Platform.isAndroid || Platform.isIOS);  Widget content = Column(  children: [  InkWell(  onTap: () async {  await Navigator.push(  context,  MaterialPageRoute(  builder: (context) => EditUserPage(  initialFullName: fullName,  initialEmail: email,  initialProfileImageUrl: profileImageUrl,  )  )  );  \_loadUserData();  },  borderRadius: BorderRadius.circular(20),  child: Container(  padding: const EdgeInsets.all(25.0),  decoration: BoxDecoration(  borderRadius: BorderRadius.circular(20),  gradient: const LinearGradient(  colors: [Color(0xFF1B588C), Color(0xFF848FAC)],  begin: Alignment.centerLeft,  end: Alignment.centerRight,  ),  boxShadow: const [  BoxShadow(  color: Colors.black26,  blurRadius: 10,  offset: Offset(0, 4),  )  ],  ),  child: Row(  children: [  ClipOval(  child: SizedBox.fromSize(  size: const Size.fromRadius(24),  child: profileImageUrl != null && profileImageUrl!.isNotEmpty  ? Image.network(  profileImageUrl!,  fit: BoxFit.cover,  )  : Container(  color: Colors.grey[300],  ),  ),  ),  const SizedBox(width: 13),  Text(  fullName ?? 'Loading...',  style: const TextStyle(  fontSize: 20,  fontWeight: FontWeight.bold,  color: Colors.white,  ),  overflow: TextOverflow.ellipsis,  maxLines: 1,  ),  const Spacer(),  const ImageIcon(  AssetImage('assets/images/edit.png'),  color: Colors.white,  size: 24,  ),  ],  ),  ),  ),  const SizedBox(height: 30),  CustomButton(  iconPath: 'assets/images/lock-circle.png',  label: S.of(context).changePassword,  onTap: () {},  ),  const SizedBox(height: 21),  CustomButton(  iconPath: 'assets/images/empty-wallet.png',  label: S.of(context).billingInformation,  onTap: () {},  ),  const SizedBox(height: 21),  CustomButton(  iconPath: 'assets/images/direct-inbox.png',  label: S.of(context).notifications,  onTap: () {},  ),  isMobile ?  const Spacer() : const SizedBox(height: 130),  Column(  children: [  CustomActionButton(  label: S.of(context).logOut,  isPrimary: true,  onTap: () async {  await \_logout();  Navigator.pushNamedAndRemoveUntil(  context,  '/login',  (Route<dynamic> route) => false,  );  },  ),  const SizedBox(height: 20),  CustomActionButton(  label: S.of(context).deleteAccount,  isPrimary: false,  onTap: () async {  await \_logout();  Navigator.pushNamedAndRemoveUntil(  context,  '/login',  (Route<dynamic> route) => false,  );  },  ),  ],  ),  const SizedBox(height: 120),  ],  );  return Scaffold(  body: Stack(  children: [  Positioned(  top: 120,  left: 0,  right: 0,  bottom: 0,  child: Container(  color: Colors.white,  child: Padding(  padding: const EdgeInsets.symmetric(horizontal: 16.0),  child: isMobile  ? content  : SingleChildScrollView(  child: content,  ),  ),  ),  ),  Positioned(  top: isMobile ? 0 : 59,  left: 0,  right: 0,  child: AppBar(  backgroundColor: Colors.white,  elevation: 0,  title: Text(  S.of(context).profile,  style: TextStyle(  color: Color(0xFF192252),  fontSize: 25,  fontWeight: FontWeight.bold,  ),  ),  centerTitle: false,  ),  ),  ],  ),  );  }  } |

Листинг 11 – edit\_user\_page.dart

|  |
| --- |
| import 'dart:io';  import 'dart:typed\_data';  import 'package:flutter/material.dart';  import 'package:image\_picker/image\_picker.dart';  import 'package:file\_picker/file\_picker.dart';  import '../Components/text\_field.dart';  import '../Components/custom\_action\_button.dart';  import 'package:flutter/foundation.dart';  import '../Services/auth\_service.dart';  import '../generated/l10n.dart';  class EditUserPage extends StatefulWidget {  final String? initialFullName;  final String? initialEmail;  final String? initialProfileImageUrl;  const EditUserPage({  Key? key,  this.initialFullName,  this.initialEmail,  this.initialProfileImageUrl,  }) : super(key: key);  @override  \_EditUserPageState createState() => \_EditUserPageState();  }  class \_EditUserPageState extends State<EditUserPage> {  late TextEditingController \_nameController;  late TextEditingController \_emailController;  File? \_image;  Uint8List? \_webImage;  final ImagePicker \_picker = ImagePicker();  final NetworkService \_networkService = NetworkService();  @override  void initState() {  super.initState();  \_nameController = TextEditingController(text: widget.initialFullName);  \_emailController = TextEditingController(text: widget.initialEmail);  }  Future<void> \_pickImage() async {  if (kIsWeb) {  final result = await FilePicker.platform.pickFiles(  type: FileType.image,  );  if (result != null) {  setState(() {  \_webImage = result.files.first.bytes;  \_image = null;  });  }  } else {  final pickedFile = await \_picker.pickImage(source: ImageSource.gallery);  if (pickedFile != null) {  setState(() {  \_image = File(pickedFile.path);  \_webImage = null;  });  }  }  }  Future<void> \_updateUserInfo() async {  String? fullName = \_nameController.text;  String? email = \_emailController.text;  final response = await \_networkService.updateUser(  fullName: fullName,  email: email,  photo: kIsWeb ? \_webImage : \_image,  );  if (response['success']) {  ScaffoldMessenger.of(context).showSnackBar(  SnackBar(content: Text(S.of(context).userUpdatedSuccessfully)),  );  } else {  ScaffoldMessenger.of(context).showSnackBar(  SnackBar(content: Text('Error: ${response['error']}')),  );  }  }  @override  void dispose() {  \_nameController.dispose();  \_emailController.dispose();  super.dispose();  }  @override  Widget build(BuildContext context) {  bool isMobile = !kIsWeb && (Platform.isAndroid || Platform.isIOS);  var content = Padding(  padding: const EdgeInsets.symmetric(vertical: 8, horizontal: 24),  child: Column(  children: [  Align(  alignment: Alignment.centerLeft,  child: Text(  S.of(context).updateUserInfo,  style: TextStyle(  fontWeight: FontWeight.bold,  fontSize: 25,  color: Color(0xFF192252),  ),  ),  ),  const SizedBox(height: 20),  GestureDetector(  onTap: \_pickImage,  child: Container(  width: 130,  height: 130,  decoration: BoxDecoration(  borderRadius: BorderRadius.circular(24),  color: (\_image == null && \_webImage == null && widget.initialProfileImageUrl == null)  ? Colors.grey[300]  : null,  image: \_image != null  ? DecorationImage(  image: FileImage(\_image!),  fit: BoxFit.cover,  )  : \_webImage != null  ? DecorationImage(  image: MemoryImage(\_webImage!),  fit: BoxFit.cover,  )  : widget.initialProfileImageUrl != null  ? DecorationImage(  image: NetworkImage(widget.initialProfileImageUrl!),  fit: BoxFit.cover,  )  : null,  ),  child: (\_image == null && \_webImage == null)  ? Center(  child: Icon(  Icons.camera\_alt,  color: Colors.white.withOpacity(0.7),  size: 50,  ),  )  : null,  ),  ),  const SizedBox(height: 35),  CustomTextField(  controller: \_nameController,  labelText: S.of(context).fullName,  initialText: widget.initialFullName ?? S.of(context).enterYourName,  ),  const SizedBox(height: 25),  CustomTextField(  controller: \_emailController,  labelText: S.of(context).emailAdress,  initialText: widget.initialEmail ?? S.of(context).enterYourEmail,  ),  isMobile ?  const Spacer() : SizedBox(height: 220),  CustomActionButton(  label: S.of(context).updateInfo,  isPrimary: true,  onTap: \_updateUserInfo,  ),  const SizedBox(height: 55),  ],  ),  );  return Scaffold(  appBar: AppBar(  backgroundColor: Colors.white,  elevation: 0,  automaticallyImplyLeading: false,  title: Row(  crossAxisAlignment: CrossAxisAlignment.start,  children: [  SizedBox(  width: 50,  height: 50,  child: IconButton(  icon: Image.asset('assets/images/backIcon.png'),  onPressed: () {  Navigator.pop(context);  },  ),  ),  ],  ),  ),  backgroundColor: Colors.white,  body: isMobile  ? content  : SingleChildScrollView(  child: content  ),  );  }  } |

Листинг 12 – tab\_bar.dart

|  |
| --- |
| import 'package:flutter/material.dart';  import 'home\_page.dart';  import 'profile\_page.dart';  import 'dart:io' show Platform; // Для проверки платформы  import 'package:flutter/foundation.dart' show kIsWeb;  class CustomBottomNavigationBar extends StatefulWidget {  const CustomBottomNavigationBar({super.key});  @override  \_CustomBottomNavigationBarState createState() => \_CustomBottomNavigationBarState();  }  class \_CustomBottomNavigationBarState extends State<CustomBottomNavigationBar> {  int \_selectedIndex = 0;  static final List<Widget> \_widgetOptions = <Widget>[  const HomePage(),  const Text('Car Rent Tab Content'),  const Text('Chat Tab Content'),  const ProfilePage()  ];  void \_onItemTapped(int index) {  setState(() {  \_selectedIndex = index;  });  }  @override  Widget build(BuildContext context) {  double screenHeight = MediaQuery.of(context).size.height;  double bottomPosition = screenHeight > 737 ? -20 : -10;  bool isMobile = !kIsWeb && (Platform.isAndroid || Platform.isIOS);  return Scaffold(  body: Stack(  children: [  Center(  child: \_widgetOptions.elementAt(\_selectedIndex),  ),  if (isMobile)  Positioned(  left: 0,  right: 0,  bottom: bottomPosition,  child: \_buildBottomNavigationBar(),  )  else  Positioned(  left: 0,  right: 0,  top: -10,  child: \_buildBottomNavigationBar(),  ),  ],  ),  );  }  Widget \_buildBottomNavigationBar() {  return Container(  padding: const EdgeInsets.only(top: 10, bottom: 10),  child: BottomNavigationBar(  type: BottomNavigationBarType.fixed,  currentIndex: \_selectedIndex,  selectedItemColor: const Color(0xFF192252),  unselectedItemColor: const Color(0xFF848FAC),  elevation: 0,  onTap: \_onItemTapped,  backgroundColor: const Color.fromARGB(255, 242, 242, 242),  showSelectedLabels: false,  showUnselectedLabels: false,  items: const <BottomNavigationBarItem>[  BottomNavigationBarItem(  icon: ImageIcon(AssetImage('assets/images/home.png')),  label: '',  ),  BottomNavigationBarItem(  icon: ImageIcon(AssetImage('assets/images/carRent.png')),  label: '',  ),  BottomNavigationBarItem(  icon: ImageIcon(AssetImage('assets/images/chat.png')),  label: '',  ),  BottomNavigationBarItem(  icon: ImageIcon(AssetImage('assets/images/profile.png')),  label: '',  ),  ],  ),  );  }  } |

Листинг 13 – auth\_service.dart

|  |
| --- |
| import 'dart:io';  import 'dart:typed\_data';  import 'package:shared\_preferences/shared\_preferences.dart';  import 'package:http/http.dart' as http;  import 'dart:convert';  import 'package:mime/mime.dart';  import 'package:http\_parser/http\_parser.dart';  import 'package:intl/intl.dart';  import 'package:flutter\_secure\_storage/flutter\_secure\_storage.dart';  class NetworkService {  final String baseUrl = 'http://localhost:8080';  final FlutterSecureStorage \_secureStorage = const FlutterSecureStorage();  String? \_cachedSessionId;  Future<String> \_getLanguage() async {  final prefs = await SharedPreferences.getInstance();  final savedLanguage = prefs.getString('language');  if (savedLanguage != null) {  return savedLanguage;  }  print(Intl.getCurrentLocale());  return Intl.getCurrentLocale();  }  Future<String?> \_getSessionId() async {  if (\_cachedSessionId == null) {  final prefs = await SharedPreferences.getInstance();  \_cachedSessionId = prefs.getString('sessionId');  }  return \_cachedSessionId;  }  Future<Map<String, dynamic>> autoLogin() async {  final prefs = await SharedPreferences.getInstance();  final email = prefs.getString('email');  final password = await \_secureStorage.read(key: 'password');  if (email != null && password != null) {  return await login(email, password);  } else {  return {'success': false, 'error': 'No saved credentials'};  }  }  Future<Map<String, dynamic>> login(String email, String password) async {  final url = Uri.parse('$baseUrl/login');  try {  final response = await http.post(  url,  headers: {'Content-Type': 'application/x-www-form-urlencoded'},  body: {'email': email, 'password': password},  );  if (response.statusCode == 200) {  final responseData = jsonDecode(response.body);  final sessionId = responseData['sessionId'];  final userId = responseData['userId'];  if (sessionId != null) {  await \_saveSessionData(sessionId, email, password, userId);  }  return {  'success': true,  'message': responseData['message'],  'userId': responseData['userId'],  'fullName': responseData['fullName'],  'sessionId': sessionId,  };  } else {  return \_handleErrorResponse(response);  }  } catch (e) {  return {'success': false, 'error': 'Something went wrong. Please try again later.'};  }  }  Future<Map<String, dynamic>> getUserData(int userId) async {  final sessionId = await \_getSessionId();  if (sessionId == null) {  return {'success': false, 'error': 'No session ID found'};  }  final url = Uri.parse('$baseUrl/user?userId=$userId');  try {  final response = await http.get(  url,  headers: {  'Content-Type': 'application/json',  'Authorization': 'Bearer $sessionId',  },  );  if (response.statusCode == 200) {  final responseData = jsonDecode(response.body);  return {  'success': true,  'userData': responseData,  };  } else {  return \_handleErrorResponse(response);  }  } catch (e) {  return {'success': false, 'error': 'Failed to retrieve user data. Please try again later.'};  }  }  Future<Map<String, dynamic>> updateUser({  String? fullName,  String? email,  dynamic photo,  }) async {  final sessionId = await \_getSessionId();  if (sessionId == null) {  return {'success': false, 'error': 'No session ID found'};  }  final url = Uri.parse('$baseUrl/updateUser');  final request = http.MultipartRequest('POST', url);  request.headers['Authorization'] = 'Bearer $sessionId';  if (fullName != null) {  request.fields['fullName'] = fullName;  }  if (email != null) {  request.fields['email'] = email;  }  if (photo != null) {  if (photo is File) {  final mimeType = lookupMimeType(photo.path);  final mimeTypeData = mimeType != null ? mimeType.split('/') : ['image', 'jpeg'];  request.files.add(  await http.MultipartFile.fromPath(  'photo',  photo.path,  contentType: MediaType(mimeTypeData[0], mimeTypeData[1]),  ),  );  } else if (photo is Uint8List) {  final mimeTypeData = ['image', 'jpeg'];  request.files.add(  http.MultipartFile.fromBytes(  'photo',  photo,  contentType: MediaType(mimeTypeData[0], mimeTypeData[1]),  filename: 'uploaded\_image.jpg',  ),  );  }  }  try {  final response = await request.send();  final responseBody = await response.stream.bytesToString();  if (response.statusCode == 200) {  if (email != null) {  final prefs = await SharedPreferences.getInstance();  await prefs.setString('email', email);  }    return {'success': true, 'message': jsonDecode(responseBody)['message']};  } else {  return \_handleErrorResponse(http.Response(responseBody, response.statusCode));  }  } catch (e) {  return {'success': false, 'error': 'Something went wrong. Please try again later.'};  }  }  Future<Map<String, dynamic>> register(String fullName, String email, String password, String confirmPassword) async {  final url = Uri.parse('$baseUrl/register');  try {  final response = await http.post(  url,  headers: {'Content-Type': 'application/x-www-form-urlencoded'},  body: {  'fullName': fullName,  'email': email,  'password': password,  'confirmPassword': confirmPassword,  },  );  if (response.statusCode == 200) {  final responseData = jsonDecode(response.body);  final sessionId = responseData['sessionId'];  final userId = responseData['userId'];  if (sessionId != null && userId != null) {  await \_saveSessionData(sessionId, email, password, userId);  }  return {  'success': true,  'message': responseData['message'],  'userId': userId,  'sessionId': sessionId,  };  } else {  return \_handleErrorResponse(response);  }  } catch (e) {  return {'success': false, 'error': 'Something went wrong. Please try again later.'};  }  }  Future<List<Map<String, dynamic>>> getPopularCars() async {  return \_getWithSession('$baseUrl/cars/popular');  }  Future<List<Map<String, dynamic>>> getAllCars() async {  return \_getWithSession('$baseUrl/cars');  }  Future<List<Map<String, dynamic>>> getPromotions() async {  return \_getWithSession('$baseUrl/promotions');  }  Future<List<Map<String, dynamic>>> \_getWithSession(String url) async {  final sessionId = await \_getSessionId();  final language = await \_getLanguage();  try {  final response = await http.get(  Uri.parse(url),  headers: {  'Content-Type': 'application/json',  if (sessionId != null) 'Authorization': 'Bearer $sessionId',  'Accept-Language': language,  },  );  if (response.statusCode == 200) {  return (jsonDecode(response.body) as List).cast<Map<String, dynamic>>();  } else {  return [];  }  } catch (e) {  return [];  }  }  Future<Map<String, dynamic>> logout() async {  final url = Uri.parse('$baseUrl/logout');  final sessionId = await \_getSessionId();  if (sessionId == null) {  return {'success': false, 'error': 'No session found to logout'};  }  try {  final response = await http.post(  url,  headers: {  'Content-Type': 'application/json',  'Authorization': 'Bearer $sessionId',  },  );  if (response.statusCode == 200) {  await \_clearSessionData();  return {  'success': true,  'message': jsonDecode(response.body)['message'],  };  } else {  return \_handleErrorResponse(response);  }  } catch (e) {  return {'success': false, 'error': 'Something went wrong. Please try again later.'};  }  }  Future<void> \_saveSessionData(String sessionId, String email, String password, int userId) async {  final prefs = await SharedPreferences.getInstance();  \_cachedSessionId = sessionId;    await prefs.setString('sessionId', sessionId);  await prefs.setString('email', email);  await prefs.setInt('userId', userId);  // Сохраняем пароль в безопасное хранилище  await \_secureStorage.write(key: 'password', value: password);  }  Future<void> \_clearSessionData() async {  final prefs = await SharedPreferences.getInstance();  await prefs.remove('sessionId');  await prefs.remove('email');  await prefs.remove('userId');  \_cachedSessionId = null;  // Удаляем пароль из безопасного хранилища  await \_secureStorage.delete(key: 'password');  }    Map<String, dynamic> \_handleErrorResponse(http.Response response) {  final errorData = jsonDecode(response.body);  return {'success': false, 'error': errorData['error'] ?? 'Request failed'};  }  } |

Листинг 14 – custom\_action\_button.dart

|  |
| --- |
| import 'package:flutter/material.dart';  class CustomActionButton extends StatelessWidget {  final String label;  final bool isPrimary;  final VoidCallback onTap;  const CustomActionButton({  super.key,  required this.label,  required this.isPrimary,  required this.onTap,  });  @override  Widget build(BuildContext context) {  return InkWell(  onTap: onTap,  borderRadius: BorderRadius.circular(15),  child: Container(  padding: const EdgeInsets.symmetric(vertical: 16, horizontal: 16),  decoration: BoxDecoration(  color: isPrimary ? const Color(0xFF192252) : Colors.white,  borderRadius: BorderRadius.circular(15),  boxShadow: [  BoxShadow(  color: Colors.grey.withOpacity(0.2),  spreadRadius: 2,  blurRadius: 5,  offset: const Offset(0, 3),  ),  ],  ),  child: Center(  child: Text(  label,  style: TextStyle(  fontSize: 20,  fontWeight: FontWeight.bold,  color: isPrimary ? Colors.white : const Color(0xFFB7264F),  ),  ),  ),  ),  );  }  } |

Листинг 15 – custom\_button.dart

|  |
| --- |
| import 'package:flutter/material.dart';  class CustomButton extends StatelessWidget {  final String iconPath;  final String label;  final VoidCallback onTap;  const CustomButton({  super.key,  required this.iconPath,  required this.label,  required this.onTap,  });  @override  Widget build(BuildContext context) {  return InkWell(  onTap: onTap,  borderRadius: BorderRadius.circular(15),  child: Container(  padding: const EdgeInsets.symmetric(vertical: 18, horizontal: 16),  decoration: BoxDecoration(  color: Colors.white,  borderRadius: BorderRadius.circular(10),  boxShadow: [  BoxShadow(  color: Colors.grey.withOpacity(0.2),  spreadRadius: 2,  blurRadius: 5,  offset: const Offset(0, 3),  ),  ],  ),  child: Row(  children: [  Image.asset(  iconPath,  width: 24,  height: 24,  fit: BoxFit.cover,  ),  const SizedBox(width: 13),  Text(  label,  style: const TextStyle(  fontSize: 20,  fontWeight: FontWeight.bold,  color: Color(0xFF192252),  ),  ),  ],  ),  ),  );  }  } |

Листинг 16 – text\_field.dart

|  |
| --- |
| import 'package:flutter/material.dart';  class CustomTextField extends StatelessWidget {  final TextEditingController controller;  final String labelText;  final bool isObscure;  final String? initialText;  const CustomTextField({  Key? key,  required this.controller,  required this.labelText,  this.isObscure = false,  this.initialText,  }) : super(key: key);  @override  Widget build(BuildContext context) {  if (initialText != null && initialText!.isNotEmpty) {  controller.text = initialText!;  }  return TextField(  controller: controller,  obscureText: isObscure,  style: const TextStyle(  color: Color(0xFF192252),  fontWeight: FontWeight.w400  ),  decoration: InputDecoration(  labelText: labelText,  labelStyle: TextStyle(  color: const Color(0xFF424F7B).withOpacity(0.7),  fontSize: 16,  ),  enabledBorder: OutlineInputBorder(  borderSide: BorderSide(  color: const Color(0xFF424F7B).withOpacity(0.7),  width: 2.0,  ),  borderRadius: BorderRadius.circular(20.0),  ),  focusedBorder: OutlineInputBorder(  borderSide: BorderSide(  color: const Color(0xFF424F7B).withOpacity(0.7),  width: 2.0,  ),  borderRadius: BorderRadius.circular(20.0),  ),  ),  );  }  } |

Листинг 17 – messages\_all.dart

|  |
| --- |
| // DO NOT EDIT. This is code generated via package:intl/generate\_localized.dart  // This is a library that looks up messages for specific locales by  // delegating to the appropriate library.  // Ignore issues from commonly used lints in this file.  // ignore\_for\_file:implementation\_imports, file\_names, unnecessary\_new  // ignore\_for\_file:unnecessary\_brace\_in\_string\_interps, directives\_ordering  // ignore\_for\_file:argument\_type\_not\_assignable, invalid\_assignment  // ignore\_for\_file:prefer\_single\_quotes, prefer\_generic\_function\_type\_aliases  // ignore\_for\_file:comment\_references  import 'dart:async';  import 'package:flutter/foundation.dart';  import 'package:intl/intl.dart';  import 'package:intl/message\_lookup\_by\_library.dart';  import 'package:intl/src/intl\_helpers.dart';  import 'messages\_en.dart' as messages\_en;  import 'messages\_ru.dart' as messages\_ru;  typedef Future<dynamic> LibraryLoader();  Map<String, LibraryLoader> \_deferredLibraries = {  'en': () => new SynchronousFuture(null),  'ru': () => new SynchronousFuture(null),  };  MessageLookupByLibrary? \_findExact(String localeName) {  switch (localeName) {  case 'en':  return messages\_en.messages;  case 'ru':  return messages\_ru.messages;  default:  return null;  }  }  /// User programs should call this before using [localeName] for messages.  Future<bool> initializeMessages(String localeName) {  var availableLocale = Intl.verifiedLocale(  localeName, (locale) => \_deferredLibraries[locale] != null,  onFailure: (\_) => null);  if (availableLocale == null) {  return new SynchronousFuture(false);  }  var lib = \_deferredLibraries[availableLocale];  lib == null ? new SynchronousFuture(false) : lib();  initializeInternalMessageLookup(() => new CompositeMessageLookup());  messageLookup.addLocale(availableLocale, \_findGeneratedMessagesFor);  return new SynchronousFuture(true);  }  bool \_messagesExistFor(String locale) {  try {  return \_findExact(locale) != null;  } catch (e) {  return false;  }  }  MessageLookupByLibrary? \_findGeneratedMessagesFor(String locale) {  var actualLocale =  Intl.verifiedLocale(locale, \_messagesExistFor, onFailure: (\_) => null);  if (actualLocale == null) return null;  return \_findExact(actualLocale);  } |

Листинг 18 – messages\_en.dart

|  |
| --- |
| // DO NOT EDIT. This is code generated via package:intl/generate\_localized.dart  // This is a library that provides messages for a en locale. All the  // messages from the main program should be duplicated here with the same  // function name.  // Ignore issues from commonly used lints in this file.  // ignore\_for\_file:unnecessary\_brace\_in\_string\_interps, unnecessary\_new  // ignore\_for\_file:prefer\_single\_quotes,comment\_references, directives\_ordering  // ignore\_for\_file:annotate\_overrides,prefer\_generic\_function\_type\_aliases  // ignore\_for\_file:unused\_import, file\_names, avoid\_escaping\_inner\_quotes  // ignore\_for\_file:unnecessary\_string\_interpolations, unnecessary\_string\_escapes  import 'package:intl/intl.dart';  import 'package:intl/message\_lookup\_by\_library.dart';  final messages = new MessageLookup();  typedef String MessageIfAbsent(String messageStr, List<dynamic> args);  class MessageLookup extends MessageLookupByLibrary {  String get localeName => 'en';  static String m0(value) => "${value} hp";  static String m1(count) =>  "(${Intl.plural(count, zero: 'No reviews', one: '1 review', other: '${count} reviews')})";  final messages = \_notInlinedMessages(\_notInlinedMessages);  static Map<String, Function> \_notInlinedMessages(\_) => <String, Function>{  "allCars": MessageLookupByLibrary.simpleMessage("All cars"),  "alreadyHaveAnAccount":  MessageLookupByLibrary.simpleMessage("Already have an account? "),  "billingInformation":  MessageLookupByLibrary.simpleMessage("Billing information"),  "carInfo": MessageLookupByLibrary.simpleMessage("CAR INFO"),  "changePassword":  MessageLookupByLibrary.simpleMessage("Change password"),  "color": MessageLookupByLibrary.simpleMessage("Color"),  "confirmPassword":  MessageLookupByLibrary.simpleMessage("Confirm password"),  "day": MessageLookupByLibrary.simpleMessage("day"),  "deleteAccount": MessageLookupByLibrary.simpleMessage("Delete account"),  "dontHaveAnAccount":  MessageLookupByLibrary.simpleMessage("Don\'t have an account? "),  "drivetrain": MessageLookupByLibrary.simpleMessage("Drivetrain"),  "emailAdress": MessageLookupByLibrary.simpleMessage("Email address"),  "engine": MessageLookupByLibrary.simpleMessage("Engine"),  "enterYourEmail":  MessageLookupByLibrary.simpleMessage("Enter your email"),  "enterYourName":  MessageLookupByLibrary.simpleMessage("Enter your name"),  "forgotPassword":  MessageLookupByLibrary.simpleMessage("Forgot password?"),  "fuel": MessageLookupByLibrary.simpleMessage("Fuel"),  "fullName": MessageLookupByLibrary.simpleMessage("Full name"),  "horsepower": m0,  "logOut": MessageLookupByLibrary.simpleMessage("Log Out"),  "login": MessageLookupByLibrary.simpleMessage("Login"),  "loginFailed": MessageLookupByLibrary.simpleMessage("Login Failed"),  "loginFailedPleaseTryAgain": MessageLookupByLibrary.simpleMessage(  "Login failed. Please try again."),  "moscowRussia": MessageLookupByLibrary.simpleMessage("Moscow, Russia"),  "notifications": MessageLookupByLibrary.simpleMessage("Notifications"),  "orLoginWith": MessageLookupByLibrary.simpleMessage("or login with"),  "password": MessageLookupByLibrary.simpleMessage("Password"),  "popularCars": MessageLookupByLibrary.simpleMessage("Popular cars"),  "power": MessageLookupByLibrary.simpleMessage("Power"),  "profile": MessageLookupByLibrary.simpleMessage("Profile"),  "registerBtn": MessageLookupByLibrary.simpleMessage("Register"),  "registerTitle": MessageLookupByLibrary.simpleMessage("Register"),  "registrationFailed":  MessageLookupByLibrary.simpleMessage("Registration Failed"),  "registrationFailedPleaseTryAgain":  MessageLookupByLibrary.simpleMessage(  "Registration failed. Please try again."),  "rentCar": MessageLookupByLibrary.simpleMessage("Rent Car"),  "reviews": m1,  "reviews\_100\_plus":  MessageLookupByLibrary.simpleMessage("(100+ reviews)"),  "seeAll": MessageLookupByLibrary.simpleMessage("see all"),  "updateInfo": MessageLookupByLibrary.simpleMessage("Update info"),  "updateUserInfo":  MessageLookupByLibrary.simpleMessage("Update user info"),  "userUpdatedSuccessfully":  MessageLookupByLibrary.simpleMessage("User updated successfully"),  "welcomeLogin": MessageLookupByLibrary.simpleMessage(  "Welcome!\nLogin to your account,\nOr create new one"),  "yourLocation": MessageLookupByLibrary.simpleMessage("Your location")  };  } |

Листинг 19 – messages\_ru.dart

|  |
| --- |
| // DO NOT EDIT. This is code generated via package:intl/generate\_localized.dart  // This is a library that provides messages for a ru locale. All the  // messages from the main program should be duplicated here with the same  // function name.  // Ignore issues from commonly used lints in this file.  // ignore\_for\_file:unnecessary\_brace\_in\_string\_interps, unnecessary\_new  // ignore\_for\_file:prefer\_single\_quotes,comment\_references, directives\_ordering  // ignore\_for\_file:annotate\_overrides,prefer\_generic\_function\_type\_aliases  // ignore\_for\_file:unused\_import, file\_names, avoid\_escaping\_inner\_quotes  // ignore\_for\_file:unnecessary\_string\_interpolations, unnecessary\_string\_escapes  import 'package:intl/intl.dart';  import 'package:intl/message\_lookup\_by\_library.dart';  final messages = new MessageLookup();  typedef String MessageIfAbsent(String messageStr, List<dynamic> args);  class MessageLookup extends MessageLookupByLibrary {  String get localeName => 'ru';  static String m0(value) => "${value} л.с.";  static String m1(count) =>  "(${Intl.plural(count, zero: 'Нет отзывов', one: '1 отзыв', few: '${count} отзыва', many: '${count} отзывов', other: '${count} отзывов')})";  final messages = \_notInlinedMessages(\_notInlinedMessages);  static Map<String, Function> \_notInlinedMessages(\_) => <String, Function>{  "allCars": MessageLookupByLibrary.simpleMessage("Все автомобили"),  "alreadyHaveAnAccount":  MessageLookupByLibrary.simpleMessage("Уже есть аккаунт? "),  "billingInformation":  MessageLookupByLibrary.simpleMessage("Платежная информация"),  "carInfo":  MessageLookupByLibrary.simpleMessage("Информация об автомобиле"),  "changePassword":  MessageLookupByLibrary.simpleMessage("Поменять пароль"),  "color": MessageLookupByLibrary.simpleMessage("Цвет"),  "confirmPassword":  MessageLookupByLibrary.simpleMessage("Подтвердите пароль"),  "day": MessageLookupByLibrary.simpleMessage("день"),  "deleteAccount":  MessageLookupByLibrary.simpleMessage("Удалить аккаунт"),  "dontHaveAnAccount":  MessageLookupByLibrary.simpleMessage("Нет аккаунта? "),  "drivetrain": MessageLookupByLibrary.simpleMessage("Привод"),  "emailAdress": MessageLookupByLibrary.simpleMessage("Почта"),  "engine": MessageLookupByLibrary.simpleMessage("Двигатель"),  "enterYourEmail":  MessageLookupByLibrary.simpleMessage("Введите вашу почту"),  "enterYourName":  MessageLookupByLibrary.simpleMessage("Введите ваше ФИО"),  "forgotPassword":  MessageLookupByLibrary.simpleMessage("Забыли пароль?"),  "fuel": MessageLookupByLibrary.simpleMessage("Топливо"),  "fullName": MessageLookupByLibrary.simpleMessage("ФИО"),  "horsepower": m0,  "logOut": MessageLookupByLibrary.simpleMessage("Выйти"),  "login": MessageLookupByLibrary.simpleMessage("Войти"),  "loginFailed":  MessageLookupByLibrary.simpleMessage("Не удалось авторизироваться"),  "loginFailedPleaseTryAgain": MessageLookupByLibrary.simpleMessage(  "Не удалось авторизироваться. Пожалуйста, попробуйте снова."),  "moscowRussia": MessageLookupByLibrary.simpleMessage("Москва, Россия"),  "notifications": MessageLookupByLibrary.simpleMessage("Уведомления"),  "orLoginWith": MessageLookupByLibrary.simpleMessage("Или войти через"),  "password": MessageLookupByLibrary.simpleMessage("Пароль"),  "popularCars":  MessageLookupByLibrary.simpleMessage("Популярные автомобили"),  "power": MessageLookupByLibrary.simpleMessage("Мощность"),  "profile": MessageLookupByLibrary.simpleMessage("Профиль"),  "registerBtn":  MessageLookupByLibrary.simpleMessage("Зарегистрироваться"),  "registerTitle": MessageLookupByLibrary.simpleMessage("Регистрация"),  "registrationFailed": MessageLookupByLibrary.simpleMessage(  "Не удалось зарегистрироваться"),  "registrationFailedPleaseTryAgain":  MessageLookupByLibrary.simpleMessage(  "Не удалось зарегистрироваться. Пожалуйста, попробуйте снова."),  "rentCar": MessageLookupByLibrary.simpleMessage("Заказать"),  "reviews": m1,  "reviews\_100\_plus":  MessageLookupByLibrary.simpleMessage("(100+ отзывов)"),  "seeAll": MessageLookupByLibrary.simpleMessage("Все"),  "updateInfo": MessageLookupByLibrary.simpleMessage("Обновить"),  "updateUserInfo":  MessageLookupByLibrary.simpleMessage("Обновить информацию"),  "userUpdatedSuccessfully":  MessageLookupByLibrary.simpleMessage("Информация обновлена"),  "welcomeLogin": MessageLookupByLibrary.simpleMessage(  "Добро пожаловать!\nАвторизируйтесь,\nИли зарегистрируйтесь"),  "yourLocation": MessageLookupByLibrary.simpleMessage("Местоположение")  };  } |

Листинг 20 – l10n.dart

|  |
| --- |
| // GENERATED CODE - DO NOT MODIFY BY HAND  import 'package:flutter/material.dart';  import 'package:intl/intl.dart';  import 'intl/messages\_all.dart';  // \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  // Generator: Flutter Intl IDE plugin  // Made by Localizely  // \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  // ignore\_for\_file: non\_constant\_identifier\_names, lines\_longer\_than\_80\_chars  // ignore\_for\_file: join\_return\_with\_assignment, prefer\_final\_in\_for\_each  // ignore\_for\_file: avoid\_redundant\_argument\_values, avoid\_escaping\_inner\_quotes  class S {  S();  static S? \_current;  static S get current {  assert(\_current != null,  'No instance of S was loaded. Try to initialize the S delegate before accessing S.current.');  return \_current!;  }  static const AppLocalizationDelegate delegate = AppLocalizationDelegate();  static Future<S> load(Locale locale) {  final name = (locale.countryCode?.isEmpty ?? false)  ? locale.languageCode  : locale.toString();  final localeName = Intl.canonicalizedLocale(name);  return initializeMessages(localeName).then((\_) {  Intl.defaultLocale = localeName;  final instance = S();  S.\_current = instance;  return instance;  });  }  static S of(BuildContext context) {  final instance = S.maybeOf(context);  assert(instance != null,  'No instance of S present in the widget tree. Did you add S.delegate in localizationsDelegates?');  return instance!;  }  static S? maybeOf(BuildContext context) {  return Localizations.of<S>(context, S);  }  /// `Welcome!\nLogin to your account,\nOr create new one`  String get welcomeLogin {  return Intl.message(  'Welcome!\nLogin to your account,\nOr create new one',  name: 'welcomeLogin',  desc: '',  args: [],  );  }  /// `Email address`  String get emailAdress {  return Intl.message(  'Email address',  name: 'emailAdress',  desc: '',  args: [],  );  }  /// `Password`  String get password {  return Intl.message(  'Password',  name: 'password',  desc: '',  args: [],  );  }  /// `Forgot password?`  String get forgotPassword {  return Intl.message(  'Forgot password?',  name: 'forgotPassword',  desc: '',  args: [],  );  }  /// `Login`  String get login {  return Intl.message(  'Login',  name: 'login',  desc: '',  args: [],  );  }  /// `or login with`  String get orLoginWith {  return Intl.message(  'or login with',  name: 'orLoginWith',  desc: '',  args: [],  );  }  /// `Don't have an account? `  String get dontHaveAnAccount {  return Intl.message(  'Don\'t have an account? ',  name: 'dontHaveAnAccount',  desc: '',  args: [],  );  }  /// `Register`  String get registerBtn {  return Intl.message(  'Register',  name: 'registerBtn',  desc: '',  args: [],  );  }  /// `Login Failed`  String get loginFailed {  return Intl.message(  'Login Failed',  name: 'loginFailed',  desc: '',  args: [],  );  }  /// `Login failed. Please try again.`  String get loginFailedPleaseTryAgain {  return Intl.message(  'Login failed. Please try again.',  name: 'loginFailedPleaseTryAgain',  desc: '',  args: [],  );  }  /// `Registration Failed`  String get registrationFailed {  return Intl.message(  'Registration Failed',  name: 'registrationFailed',  desc: '',  args: [],  );  }  /// `Registration failed. Please try again.`  String get registrationFailedPleaseTryAgain {  return Intl.message(  'Registration failed. Please try again.',  name: 'registrationFailedPleaseTryAgain',  desc: '',  args: [],  );  }  /// `Full name`  String get fullName {  return Intl.message(  'Full name',  name: 'fullName',  desc: '',  args: [],  );  }  /// `Confirm password`  String get confirmPassword {  return Intl.message(  'Confirm password',  name: 'confirmPassword',  desc: '',  args: [],  );  }  /// `Already have an account? `  String get alreadyHaveAnAccount {  return Intl.message(  'Already have an account? ',  name: 'alreadyHaveAnAccount',  desc: '',  args: [],  );  }  /// `Register`  String get registerTitle {  return Intl.message(  'Register',  name: 'registerTitle',  desc: '',  args: [],  );  }  /// `see all`  String get seeAll {  return Intl.message(  'see all',  name: 'seeAll',  desc: '',  args: [],  );  }  /// `All cars`  String get allCars {  return Intl.message(  'All cars',  name: 'allCars',  desc: '',  args: [],  );  }  /// `Your location`  String get yourLocation {  return Intl.message(  'Your location',  name: 'yourLocation',  desc: '',  args: [],  );  }  /// `Moscow, Russia`  String get moscowRussia {  return Intl.message(  'Moscow, Russia',  name: 'moscowRussia',  desc: '',  args: [],  );  }  /// `Popular cars`  String get popularCars {  return Intl.message(  'Popular cars',  name: 'popularCars',  desc: '',  args: [],  );  }  /// `({count, plural, =0 {No reviews} =1 {1 review} other {{count} reviews}})`  String reviews(num count) {  return Intl.message(  '(${Intl.plural(count, zero: 'No reviews', one: '1 review', other: '$count reviews')})',  name: 'reviews',  desc: '',  args: [count],  );  }  /// `(100+ reviews)`  String get reviews\_100\_plus {  return Intl.message(  '(100+ reviews)',  name: 'reviews\_100\_plus',  desc: '',  args: [],  );  }  /// `day`  String get day {  return Intl.message(  'day',  name: 'day',  desc: '',  args: [],  );  }  /// `CAR INFO`  String get carInfo {  return Intl.message(  'CAR INFO',  name: 'carInfo',  desc: '',  args: [],  );  }  /// `Engine`  String get engine {  return Intl.message(  'Engine',  name: 'engine',  desc: '',  args: [],  );  }  /// `Power`  String get power {  return Intl.message(  'Power',  name: 'power',  desc: '',  args: [],  );  }  /// `Fuel`  String get fuel {  return Intl.message(  'Fuel',  name: 'fuel',  desc: '',  args: [],  );  }  /// `Color`  String get color {  return Intl.message(  'Color',  name: 'color',  desc: '',  args: [],  );  }  /// `Drivetrain`  String get drivetrain {  return Intl.message(  'Drivetrain',  name: 'drivetrain',  desc: '',  args: [],  );  }  /// `{value} hp`  String horsepower(Object value) {  return Intl.message(  '$value hp',  name: 'horsepower',  desc: '',  args: [value],  );  }  /// `Rent Car`  String get rentCar {  return Intl.message(  'Rent Car',  name: 'rentCar',  desc: '',  args: [],  );  }  /// `Change password`  String get changePassword {  return Intl.message(  'Change password',  name: 'changePassword',  desc: '',  args: [],  );  }  /// `Billing information`  String get billingInformation {  return Intl.message(  'Billing information',  name: 'billingInformation',  desc: '',  args: [],  );  }  /// `Notifications`  String get notifications {  return Intl.message(  'Notifications',  name: 'notifications',  desc: '',  args: [],  );  }  /// `Log Out`  String get logOut {  return Intl.message(  'Log Out',  name: 'logOut',  desc: '',  args: [],  );  }  /// `Delete account`  String get deleteAccount {  return Intl.message(  'Delete account',  name: 'deleteAccount',  desc: '',  args: [],  );  }  /// `Profile`  String get profile {  return Intl.message(  'Profile',  name: 'profile',  desc: '',  args: [],  );  }  /// `User updated successfully`  String get userUpdatedSuccessfully {  return Intl.message(  'User updated successfully',  name: 'userUpdatedSuccessfully',  desc: '',  args: [],  );  }  /// `Update user info`  String get updateUserInfo {  return Intl.message(  'Update user info',  name: 'updateUserInfo',  desc: '',  args: [],  );  }  /// `Enter your name`  String get enterYourName {  return Intl.message(  'Enter your name',  name: 'enterYourName',  desc: '',  args: [],  );  }  /// `Enter your email`  String get enterYourEmail {  return Intl.message(  'Enter your email',  name: 'enterYourEmail',  desc: '',  args: [],  );  }  /// `Update info`  String get updateInfo {  return Intl.message(  'Update info',  name: 'updateInfo',  desc: '',  args: [],  );  }  }  class AppLocalizationDelegate extends LocalizationsDelegate<S> {  const AppLocalizationDelegate();  List<Locale> get supportedLocales {  return const <Locale>[  Locale.fromSubtags(languageCode: 'en'),  Locale.fromSubtags(languageCode: 'ru'),  ];  }  @override  bool isSupported(Locale locale) => \_isSupported(locale);  @override  Future<S> load(Locale locale) => S.load(locale);  @override  bool shouldReload(AppLocalizationDelegate old) => false;  bool \_isSupported(Locale locale) {  for (var supportedLocale in supportedLocales) {  if (supportedLocale.languageCode == locale.languageCode) {  return true;  }  }  return false;  }  } |

Листинг 21 –intl\_en.arb

|  |
| --- |
| {  "welcomeLogin": "Welcome!\nLogin to your account,\nOr create new one",  "emailAdress": "Email address",  "password": "Password",  "forgotPassword": "Forgot password?",  "login": "Login",  "orLoginWith": "or login with",  "dontHaveAnAccount": "Don't have an account? ",  "registerBtn": "Register",  "loginFailed": "Login Failed",  "loginFailedPleaseTryAgain": "Login failed. Please try again.",  "registrationFailed": "Registration Failed",  "registrationFailedPleaseTryAgain": "Registration failed. Please try again.",  "fullName": "Full name",  "confirmPassword": "Confirm password",  "alreadyHaveAnAccount": "Already have an account? ",  "registerTitle": "Register",  "seeAll": "see all",  "allCars": "All cars",  "yourLocation": "Your location",  "moscowRussia": "Moscow, Russia",  "popularCars": "Popular cars",  "reviews": "({count, plural, =0 {No reviews} =1 {1 review} other {{count} reviews}})",  "reviews\_100\_plus": "(100+ reviews)",  "day": "day",  "carInfo": "CAR INFO",  "engine": "Engine",  "power": "Power",  "fuel": "Fuel",  "color": "Color",  "drivetrain": "Drivetrain",  "horsepower": "{value} hp",  "rentCar": "Rent Car",  "changePassword": "Change password",  "billingInformation": "Billing information",  "notifications": "Notifications",  "logOut": "Log Out",  "deleteAccount": "Delete account",  "profile": "Profile",  "userUpdatedSuccessfully": "User updated successfully",  "updateUserInfo": "Update user info",  "enterYourName": "Enter your name",  "enterYourEmail": "Enter your email",  "updateInfo": "Update info"  } |

Листинг 22 – intl\_ru.arb

|  |
| --- |
| {  "welcomeLogin": "Добро пожаловать!\nАвторизируйтесь,\nИли зарегистрируйтесь",  "emailAdress": "Почта",  "password": "Пароль",  "forgotPassword": "Забыли пароль?",  "login": "Войти",  "orLoginWith": "Или войти через",  "dontHaveAnAccount": "Нет аккаунта? ",  "registerBtn": "Зарегистрироваться",  "loginFailed": "Не удалось авторизироваться",  "loginFailedPleaseTryAgain": "Не удалось авторизироваться. Пожалуйста, попробуйте снова.",  "registrationFailed": "Не удалось зарегистрироваться",  "registrationFailedPleaseTryAgain": "Не удалось зарегистрироваться. Пожалуйста, попробуйте снова.",  "fullName": "ФИО",  "confirmPassword": "Подтвердите пароль",  "alreadyHaveAnAccount": "Уже есть аккаунт? ",  "registerTitle": "Регистрация",  "seeAll": "Все",  "allCars": "Все автомобили",  "yourLocation": "Местоположение",  "moscowRussia": "Москва, Россия",  "popularCars": "Популярные автомобили",  "reviews": "({count, plural, =0 {Нет отзывов} =1 {1 отзыв} few {{count} отзыва} many {{count} отзывов} other {{count} отзывов}})",  "reviews\_100\_plus": "(100+ отзывов)",  "day": "день",  "carInfo": "Информация об автомобиле",  "engine": "Двигатель",  "power": "Мощность",  "fuel": "Топливо",  "color": "Цвет",  "drivetrain": "Привод",  "horsepower": "{value} л.с.",  "rentCar": "Заказать",  "changePassword": "Поменять пароль",  "billingInformation": "Платежная информация",  "notifications": "Уведомления",  "logOut": "Выйти",  "deleteAccount": "Удалить аккаунт",  "profile": "Профиль",  "userUpdatedSuccessfully": "Информация обновлена",  "updateUserInfo": "Обновить информацию",  "enterYourName": "Введите ваше ФИО",  "enterYourEmail": "Введите вашу почту",  "updateInfo": "Обновить"  } |

Листинг 23 – car\_model.dart

|  |
| --- |
| class Car {  final String name;  final List<String> imageUrl;  final int rentalPricePerDay;  final double rating;  final int reviewCount;  final String description;  final String engine;  final int power;  final String fuel;  final String color;  final String drivetrain;  Car({  required this.name,  required this.imageUrl,  required this.rentalPricePerDay,  required this.rating,  required this.reviewCount,  required this.engine,  required this.description,  required this.power,  required this.fuel,  required this.color,  required this.drivetrain,  });  } |

Листинг 24 – pubspec.yaml

|  |
| --- |
| name: rent\_car\_project  description: "A new Flutter project."  # The following line prevents the package from being accidentally published to  # pub.dev using `flutter pub publish`. This is preferred for private packages.  publish\_to: 'none' # Remove this line if you wish to publish to pub.dev  # The following defines the version and build number for your application.  # A version number is three numbers separated by dots, like 1.2.43  # followed by an optional build number separated by a +.  # Both the version and the builder number may be overridden in flutter  # build by specifying --build-name and --build-number, respectively.  # In Android, build-name is used as versionName while build-number used as versionCode.  # Read more about Android versioning at https://developer.android.com/studio/publish/versioning  # In iOS, build-name is used as CFBundleShortVersionString while build-number is used as CFBundleVersion.  # Read more about iOS versioning at  # https://developer.apple.com/library/archive/documentation/General/Reference/InfoPlistKeyReference/Articles/CoreFoundationKeys.html  # In Windows, build-name is used as the major, minor, and patch parts  # of the product and file versions while build-number is used as the build suffix.  version: 1.0.0+1  environment:  sdk: ^3.5.2  # Dependencies specify other packages that your package needs in order to work.  # To automatically upgrade your package dependencies to the latest versions  # consider running `flutter pub upgrade --major-versions`. Alternatively,  # dependencies can be manually updated by changing the version numbers below to  # the latest version available on pub.dev. To see which dependencies have newer  # versions available, run `flutter pub outdated`.  dependencies:  flutter:  sdk: flutter  flutter\_localizations:  sdk: flutter  smooth\_page\_indicator: ^1.2.0+3  cupertino\_icons: ^1.0.8  flutter\_launcher\_icons: ^0.14.1  image\_picker: ^1.1.2  http: ^1.2.2  intl: ^0.19.0  shared\_preferences: ^2.0.6  file\_picker: ^6.1.1  flutter\_secure\_storage: ^9.0.0  dev\_dependencies:  flutter\_test:  sdk: flutter  # The "flutter\_lints" package below contains a set of recommended lints to  # encourage good coding practices. The lint set provided by the package is  # activated in the `analysis\_options.yaml` file located at the root of your  # package. See that file for information about deactivating specific lint  # rules and activating additional ones.  flutter\_lints: ^5.0.0  # For information on the generic Dart part of this file, see the  # following page: https://dart.dev/tools/pub/pubspec  # The following section is specific to Flutter packages.  flutter:  # The following line ensures that the Material Icons font is  # included with your application, so that you can use the icons in  # the material Icons class.  uses-material-design: true  # To add assets to your application, add an assets section, like this:  assets:  - assets/images/car.png  - assets/images/google\_icon.png  - assets/images/vk\_icon.png  - assets/images/backIcon.png  - assets/images/home.png  - assets/images/carRent.png  - assets/images/chat.png  - assets/images/profile.png  - assets/images/bmwm5.jpg  - assets/images/ferrari.webp  - assets/images/porsche911.webp  - assets/images/arrow-down.png  - assets/images/sms.png  - assets/images/search.png  - assets/images/star.png  - assets/images/userPhoto.jpg  - assets/images/edit.png  - assets/images/lock-circle.png  - assets/images/direct-inbox.png  - assets/images/empty-wallet.png  # An image asset can refer to one or more resolution-specific "variants", see  # https://flutter.dev/to/resolution-aware-images  # For details regarding adding assets from package dependencies, see  # https://flutter.dev/to/asset-from-package  # To add custom fonts to your application, add a fonts section here,  # in this "flutter" section. Each entry in this list should have a  # "family" key with the font family name, and a "fonts" key with a  # list giving the asset and other descriptors for the font. For  # example:  fonts:  - family: Urbanist  fonts:  - asset: assets/fonts/Urbanist-VariableFont\_wght.ttf  # For details regarding fonts from package dependencies,  # see https://flutter.dev/to/font-from-package  flutter\_icons:  android: true  ios: true  image\_path: "assets/images/appIcon.png" # Укажите путь к вашей иконке  flutter\_intl:  enabled: true |