

ГУАП

КАФЕДРА № 41

ОТЧЕТ  
ЗАЩИЩЕН С ОЦЕНКОЙ  
ПРЕПОДАВАТЕЛЬ

Доцент, к.т.н.

\_\_\_\_\_  
должность, уч. степень, звание

\_\_\_\_\_  
подпись, дата

С. А. Чернышев

\_\_\_\_\_  
инициалы, фамилия

## ОТЧЕТ О ПРАКТИЧЕСКОЙ РАБОТЕ №3

«Работа с сетью»

по курсу: «Методы объектно-ориентированного проектирования»

РАБОТУ ВЫПОЛНИЛ

СТУДЕНТ гр. №

\_\_\_\_\_  
подпись, дата

\_\_\_\_\_  
инициалы, фамилия

Санкт-Петербург 2024

## 1. Цель работы

Реализовать для приложения из второй практической работы серверную часть, описав спецификацию сопряжения с сервером в формате OpenAPI v3, саму серверную часть системы и осуществить ее сопряжение с клиентской частью.

## 2. Спецификация

```
openapi: 3.0.3
info:
  title: Dart Server API
  description: API for user management, cars, and promotions.
  version: 1.0.0
servers:
  - url: http://localhost:8080
paths:
  /register:
    post:
      summary: Register a new user.
      requestBody:
        required: true
        content:
          application/x-www-form-urlencoded:
            schema:
              type: object
              properties:
                fullName:
                  type: string
                  example: John Doe
                email:
                  type: string
                  example: john.doe@example.com
                password:
                  type: string
                  example: secretpassword
                confirmPassword:
                  type: string
                  example: secretpassword
              required:
                - fullName
                - email
                - password
                - confirmPassword
      responses:
        "200":
          description: User registered successfully.
          content:
            application/json:
              schema:
                type: object
                properties:
```

```
      message:
        type: string
      userId:
        type: integer
      sessionId:
        type: string
    "400":
      description: Bad request.
/login:
  post:
    summary: Log in a user.
    requestBody:
      required: true
      content:
        application/x-www-form-urlencoded:
          schema:
            type: object
            properties:
              email:
                type: string
                example: john.doe@example.com
              password:
                type: string
                example: secretpassword
            required:
              - email
              - password
    responses:
      "200":
        description: User logged in successfully.
        content:
          application/json:
            schema:
              type: object
              properties:
                message:
                  type: string
                userId:
                  type: integer
                fullName:
                  type: string
                sessionId:
                  type: string
      "403":
        description: Invalid email or password.
/logout:
  post:
    summary: Log out a user.
    security:
      - bearerAuth: []
    responses:
      "200":
        description: User logged out successfully.
```

```
"400":
  description: Bad request.
/users:
  get:
    summary: Retrieve all users.
    security:
      - bearerAuth: []
    responses:
      "200":
        description: List of all users.
        content:
          application/json:
            schema:
              type: array
              items:
                type: object
                properties:
                  id:
                    type: integer
                  fullName:
                    type: string
                  email:
                    type: string
/user:
  get:
    summary: Retrieve a specific user by ID.
    security:
      - bearerAuth: []
    parameters:
      - name: userId
        in: query
        required: true
        schema:
          type: integer
    responses:
      "200":
        description: User details.
        content:
          application/json:
            schema:
              type: object
              properties:
                id:
                  type: integer
                fullName:
                  type: string
                email:
                  type: string
                photo:
                  type: string
      "404":
        description: User not found.
/updateUser:
```

```
post:
  summary: Update user details.
  security:
    - bearerAuth: []
  requestBody:
    required: true
    content:
      multipart/form-data:
        schema:
          type: object
          properties:
            fullName:
              type: string
            email:
              type: string
            photo:
              type: string
              format: binary
  responses:
    "200":
      description: User updated successfully.
```

```
/cars:
  get:
    summary: Retrieve all cars.
    security:
      - bearerAuth: []
    responses:
      "200":
        description: List of all cars.
        content:
          application/json:
            schema:
              type: array
              items:
                type: object
                properties:
                  id:
                    type: integer
                  photos:
                    type: array
                    items:
                      type: string
                  name:
                    type: string
                  rating:
                    type: number
                  reviewCount:
                    type: integer
                  rentalPricePerDay:
                    type: number
                  isPopular:
                    type: boolean
                  description:
```

```
        type: string
      engineType:
        type: string
      power:
        type: integer
      fuelType:
        type: string
      color:
        type: string
      driveType:
        type: string
/cars/popular:
  get:
    summary: Retrieve popular cars.
    security:
      - bearerAuth: []
    responses:
      "200":
        description: List of popular cars.
        content:
          application/json:
            schema:
              type: array
            items:
              type: object
            properties:
              id:
                type: integer
              photos:
                type: array
                items:
                  type: string
              name:
                type: string
              rating:
                type: number
              reviewCount:
                type: integer
              rentalPricePerDay:
                type: number
              isPopular:
                type: boolean
              description:
                type: string
              engineType:
                type: string
              power:
                type: integer
              fuelType:
                type: string
              color:
                type: string
              driveType:
```

```
        type: string
/addCar:
  post:
    summary: Add a new car.
    security:
      - bearerAuth: []
    requestBody:
      required: true
      content:
        multipart/form-data:
          schema:
            type: object
            properties:
              name:
                type: string
              rating:
                type: number
              reviewCount:
                type: number
              rentalPricePerDay:
                type: number
              isPopular:
                type: boolean
              description:
                type: string
              engineType:
                type: string
              power:
                type: integer
              fuelType:
                type: string
              color:
                type: string
              driveType:
                type: string
              photos:
                type: array
                items:
                  type: string
                  format: binary
    responses:
      "200":
        description: Car added successfully.
/promotions:
  get:
    summary: Retrieve all promotions.
    security:
      - bearerAuth: []
    responses:
      "200":
        description: List of all promotions.
        content:
          application/json:
```

```

    schema:
      type: array
      items:
        type: object
        properties:
          id:
            type: integer
          name:
            type: string
          photo:
            type: string
  /addPromotion:
    post:
      summary: Add a new promotion.
      security:
        - bearerAuth: []
      requestBody:
        required: true
        content:
          multipart/form-data:
            schema:
              type: object
              properties:
                name:
                  type: string
                photo:
                  type: string
                  format: binary
      responses:
        "200":
          description: Promotion added successfully.
  components:
    securitySchemes:
      bearerAuth:
        type: http
        scheme: bearer
        bearerFormat: JWT

```

### 3. Выполнение работы

#### 3.1. Реализация серверной части

Для реализации серверной части приложения создадим Dart приложение с типом Server app. Этапы создания приведены на рисунках 1 – 3.



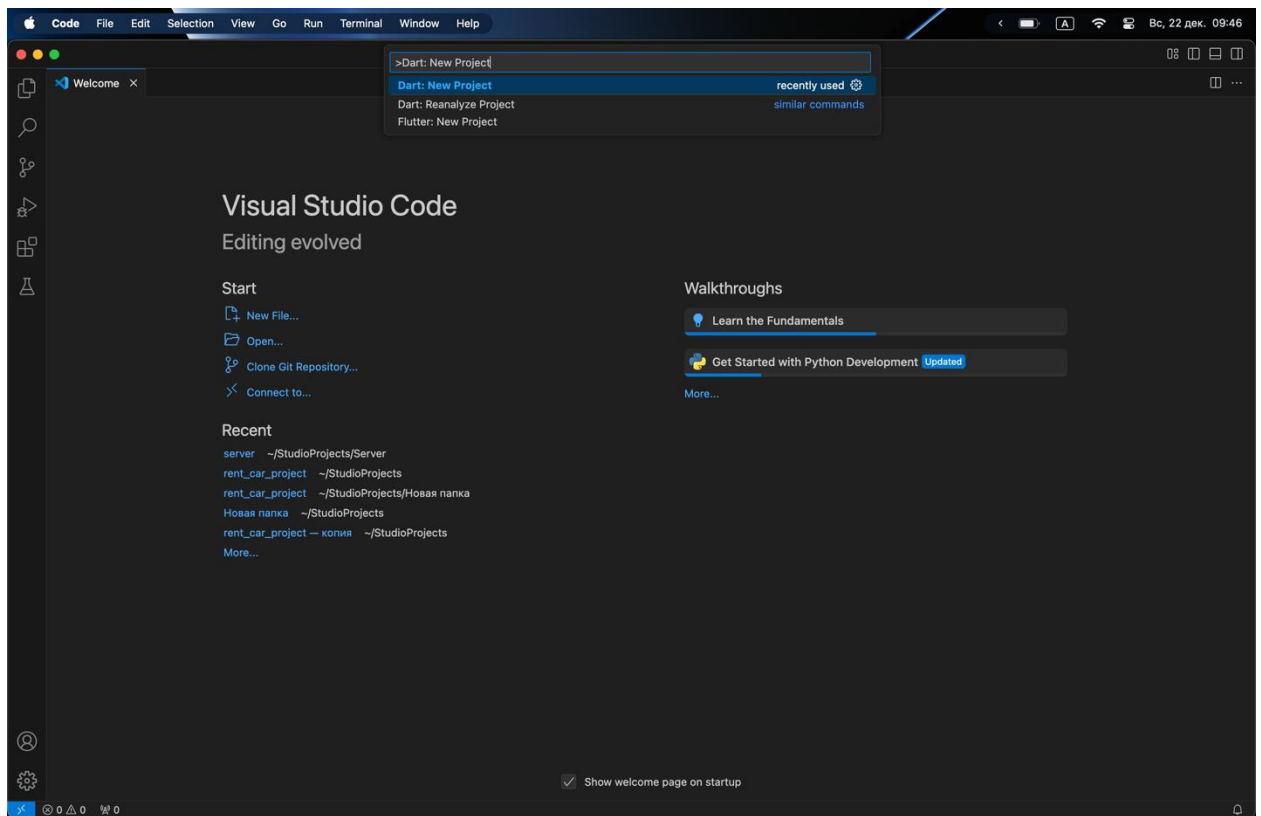


Рисунок 1 – создание Dart приложения (часть1)

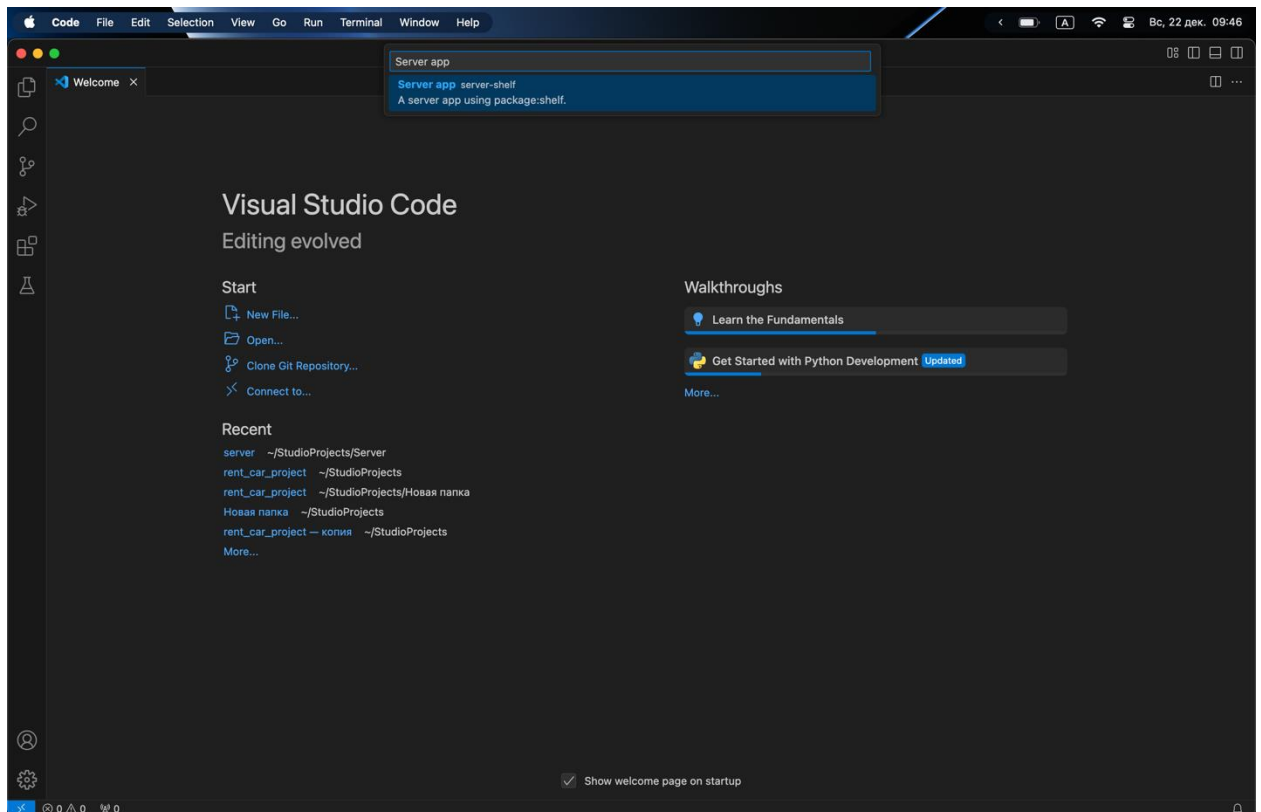


Рисунок 2 – создание Dart приложения (часть2)

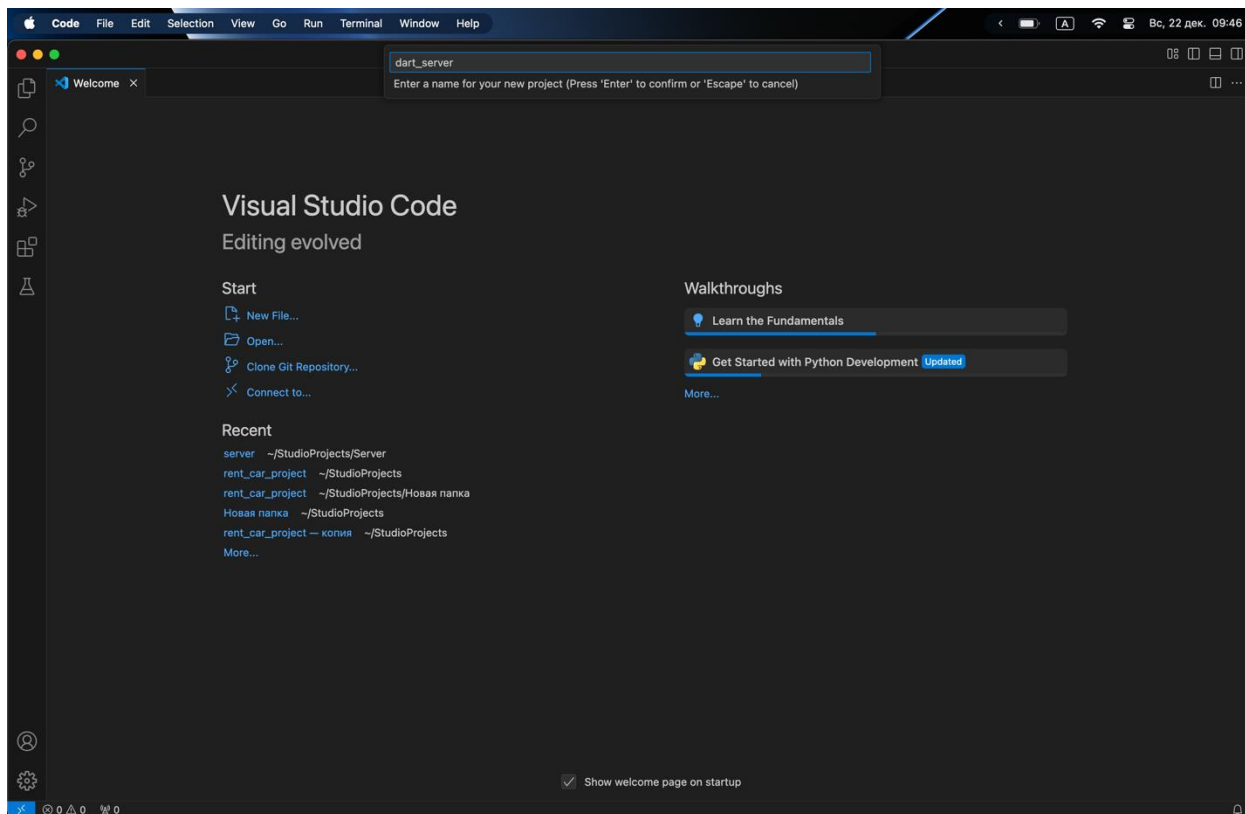


Рисунок 3 – создание Dart приложения (часть3)

После создания проекта перейдем в файл pubspec.yaml и добавим необходимые зависимости – листинг 1:

- Paths – для работы с путями файловой системы;
- Shelf – пакет для создания http-серверов на dart;
- Shelf router – расширение для shelf, добавляющее маршрутизацию;
- Shelf\_static – пакет для обслуживания статических файлов;
- Drift – библиотека для работы с базой данных;

Листинг 1 – pubspec.yaml

```
name: server
description: A server app using the shelf package
version: 1.0.0

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
```

```
dev_dependencies:  
  lints: ^5.0.0  
  test: ^1.24.0  
  build_runner: ^2.1.0  
  drift_dev: ^2.3.0
```

Создадим папку lib, в которой будет находиться код для создания базы данных. В данной папке создадим файл database.dart, импортируем необходимые библиотеки – листинг 2.

Листинг 2 – подключение необходимых пакетов

```
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';
```

Далее произведем объявление таблиц как Dart классы, наследуемые от Table. Всего таблиц будет 4:

- Sessions – хранит информацию о пользовательских сессиях, включая id сессии, id пользователя, время истечения и последнего использования;
- Users – хранит информацию о пользователе, включая его id, полное имя, уникальный email, хэш пароля, фотографию пользователя;
- Promotions – хранит информацию об акциях / новостях компании, включая id, название и фотографию;
- Cars – хранит информацию об автомобилях, включая id автомобиля, фотографии, название, рейтинг, число отзывов, стоимость аренды, популярный автомобиль или нет, описание, тип двигателя, мощность, тип топлива, цвет, привод.

В листингах 3 – 6 представлено объявление таблиц с их полями.

Листинг 3 – Таблица Sessions

```
@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());  
}
```

```
}
```

Листинг 4 – Таблица Users

```
@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();
}
```

Листинг 5 – Таблица Promotions

```
@DataClassName('Promotion')
class Promotions extends Table {
    IntColumn get id => integer().autoIncrement();
    TextColumn get name => text();
    TextColumn get photo => text();
}
```

Листинг 6 – Таблица Cars

```
@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 description => text();
    TextColumn get engineType => text();
    IntColumn get power => integer();
    TextColumn get fuelType => text();
    TextColumn get color => text();
    TextColumn get driveType => text();
}
```

Далее создаем основной класс базы данных AppDatabase, в @DriftDatabase(tables: []) перечисляем таблицы, которые будут включены в базу данных. Добавляем конструктор AppDatabase() : super(\_openConnection()), определяем метод \_openConnection который определяет путь к базе данных и подключает SQLite – листинг 7.

Листинг 7 – Создание основного класса базы данных

```
@DriftDatabase(tables: [Users, Cars, Promotions, Sessions])
class AppDatabase extends _$AppDatabase {
    AppDatabase() : super(_openConnection());
}

LazyDatabase _openConnection() {
```

```

return LazyDatabase() async {
  final dbFolder = Directory.current.path;
  final file = File(p.join(dbFolder, 'app.db'));
  return NativeDatabase(file);
};
}

```

После чего в класс добавляем миграцию базы данных, которая отвечает за создание всех таблиц при первом запуске – листинг 8

Листинг 8 – добавление миграции

```

@override
MigrationStrategy get migration => MigrationStrategy(
  onCreate: (Migrator m) async {
    await m.createAll();
  },
  onUpgrade: (Migrator m, int from, int to) async {
  },
);

```

Далее добавляем методы:

- Регистрации пользователя - получает в качестве параметров полное имя пользователя, email, пароль. После чего создается переменная user, в которой при помощи автоматически созданного файла database.g.dart вызывается UsersCompanion(в который передаются fullname, email, хешируемый пароль), после чего функция registerUser возвращает вставку в таблицу users пользователя. Для хеширования пароля был написан метод hashPassword(password), который принимает пароль, а возвращает hash при помощи sha256.convert() – листинг 9;
- Аутентификации пользователя – получает на вход email и пароль, после чего создается переменная query, которая создает запрос к таблице пользователей и ищет там запись с заданным email, создается переменная user, которая возвращает пользователя, если запись найдена или null – если нет; дальше происходит проверка, что если пользователь найден, введенный пароль хэшируется и сравнивается с хэшем, хранящимся в базе данных и они совпадают, то возвращается объект User;
- Получение всех пользователей – возвращает всех пользователей при

помощи запроса к базе users;

- Получение пользователя по email – принимает email и обращается к базе данных, где происходит поиск пользователя по email, если не найдено – возвращается null;
- Получение пользователя по id – принимает id пользователя и обращается к базе данных, где происходит поиск пользователя по id, если не найдено – возвращается null;
- Обновление данных о пользователе – получает id пользователя, полное имя, email, фото, после чего создается UsersCompanion, в который передаются параметры, после чего происходит обновление строки в таблице БД users, где id совпадает с переданным параметром.
- Получение всех автомобилей – возвращает список всех автомобилей;
- Получение популярных автомобилей – возвращает список автомобилей, у которых поле isPopular установлено в true;
- Добавление автомобиля – получает на вход необходимые параметры, после чего создается CarsCompanion, в параметры которого передаются переданные параметры, после чего выполняется запрос на добавление записи в таблицу cars;
- Получение всех акций / новостей – возвращает список всех акций / новостей;
- Добавление акции / новости – получает на вход необходимые параметры, после чего создается PromotionsCompanion, в параметры которого передаются переданные параметры, после чего выполняется запрос на добавление записи в таблицу promotions;
- Создание сессии – создает новую сессию для пользователя с переданным id в параметр. Для генерации id сессии необходимо написать метод generateSessionId() – листинг 22. Создается переменная expiredAt, в которую устанавливается время завершения сессии – через 15 минут от создания, дальше создается SessionsCompanion, в параметры которого передаются созданные и переданные параметры, после чего

создается запрос на добавление сессии в таблицу sessions, а функция возвращает объект сессии извлеченный из таблицы по ее id;

- Обновление времени использования сессии – обновляет время использования для указанной сессии;
- Удаление сессии – удаляет сессию с указанным sessionId;

В листингах 9 – 25 представлены методы и необходимые для них дополнительные методы.

Листинг 9 – метод хэширования пароля

```
String hashPassword(String password) {  
    return sha256.convert(utf8.encode(password)).toString();  
}
```

Листинг 10 – метод регистрации пользователя

```
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);  
}
```

Листинг 11 – метод авторизации пользователя

```
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;  
}
```

Листинг 12 – метод получения всех пользователей

```
Future<List<User>> getAllUsers() async {  
    return select(users).get();  
}
```

Листинг 13 – метод получения пользователя по email

```
Future<User?> getUserByEmail(String email) async {  
    final query = select(users)..where((tbl) => tbl.email.equals(email));  
    return await query.getSingleOrNull();  
}
```

Листинг 14 – метод получения пользователя по id

```
Future<User?> getUserById(int id) async {  
    final query = select(users)..where((tbl) => tbl.id.equals(id));  
    return await query.getSingleOrNull();  
}
```

```
}
```

Листинг 15 – метод обновления данных пользователя

```
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);
}
```

Листинг 16 – метод получения всех автомобилей

```
Future<List<Car>> getAllCars() async {
  return select(cars).get();
}
```

Листинг 17 – метод получения всех популярных автомобилей

```
Future<List<Car>> getPopularCars() async {
  return (select(cars)..where((tbl) => tbl.isPopular.equals(true))).get();
}
```

Листинг 18 – метод добавления автомобиля

```
Future<int> addCar({
  required String photos,
  required String name,
  required double rating,
  required double reviewCount,
  required double rentalPricePerDay,
  required bool isPopular,
  required String description,
  required String engineType,
  required int power,
  required String fuelType,
  required String color,
  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),
    description: Value(description),
    engineType: Value(engineType),
    power: Value(power),
    fuelType: Value(fuelType),
    color: Value(color),
  );
}
```



```

        driveType: Value(driveType),
    );
    return into(cars).insert(car);
}

```

Листинг 19 – метод получения всех акций / новостей

```

Future<List<Promotion>> getAllPromotions() async {
    return select(promotions).get();
}

```

Листинг 20 – метод добавления акции / новости

```

Future<int> addPromotion({
    required String name,
    required String photo,
}) async {
    final promotion = PromotionsCompanion(
        name: Value(name),
        photo: Value(photo),
    );
    return into(promotions).insert(promotion);
}

```

Листинг 21 – метод создания сессии

```

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();
}

```

Листинг 22 – метод генерации сессии

```

String _generateSessionId(int userId) {
    final bytes = utf8.encode(DateTime.now().toString() + userId.toString());
    return sha256.convert(bytes).toString();
}

```

Листинг 23 – метод получения сессии по ее id

```

Future<Session?> getSessionById(String sessionId) async {
    final query = select(sessions)..where((tbl) => tbl.sessionId.equals(sessionId));
    return await query.getSingleOrNull();
}

```

Листинг 24 – метод обновления времени сессии

```

Future<void> updateSessionLastUsed(String sessionId, DateTime lastUsed) async {
    await (update(sessions)

```

```

        ..where((tbl) => tbl.sessionId.equals(sessionId)))
        .write(SessionsCompanion(lastUsed: Value(lastUsed)));
    }

```

## Листинг 25 – метод удаления сессии

```

Future<void> deleteSession(String sessionId) async {
    await (delete(sessions)..where((tbl) => tbl.sessionId.equals(sessionId))).go();
}

```

## Листинг 26 – Полный код 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 name => 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 description => text();
}

```

```

TextColumn get engineType => text();
IntColumn get power => integer();
TextColumn get fuelType => text();
TextColumn get color => text();
TextColumn get driveType => text();
}

@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<Car>> getAllCars() async {
  return select(cars).get();
}

Future<List<Car>> getPopularCars() async {
  return (select(cars)..where((tbl) => tbl.isPopular.equals(true))).get();
}

Future<int> addCar({
  required String photos,
  required String name,
  required double rating,
  required double reviewCount,
  required double rentalPricePerDay,
  required bool isPopular,
  required String description,
  required String engineType,
  required int power,
  required String fuelType,
  required String color,
  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),
    description: Value(description),
    engineType: Value(engineType),
    power: Value(power),
  );

```

```

        fuelType: Value(fuelType),
        color: Value(color),
        driveType: Value(driveType),
    );
    return into(cars).insert(car);
}

Future<List<Promotion>> getAllPromotions() async {
    return select(promotions).get();
}

Future<int> addPromotion({
    required String name,
    required String photo,
}) async {
    final promotion = PromotionsCompanion(
        name: Value(name),
        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);
  });
}

```

Перейдем в папку bin и откроем файл server.dart.

Импортируем необходимые библиотеки – листинг 27.

Листинг 27 – импорт необходимых библиотек

```

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';

```

Создадим экземпляр класса AppDatabase путем создания переменной db – листинг 28.

Листинг 28 – Создание экземпляра AppDatabase

```

final db = AppDatabase();

```

Добавим заголовки поддержки CORS при помощи Middleware, чтобы клиенты могли взаимодействовать с API – Листинг 29.

Листинг 29 – Добавление заголовков поддержки CORS при помощи Middleware

```

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);
  };
};
}

```

Создадим точку запуска main – листинг 30.

Листинг 30 – точка запуска

```

void main() async {
}

```

В точке запуска создадим роутер – Router() – листинг 31.

Листинг 31 – создание роутера

```

final router = Router();

```

Так как в дальнейшем понадобится обработка статических файлов, то необходимо добавить обслуживание для них. Для этого создадим обработчик статических файлов и добавим обработчик на маршрут – листинг 32

Листинг 32 – обслуживание статических файлов

```

final staticFilesHandler = createStaticHandler('uploads', defaultDocument: 'index.html');

router.mount('/uploads/', staticFilesHandler);

```

Теперь создадим незащищенные маршруты, которые будут доступны пользователю без id сессии:

- Регистрация – читает данные запроса (имя, email, пароль и подтвержденный пароль), после проверяет их корректность и полноту, проверяет существует ли пользователь с такой почтой или нет, если нет, вызывает функцию регистрации пользователя из БД и создает сессию, после чего возвращает ответ с id пользователя и id сессии;
- Авторизация – читает данные запроса (email и пароль), вызывает функцию авторизации из БД, если пользователь найден, то создается новая сессия и в ответе приходят данные пользователя и id сессии;
- Выход – извлекает id сессии из заголовка Authorization, удаляет сессию из базы данных, возвращает сообщение о выходе из системы;

Код для описанных маршрутов представлен в листингах 33 – 36.

Листинг 33 – регистрация

```

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()}));
}
});

```

Листинг 34 – авторизация

```

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'}));
    }
});

```

Листинг 35 – метод извлечения id сессии из заголовка Authorization

```

String? extractSessionId(Request request) {
    final authHeader = request.headers['Authorization'];
    if (authHeader != null && authHeader.startsWith('Bearer ')) {
        return authHeader.substring(7);
    }
    return null;
}

```

Листинг 36 – выход

```

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 ',
    });
});

```

Создадим переменную для защищенного роутера – листинг 37.

Листинг 37 – создание защищенного роутера

```

final protectedRouter = Router();

```

Теперь создадим защищенные маршруты, которые будут доступны пользователю только с id сессии:

- Получение списка пользователей – получает список всех пользователей из БД и возвращает JSON объект с информацией о пользователях (список пользователей);
- Получение данных пользователя – получает id пользователя из

параметров запроса, обращается к БД и ищет пользователя с заданным id, возвращает JSON объект с данными о пользователе;

- Обновление данных пользователя – получает данные из запроса и изображение, проверяет авторизацию пользователя, если все хорошо, то обновляет данные пользователя в БД и возвращает сообщение об успешном обновлении;
- Получение списка автомобилей – обращается к функции получения всех автомобилей в БД, формирует список автомобилей и возвращает его в JSON формате;
- Получение списка популярных автомобилей - обращается к функции получения популярных автомобилей в БД, формирует список автомобилей и возвращает его в JSON формате;
- Добавление автомобиля – использует MimeTypePartTransformer для обработки multipart-данных, все части запроса (текстовые поля и файлы) разделяются и обрабатываются по очереди – файлы загружаются в папку uploads на сервере и их пути сохраняются, текстовые поля - извлекаются значения. Происходит проверка, что все обязательные поля заполнены, если успешно, то автомобиль добавляется в БД и возвращается сообщение об успешном добавлении автомобиля;
- Получение всех акций / новостей – выполняется функция БД для получения всех акций / новостей, после чего формируется список и возвращается в виде JSON объекта;
- Добавление акции / новости – использует MimeTypePartTransformer для обработки multipart-данных, все части запроса (текстовые поля и файлы) разделяются и обрабатываются по очереди – файлы загружаются в папку uploads на сервере и их пути сохраняются, текстовые поля - извлекаются значения. Проверяются обязательные поля, если успешно, то запись добавляется в БД и возвращается ответ об успешном добавлении.

Код для описанных маршрутов представлен в листингах 38 – 45.

Листинг 38 – получение списка пользователей

```
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');
  }
});
```

Листинг 39 – получение данных пользователя

```
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()}));
  }
});
```

Листинг 40 – изменение данных пользователя

```
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()}));
}
});

```

## Листинг 41 – получение списка автомобилей

```
protectedRouter.get('/cars', (Request request) async {
  try {
    final cars = await db.getAllCars();
    final carList = cars.map((car) => {
      'id': car.id,
      'photos': jsonDecode(car.photos),
      'name': car.name,
      'rating': car.rating,
      'reviewCount': car.reviewCount,
      'rentalPricePerDay': car.rentalPricePerDay,
      'isPopular': car.isPopular,
      'description': car.description,
      'engineType': car.engineType,
      'power': car.power,
      'fuelType': car.fuelType,
      'color': car.color,
      'driveType': car.driveType,
    }).toList();

    return Response.ok(jsonEncode(carList), headers: {'Content-Type': 'application/json'});
  } catch (e) {
    return Response.internalServerError(body: 'Failed to retrieve cars: $e');
  }
});
```

## Листинг 42 – получение списка популярных автомобилей

```
protectedRouter.get('/cars/popular', (Request request) async {
  try {
    final cars = await db.getPopularCars();
    final popularCarsList = cars.map((car) => {
      'id': car.id,
      'photos': jsonDecode(car.photos),
      'name': car.name,
      'rating': car.rating,
      'reviewCount': car.reviewCount,
      'rentalPricePerDay': car.rentalPricePerDay,
      'isPopular': car.isPopular,
      'description': car.description,
      'engineType': car.engineType,
      'power': car.power,
      'fuelType': car.fuelType,
      'color': car.color,
      'driveType': car.driveType,
    }).toList();

    return Response.ok(jsonEncode(popularCarsList), headers: {'Content-Type': 'application/json'});
  } catch (e) {
    return Response.internalServerError(body: 'Failed to retrieve popular cars: $e');
  }
});
```

## Листинг 43 – добавление автомобиля

```

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? description;
    String? engineType;
    int? power;
    String? fuelType;
    String? color;
    String? driveType;
    List<String> photoPaths = [];

    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);

            photoPaths.add(filePath);
        } else {
            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="description"')) {
                description = field;
            } else if (contentDisposition.contains('name="engineType"')) {
                engineType = field;
            } else if (contentDisposition.contains('name="power"')) {
                power = int.parse(field);
            }
        }
    }
}

```

```

    } else if (contentDisposition.contains('name="fuelType"')) {
        fuelType = field;
    } else if (contentDisposition.contains('name="color"')) {
        color = field;
    } else if (contentDisposition.contains('name="driveType"')) {
        driveType = field;
    }
}
}

if (name == null || rating == null || reviewCount == null || rentalPricePerDay == null || isPopular == null ||
description == null || engineType == null || power == null || fuelType == null || color == null || driveType == null) {
    return Response.badRequest(body: jsonEncode({'error': 'Missing required fields'}));
}

await db.addCar(
    photos: jsonEncode(photoPaths),
    name: name,
    rating: rating,
    reviewCount: reviewCount,
    rentalPricePerDay: rentalPricePerDay,
    isPopular: isPopular,
    description: description,
    engineType: engineType,
    power: power,
    fuelType: fuelType,
    color: color,
    driveType: driveType,
);

return Response.ok(jsonEncode({'message': 'Car added successfully'}));
});

```

Листинг 44 – получение списка акций / новостей

```

protectedRouter.get('/promotions', (Request request) async {
    try {
        final promotions = await db.getAllPromotions();
        final promotionsList = promotions.map((promo) => {
            'id': promo.id,
            'name': promo.name,
            'photo': promo.photo,
        }).toList();

        return Response.ok(jsonEncode(promotionsList), headers: {'Content-Type': 'application/json'});
    } catch (e) {
        return Response.internalServerError(body: 'Failed to retrieve promotions: $e');
    }
});

```

Листинг 45 – добавление акции / новости

```

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 = MultipartTransformer(boundary);
final parts = await transformer.bind(request.read()).toList();

String? name;
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="name"')) {
      name = field;
    }
  }
}

if (name == null || photoPath == null) {
  return Response.badRequest(body: jsonEncode({'error': 'Missing required fields'}));
}

await db.addPromotion(
  name: name,
  photo: photoPath,
);

return Response.ok(jsonEncode({'message': 'Promotion added successfully'}));
});

```

Добавим проверку авторизации пользователя с использованием сессии при помощи Middleware `sessionChecker()`. Получаем идентификатор сессии из заголовка `Authorization`, проверяем полученный идентификатор, проверяем время бездействия, если меньше 15 минут, то обновляем время последнего использования, после чего передаем управление следующему обработчику – листинг 46.

Листинг 46 – проверка авторизации пользователя с использованием сессии

```

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);
};
};
}

```

Далее создадим обработчик для защищенных маршрутов, которые требуют авторизации при помощи Pipeline(). Каждый запрос проходит проверку авторизации, если проверка пройдена, то запрос передается обработчику маршрутов – листинг 47.

Листинг 47 – обработчик защищенных маршрутов

```

final protectedHandler = Pipeline()
    .addMiddleware(sessionChecker())
    .addHandler(protectedRouter.call);

```

Создадим теперь основной обработчик, объединяющий незащищенные и защищенные маршруты. Создаем Pipeline(), добавляем логирование всех входящих запросов и заголовки для поддержки CORS при помощи Middleware, добавляем роутинг, где router – незащищенные маршруты, а protectedHandler – защищенные, добавляем комбинированный обработчик – все маршруты объединяются в едином Router() – листинг 48.

Листинг 48 – Основной обработчик, объединяющий маршруты

```

final handler = Pipeline()
    .addMiddleware(logRequests())

```

```

    .addMiddleware(corsHeaders())
    .addHandler((Router()
    ..mount('/', router.call)
    ..mount('/', protectedHandler)).call);

```

Добавим запуск сервера и вывод ссылки, по которой он работает – листинг 49.

Листинг 49 – запуск сервера

```

final server = await io.serve(handler, InternetAddress.anyIPv4, 8080);
print('Server running on http://localhost:${server.port}');

```

Листинг 50 – полный код 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';

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 userList = users.map((user) => {
            'id': user.id,
            'fullName': user.fullName,
            'email': user.email,
        }).toList();

        return Response.ok(jsonEncode(userList), 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 = MimeTypeMultipartTransformer(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 {
    try {
        final cars = await db.getAllCars();
        final carList = cars.map((car) => {
            'id': car.id,
            'photos': jsonDecode(car.photos),
            'name': car.name,
            'rating': car.rating,
            'reviewCount': car.reviewCount,
            'rentalPricePerDay': car.rentalPricePerDay,
            'isPopular': car.isPopular,
            'description': car.description,
            'engineType': car.engineType,
            'power': car.power,
        });
    }
});

```

```

        'fuelType': car.fuelType,
        'color': car.color,
        'driveType': car.driveType,
    }).toList();

    return Response.ok(jsonEncode(carList), headers: {'Content-Type': 'application/json'});
} catch (e) {
    return Response.internalServerError(body: 'Failed to retrieve cars: $e');
}
});

protectedRouter.get('/cars/popular', (Request request) async {
    try {
        final cars = await db.getPopularCars();
        final popularCarsList = cars.map((car) => {
            'id': car.id,
            'photos': jsonDecode(car.photos),
            'name': car.name,
            'rating': car.rating,
            'reviewCount': car.reviewCount,
            'rentalPricePerDay': car.rentalPricePerDay,
            'isPopular': car.isPopular,
            'description': car.description,
            'engineType': car.engineType,
            'power': car.power,
            'fuelType': car.fuelType,
            'color': car.color,
            'driveType': car.driveType,
        }).toList();

        return Response.ok(jsonEncode(popularCarsList), 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? description;
    String? engineType;
    int? power;

```

```

String? fuelType;
String? color;
String? driveType;
List<String> photoPaths = [];

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);

    photoPaths.add(filePath);
  } else {
    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="description")) {
      description = field;
    } else if (contentDisposition.contains("name="engineType")) {
      engineType = field;
    } else if (contentDisposition.contains("name="power")) {
      power = int.parse(field);
    } else if (contentDisposition.contains("name="fuelType")) {
      fuelType = field;
    } else if (contentDisposition.contains("name="color")) {
      color = field;
    } else if (contentDisposition.contains("name="driveType")) {
      driveType = field;
    }
  }
}

if (name == null || rating == null || reviewCount == null || rentalPricePerDay == null || isPopular == null ||
description == null || engineType == null || power == null || fuelType == null || color == null || driveType == null) {
  return Response.badRequest(body: jsonEncode({'error': 'Missing required fields'}));
}

await db.addCar(
  photos: jsonEncode(photoPaths),

```



```

        name: name,
        rating: rating,
        reviewCount: reviewCount,
        rentalPricePerDay: rentalPricePerDay,
        isPopular: isPopular,
        description: description,
        engineType: engineType,
        power: power,
        fuelType: fuelType,
        color: color,
        driveType: driveType,
    );

    return Response.ok(jsonEncode({'message': 'Car added successfully'}));
});

protectedRouter.get('/promotions', (Request request) async {
    try {
        final promotions = await db.getAllPromotions();
        final promotionsList = promotions.map((promo) => {
            'id': promo.id,
            'name': promo.name,
            'photo': promo.photo,
        }).toList();

        return Response.ok(jsonEncode(promotionsList), 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? name;
    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="name"')) {
            name = field;
        }
    }
}

if (name == null || photoPath == null) {
    return Response.badRequest(body: jsonEncode({'error': 'Missing required fields'}));
}

await db.addPromotion(
    name: name,
    photo: photoPath,
);

return Response.ok(jsonEncode({'message': 'Promotion added successfully'}));
});

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);

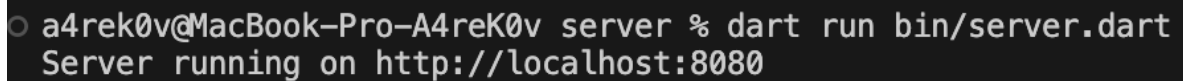
            return handler(request);
        };
    };
}

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}');  
}
```

Запустим сервер при помощи ввода команды `dart run bin/server.dart` в терминале и получим вывод в консоль сообщения о запуске сервера – рисунок 4.



```
a4rek0v@MacBook-Pro-A4reK0v server % dart run bin/server.dart  
Server running on http://localhost:8080
```

Рисунок 4 – запуск сервера в терминале

Теперь при помощи Postman протестируем работу запросов, а также работу защищенных запросов, если не передать id сессии. На рисунках 5 – 10 представлены ответы для части запросов в Postman.

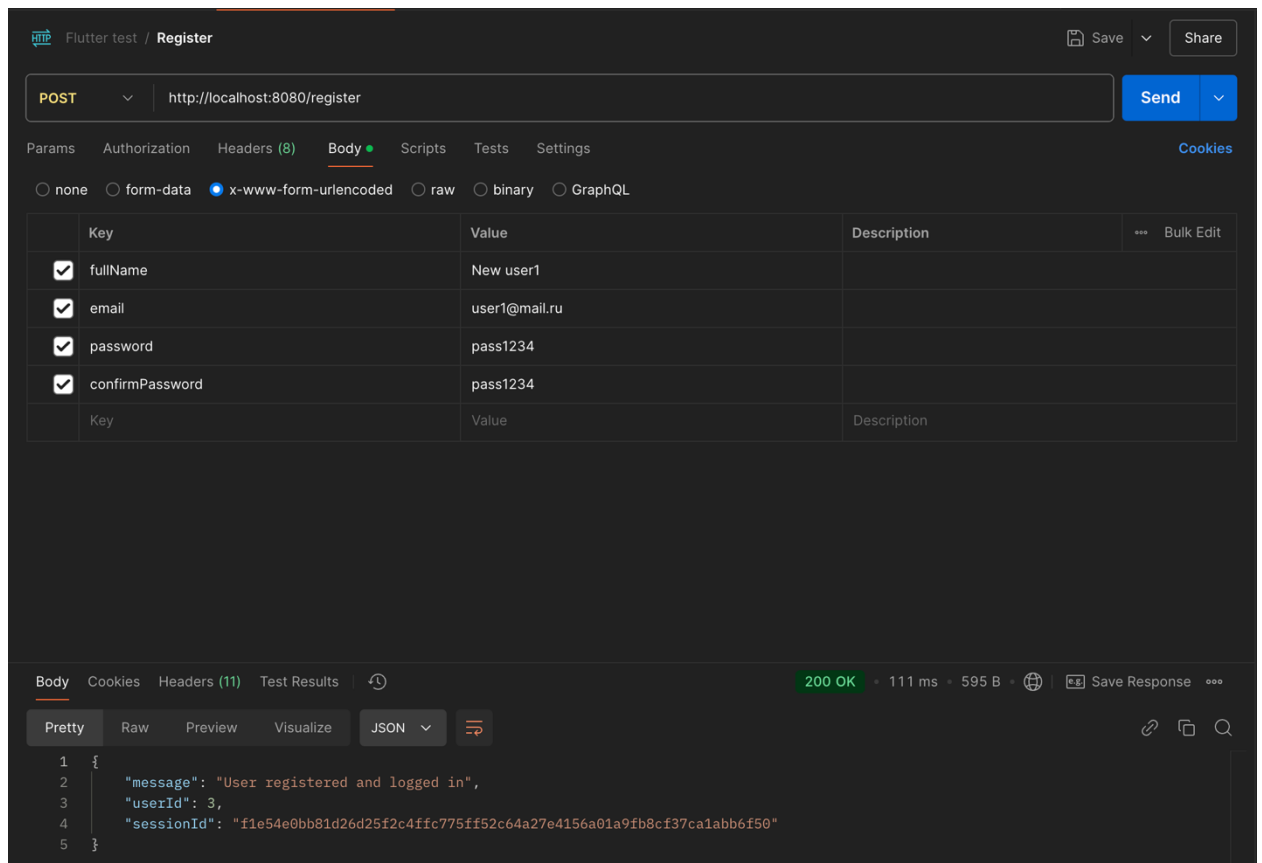


Рисунок 5 – Создание нового пользователя

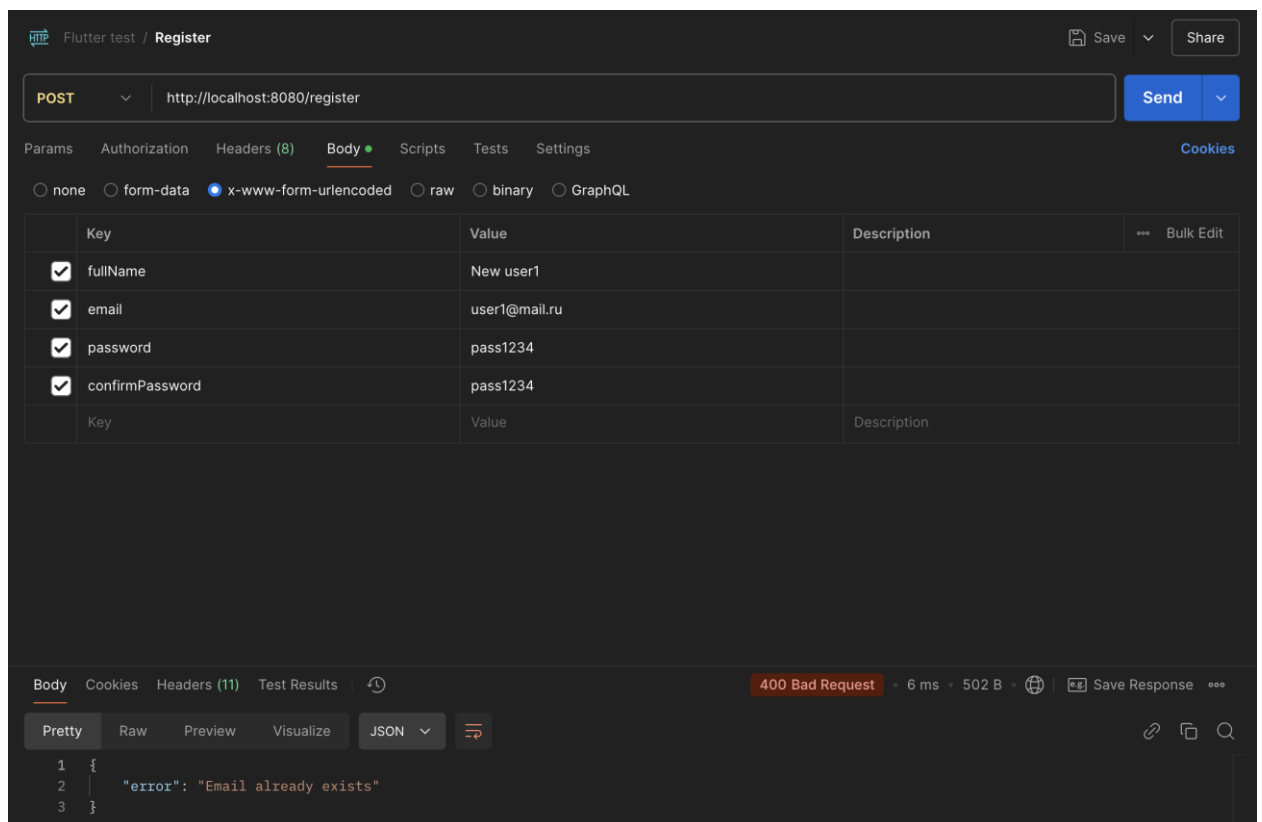


Рисунок 6 – попытка создания пользователя с тем же email

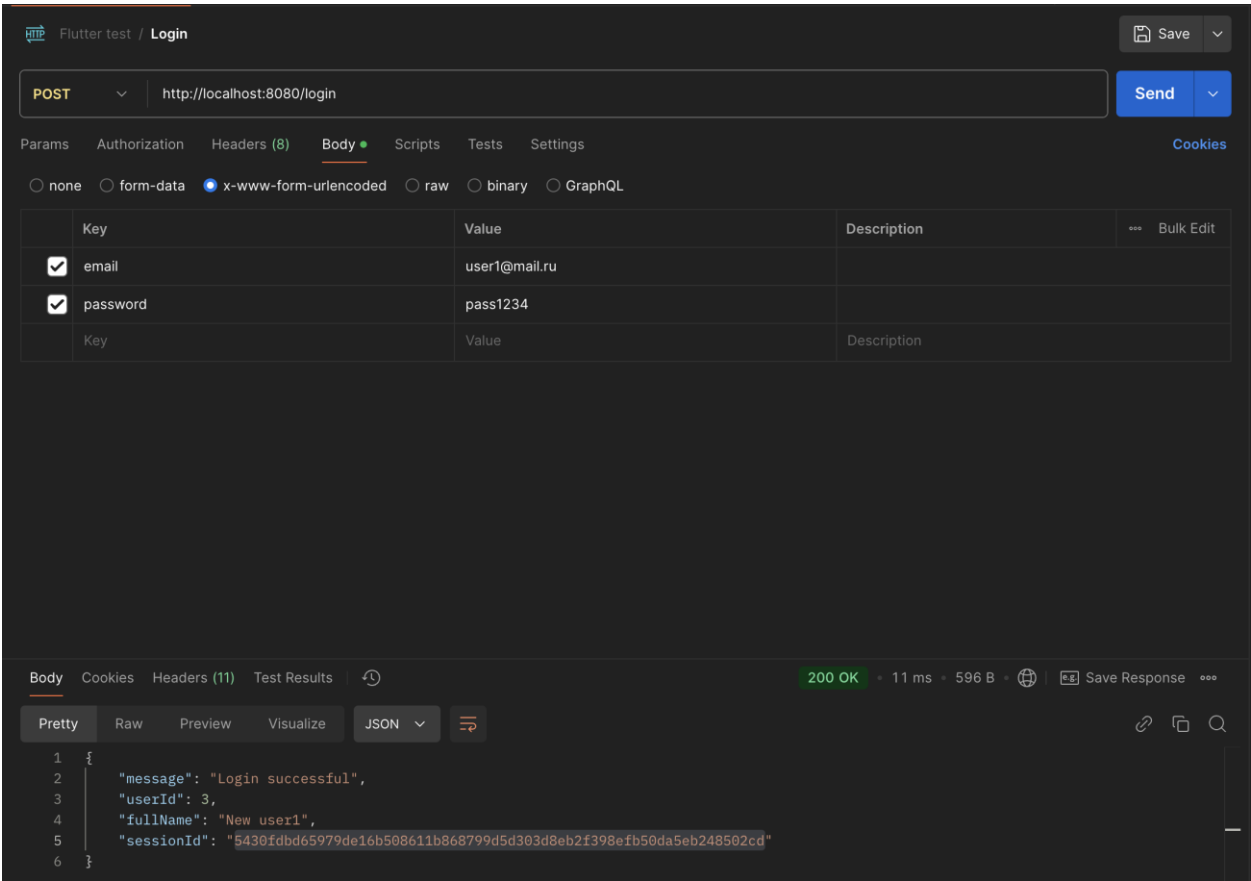


Рисунок 7 – авторизация пользователя

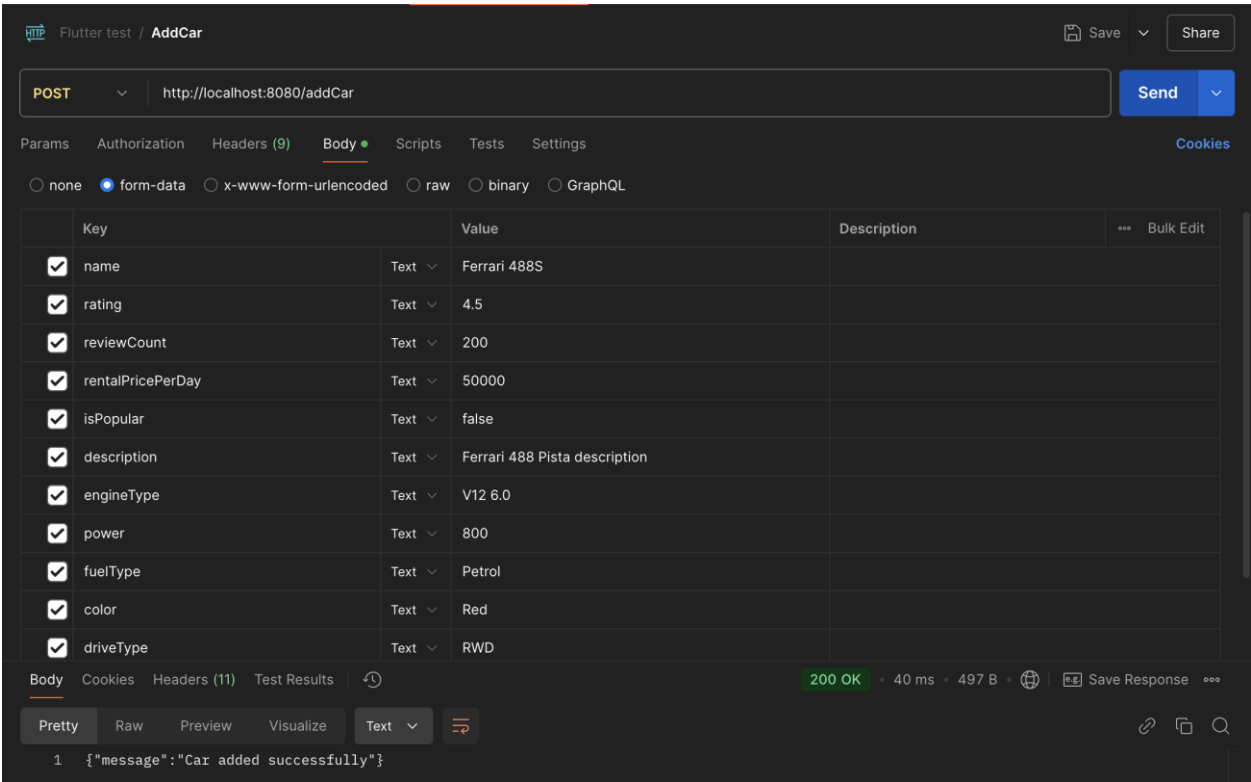


Рисунок 8 – Добавление нового автомобиля с передачей sessionId

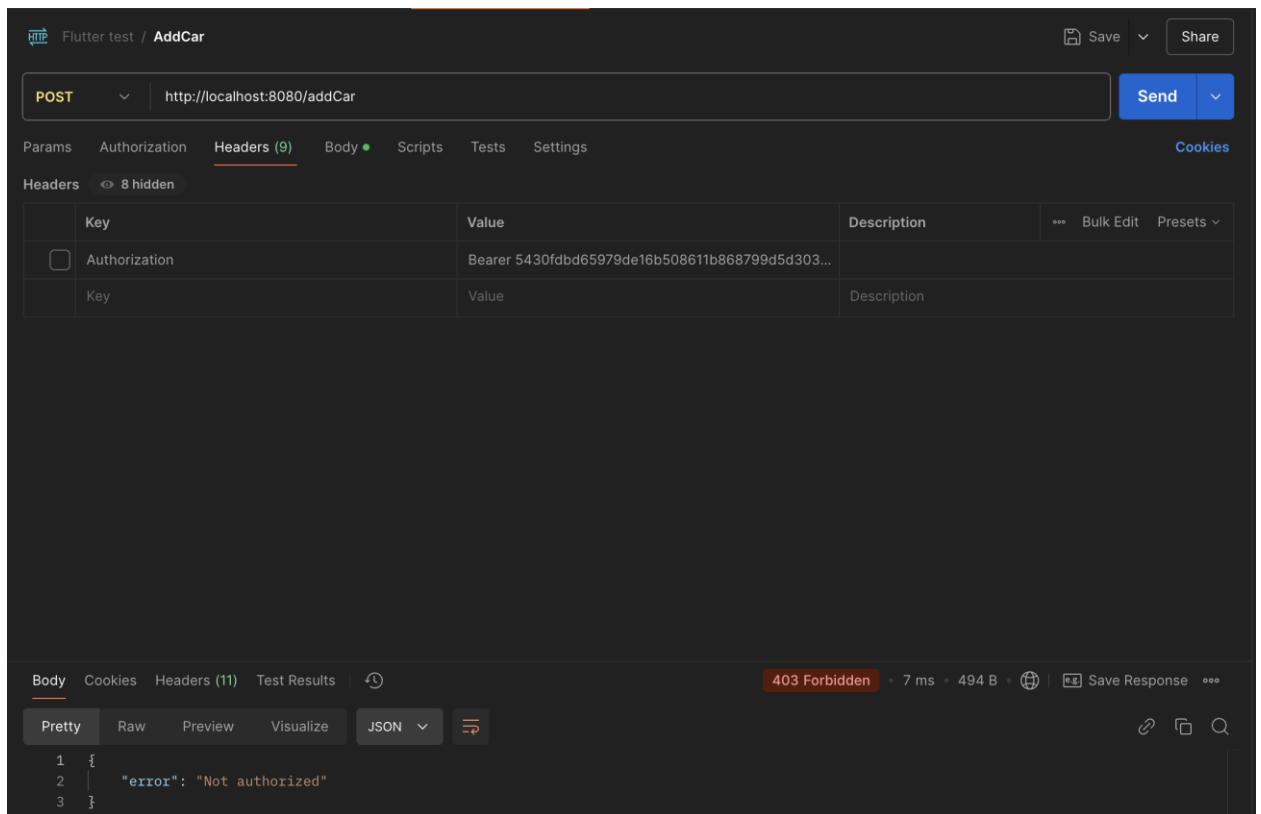


Рисунок 9 – Добавление нового автомобиля без передачи sessionId

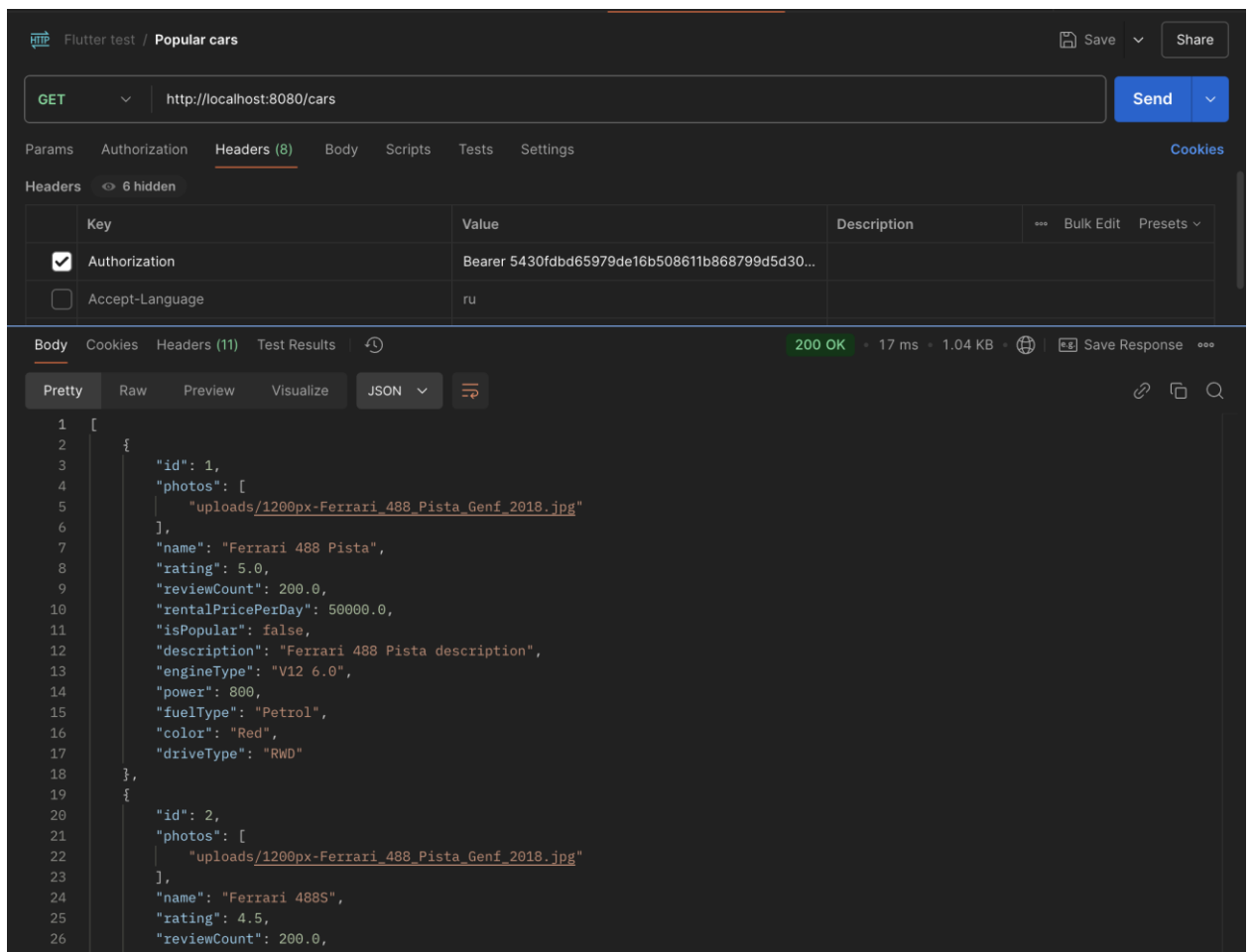


Рисунок 10 – получение всех автомобилей

### 3.2 Реализация клиентской части

Откроем наше приложение (клиент), в папке `lib` создадим новую папку `Services`, создадим там файл `network_service.dart`. Добавим необходимые библиотеки в `pubspec.yaml` – листинг 51, а потом в файл `network_service.dart` – листинг 52.

Листинг 51 – Добавление библиотек в `pubspec.yaml`

```
dependencies:  
  flutter:  
    sdk: flutter  
  http: ^1.2.2
```

Листинг 52 – добавление необходимых библиотек

```
import 'dart:io';  
import 'dart:typed_data';  
import 'package:http/http.dart' as http;  
import 'dart:convert';  
import 'package:mime/mime.dart';  
import 'package:http_parser/http_parser.dart';
```

Создадим класс `NetworkService`. Добавим константой ссылку на базовый URL нашего сервера – листинг 53.

Листинг 53 – Network Service

```
class NetworkService {  
  final String baseUrl = 'http://localhost:8080';  
}
```

Создадим единую точку доступа к классу, чтобы не создавать на каждом из экранов новый экземпляр `NetworkService` – Листинг 54.

Листинг 54 – создание единой точки доступа (Singleton)

```
class NetworkService {  
  final String baseUrl = 'http://localhost:8080';  
  
  static final NetworkService _instance = NetworkService._internal();  
  
  NetworkService._internal();  
  
  factory NetworkService() {  
    return _instance;  
  }  
}
```

Создадим приватные переменные, которые будут хранить `id` сессии и `id` пользователя. Перейдем к созданию функций для взаимодействия клиента с сервером:

- Авторизация;
- Регистрация;
- Получение данных о пользователе;

- Обновление данных пользователя;
- Получение популярных автомобилей;
- Получение всех автомобилей;
- Получение всех автомобилей;
- Выход;

Функция авторизации – принимает в качестве параметров email и пароль. Внутри функции создадим переменную url, которая будет хранить полный url запроса. Далее происходит попытка post запроса по созданному url, передачи заголовка content-type и передачи в body полей email и password, если запрос выполнен успешно, то декодируем тело ответа при помощи jsonDecode, и в \_sessionId записываем поле 'sessionId', а в \_userId – 'userId' после чего выходим из функции и возвращаем необходимые параметры, если что-то пошло не так, то обрабатываем сообщение об ошибке – листинг 55.

Листинг 55 – функция авторизации

```
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);
      _sessionId = responseData['sessionId'];
      _userId = responseData['userId'];

      return {
        'success': true,
        'message': responseData['message'],
        'userId': _userId,
        'fullName': responseData['fullName'],
        'sessionId': _sessionId,
      };
    } else {
      return _handleErrorResponse(response);
    }
  } catch (e) {
    return {
      'success': false,
      'error': 'Something went wrong. Please try again later.'
    };
  }
}
```



```

Map<String, dynamic> _handleErrorResponse(http.Response response) {
  final errorData = jsonDecode(response.body);
  return {'success': false, 'error': errorData['error'] ?? 'Request failed'};
}

```

Функция регистрации аналогична авторизации, только имеет другой путь и входные параметры – листинг 56.

Листинг 56 – функция регистрации

```

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);
      _sessionId = responseData['sessionId'];
      _userId = responseData['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.'
    };
  }
}

```

Теперь необходимо создать функции, которые будут возвращать id сессии и пользователя – листинг 57.

Листинг 57 – функции, возвращаемые sessionId и userId

```
Future<String?> _getSessionId() async {
  return _sessionId;
}

Future<int?> _getUserId() async {
  return _userId;
}
```

Создадим функцию получения данных о пользователе. Создадим внутри переменные, которые будут хранить в себе id сессии и id пользователя, переменную, которая будет хранить полный url. При помощи try делаем запрос к url, аналогично авторизации и регистрации, только используем вместо post – get запрос, если запрос успешен, то возвращаем тело ответа из функции, а если что-то пошло не так – ошибку – листинг 58.

Листинг 58 – функция получения данных о пользователе

```
Future<Map<String, dynamic>> getUserData() async {
  final sessionId = await _getSessionId();
  final userId = await _getUserId();
  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.'
    };
  }
}
```

Создадим функцию обновления данных пользователя. Аналогично предыдущему запросу – получаем id сессии, формируем полный url, создаем объект `http.MultipartRequest`, так как у нас post запрос, в котором есть возможность передачи файла, а заголовок запроса 'Authorization' передаем 'Bearer id сессии'. Далее проверяем переданные поля, если оно есть, то добавляем в поле `fields` запроса. Если передается фотография, то надо проверять, что передается.

- Если это `File`, то определяем MIME тип файла, после чего добавляем файл в запрос с помощью `MultipartFile.fromPath`;
- Если это `Uint8List`, то данные преобразуем в файл и добавляет их с помощью `MultipartFile.fromBytes`.

После чего отправляем запрос на сервер с помощью `request.send()`. Если ответ успешен, то возвращаем `true` и сообщение полученное от сервера, иначе – обрабатываем ошибку - листинг 59.

Листинг 59 – функция обновления данных пользователя

```
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) {
        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.'
    };
}
}

```

Так как запросы на получение всех автомобилей, популярных автомобилей, акций / новостей похожи, то создадим функцию, которая будет возвращать список, а принимать полный url – листинг 60.

Листинг 60 – функция запроса для получения ответа в виде списка

```

Future<List<Map<String, dynamic>>> _getWithSession(String url) async {
    final sessionId = await _getSessionId();

    try {
        final response = await http.get(
            Uri.parse(url),

```

```

        headers: {
            'Content-Type': 'application/json',
            if (sessionId != null) 'Authorization': 'Bearer $sessionId'
        },
    );

    if (response.statusCode == 200) {
        return (jsonDecode(response.body) as List).cast<Map<String, dynamic>>();
    } else {
        return [];
    }
} catch (e) {
    return [];
}
}

```

В листингах 61 – 63 представлен код для запросов получения всех автомобилей, популярных автомобилей, акций / новостей.

Листинг 61 – запрос на получение всех автомобилей

```

Future<List<Map<String, dynamic>>> getAllCars() async {
    return _getWithSession('$baseUrl/cars');
}

```

Листинг 62 – запрос на получение популярных автомобилей

```

Future<List<Map<String, dynamic>>> getPopularCars() async {
    return _getWithSession('$baseUrl/cars/popular');
}

```

Листинг 63 – запрос на получение всех акций / новостей

```

Future<List<Map<String, dynamic>>> getPromotions() async {
    return _getWithSession('$baseUrl/promotions');
}

```

Добавим теперь функцию выхода, она аналогична предыдущим post запросам – листинг 64.

Листинг 64 – функция выхода

```

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) {
  _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> _clearSessionData() async {
  _sessionId = null;
  _userId = null;
}

```

Листинг 65 – полный код NetwoksService

```

import 'dart:io';
import 'dart:typed_data';
import 'package:http/http.dart' as http;
import 'dart:convert';
import 'package:mime/mime.dart';
import 'package:http_parser/http_parser.dart';

class NetworkService {
  final String baseUrl = 'http://localhost:8080';

  static final NetworkService _instance = NetworkService._internal();

  NetworkService._internal();

  factory NetworkService() {
    return _instance;
  }

  String? _sessionId;
  int? _userId;

  Future<String?> _getSessionId() async {
    return _sessionId;
  }

  Future<int?> _getUserId() async {
    return _userId;
  }
}

```

```

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);
      _sessionId = responseData['sessionId'];
      _userId = responseData['userId'];

      return {
        'success': true,
        'message': responseData['message'],
        'userId': _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>> 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);
  _sessionId = responseData['sessionId'];
  _userId = responseData['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<Map<String, dynamic>> getUserData() async {
  final sessionId = await _getSessionId();
  final userId = await _getUserId();
  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) {
        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<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();

    try {
        final response = await http.get(
            Uri.parse(url),
            headers: {
                'Content-Type': 'application/json',
                if (sessionId != null) 'Authorization': 'Bearer $sessionId'
            },
        );

        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) {
    _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> _clearSessionData() async {
  _sessionId = null;
  _userId = null;
}

Map<String, dynamic> _handleErrorResponse(http.Response response) {
  final errorData = jsonDecode(response.body);
  return {'success': false, 'error': errorData['error'] ?? 'Request failed'};
}

```

Перейдем к настройкам страниц. Для начала настроим loginPage().

Импортируем путь до нашего NetworkService

```
import '../Services/network_service.dart';
```

Обновим наш CustomTextField, чтобы он мог принимать в себя контроллер и переданный текст – листинг 66.

Листинг 66 – обновленный CustomTextField

```
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),
  ),
);
}

```

Создадим TextEditingController для поля email и для поля password – листинг 67.

#### Листинг 67 создание TextEditingController

```
final TextEditingController _emailController = TextEditingController();
final TextEditingController _passwordController = TextEditingController();

bool isEnabled = false;

@override
void initState() {
  super.initState();
  _emailController.addListener(_checkFields);
  _passwordController.addListener(_checkFields);
}
// остальной код

CustomTextField(
  controller: _emailController,
  labelText: "Email address"
),
const SizedBox(height: 25),
CustomTextField(
  controller: _passwordController,
  labelText: "Password",
  obscure: true,
),
```

Добавим networkService в файл и создадим функцию login, которая будет выполнять http запрос. Создадим внутри функции две переменных, которые будут хранить текст из полей email и password. Произведем выполнение запроса, если ответ успешен – то производим переход на главную страницу, если ошибка, то выведем диалоговое окно с сообщением – листинг 68.

#### Листинг 68 – функция login

```
Future<void> _login() async {
  final email = _emailController.text;
  final password = _passwordController.text;

  final result = await networkService.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("Login Failed"),
      content: Text(result['error'] ?? "Login failed. Please try again."),
      actions: [
        TextButton(
          onPressed: () {
            Navigator.of(context).pop();
          },
          child: const Text('OK'),
        ),
      ],
    );
  },
);
}

```

Создадим функцию проверки полей, которая будет устанавливать `isButtonEnabled`, если поле `email` и поле `password` не пустые – листинг 69.

Листинг 69 - функцию проверки полей

```

void _checkFields() {
  setState(() {
    isButtonEnabled =
      _emailController.text.isNotEmpty &&
      _passwordController.text.isNotEmpty;
  });
}

```

Обернем кнопку `login` в `opacity`, чтобы визуально отображать ее доступность / недоступность, в зависимости от заполненности полей, а также добавим вызов функции `login` при нажатии на кнопку – листинг 70.

Листинг 70 – обновление кнопки `login`

```

Padding(
  padding: EdgeInsets.fromLTRB(0, 9, 0, 53),
  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(
        "Login",

```

```
style: TextStyle(
  fontFamily: "Urbanist",
  fontSize: 25,
  color: Colors.white,
  fontWeight: FontWeight.w600,
),
),
),
),
),
```

Листинг 71 – полный код LoginPage()

```
import 'package:flutter/material.dart';
import '../Services/network_service.dart';
import 'tab_bar.dart';
import '../Components/text_field.dart';
import 'package:flutter/foundation.dart';
import 'dart:io';

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 = NetworkService();

  bool isEnabled = 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 networkService.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("Login Failed"),
      content: Text(result['error'] ?? "Login failed. Please try again."),
      actions: [
        TextButton(
          onPressed: () {
            Navigator.of(context).pop();
          },
          child: const Text('OK'),
        ),
      ],
    );
  },
);
}
}

void _checkFields() {
  setState(() {
    isEnabled =
      _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: [
      Padding(
        padding: EdgeInsets.fromLTRB(0, 82, 0, 62),
        child: Text(
          "Welcome!\nLogin to your account,\nOr create new one",
          style: TextStyle(
            fontSize: 24,
            fontWeight: FontWeight.bold,
            color: Color(0xFF1B588C),
          ),
        ),
      ),
      CustomTextField(
        controller: _emailController,
        labelText: "Email address"
      ),
      const SizedBox(height: 25),
      CustomTextField(

```



```
controller: _passwordController,  
labelText: "Password",  
isObscure: true,  
),  
Align(  
alignment: Alignment.centerRight,  
child: TextButton(  
onPressed: () {},  
child: Text(  
"Forgot password?",  
style: TextStyle(  
color: const Color.fromARGB(255, 178, 188, 222).withOpacity(0.7),  
fontFamily: "Urbanist",  
fontSize: 12,  
fontWeight: FontWeight.w600,  
),  
),  
),  
),  
Padding(  
padding: EdgeInsets.fromLTRB(0, 9, 0, 53),  
child: Opacity(  
opacity: isEnabled ? 1.0 : 0.5,  
child: ElevatedButton(  
onPressed: isEnabled ? login : null,  
style: ElevatedButton.styleFrom(  
backgroundColor: const Color(0xFF192252),  
shape: RoundedRectangleBorder(  
borderRadius: BorderRadius.circular(8),  
),  
minimumSize: const Size(double.infinity, 50),  
),  
child: Text(  
"Login",  
style: TextStyle(  
fontFamily: "Urbanist",  
fontSize: 25,  
color: Colors.white,  
fontWeight: FontWeight.w600,  
),  
),  
),  
),  
Row(  
children: [  
Expanded(  
child: Divider(  
color: const Color(0xFF424F7B).withOpacity(0.5),  
),  
),  
Padding(  
padding: EdgeInsets.symmetric(horizontal: 8.0),
```

```

        child: Text(
          "or login with",
          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: "Don't have an account? ",
      style: TextStyle(
        fontFamily: "Urbanist",
        color: Color(0xFF424F7B),

```

```

        fontSize: 18,
      ),
      children: <TextSpan>[
        TextSpan(
          text: "Register",
          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();
}
}

```

Теперь настроим RegisterPage(). Повторяем аналогичные действия как на loginPage.

Листинг 72 – полный код RegisterPage

```

import 'package:flutter/material.dart';
import '../Services/network_service.dart';
import 'tab_bar.dart';
import '../Components/text_field.dart';
import 'package:flutter/foundation.dart';
import 'dart:io';

```

```

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 = NetworkService();

  bool isEnabled = false;

  @override
  void initState() {
    super.initState();
    _fullNameController.addListener(_checkFields);
    _emailController.addListener(_checkFields);
    _passwordController.addListener(_checkFields);
    _confirmPasswordController.addListener(_checkFields);
  }

  void _checkFields() {
    setState(() {
      isEnabled =
        _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 networkService.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("Registration Failed"),
    content: Text(result['error'] ?? "Registration failed. Please try again."),
    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: "Full name"
    ),
    const SizedBox(height: 24),
    CustomTextField(
      controller: _emailController,
      labelText: "Email address"
    ),
    const SizedBox(height: 24),
    CustomTextField(
      controller: _passwordController,
      labelText: "Password",
      isObscure: true,
    ),
    const SizedBox(height: 24),
    CustomTextField(
      controller: _confirmPasswordController,
      labelText: "Confirm password",
      isObscure: true,
    ),
    const SizedBox(height: 43),
    Center(
      child: Opacity(
        opacity: isEnabled ? 1.0 : 0.5,
        child: ElevatedButton(

```

```

onPressed: isEnabled ? _register : null,
style: ElevatedButton.styleFrom(
  backgroundColor: const Color(0xFF192252),
  shape: RoundedRectangleBorder(
    borderRadius: BorderRadius.circular(8),
  ),
  minimumSize: const Size(double.infinity, 50),
),
child: Text(
  "Register",
  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: "Already have an account? ",
        style: TextStyle(
          fontFamily: "Urbanist",
          color: Color(0xFF424F7B),
          fontSize: 18,
        ),
      ),
      children: <TextSpan>[
        TextSpan(
          text: "Login",
          style: TextStyle(
            fontFamily: "Urbanist",
            color: Color(0xFF103F74),
            fontWeight: FontWeight.bold,
          ),
        ),
      ],
    ),
  ),
),
),
),
),
),
),
);

```

```

return Scaffold(
  appBar: AppBar(
    backgroundColor: Colors.white,
    title: Text(
      "Register",
      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();
}
}

```

Теперь настроим HomePage(). Для начала создадим модель данных Car. Для этого в lib создадим папку Models и создадим файл car\_model.dart – листинг 73.

Листинг 73 – код CarModel

```

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,
    });
}

```

Аналогично предыдущим экранам добавляем NetworkService. Создаем переменные, которые будут хранить акции, автомобили, популярные автомобили – листинг 74.

Листинг 74 – создание переменных

```

List<Map<String, dynamic>> allPromotions = [];
List<Car> popularCars = [];
List<Car> allCars = [];
bool isLoading = true;

```

Создадим функцию, которая будет конвертировать ответ запроса в массив типа Car – листинг 75.

Листинг 75 – функция конвертации ответа в массив

```

List<Car> convertToCarList(List<Map<String, dynamic>> carData) {
    return carData.map((data) {
        return Car(
            name: data['name'],
            imageUrl: data['photos'].isEmpty ? 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();
}

```

Создадим функции получения автомобилей и акций – листинг 76.

Листинг 76 – функции получения автомобилей и акций.

```

@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;
  });
}

```

Добавим функцию обновления данных, чтобы можно было перезагружать экран – листинг 77.

Листинг 77 – функция обновления данных

```

Future<void> _refreshData() async {
  setState(() {
    isLoading = true;
  });

  await fetchPromotions();
  await fetchCarsData();

  setState(() {

```

```

        isLoading = false;
    });
}

```

Внесем правку в Positioned.fill, где child – добавляем проверку isLoading, если да, то будем показывать индикатор загрузки, иначе RefreshIndicator с вызовом в onRefresh функции refreshData – листинг 78.

Листинг 78 – правка в Positioned.fill

```

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(

```

Теперь перейдем в блок, где происходит настройка отображения промоакции / новости. В параметр itemCount в PageView передаем количество из массива allPromotions при помощи вызова length. В itemBuilder создаем экземпляр промоакции путем обращения к allPromotions и передачи index – номера страницы, создаем переменную, которая будет хранить фотографию – promotion['photo']. В BoxDecoration в color проверяем, что изображение не пустое, если пустое, то серый цвет, иначе – null, в image проверяем imageUrl на пустоту, если не пустое, то возвращаем DecorationImage(), где в image мы передаем NetworkImage('полный путь до изображения: базовая ссылка + ссылка из ответа), в название акции передаем promotion['name']. Далее настроим индикатор страниц. В child проверим длину массива, если больше 1, то показываем SmoothPageIndicator с allPromotions.length, иначе SizedBox.shrink() – листинг 79

Листинг 79 – обновленный блок с промоакциями

```

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(),
),
),
),

```

Перейдем к блоку с популярными автомобилями. В `ListView.builder` в поле `item count` передадим длину массива `popularCars`, в карточку будем передавать элемент из `popularCars` с нужным индексом и полем – листинг 82.

Также обновим `DetailPage`. В `StatefulWidget` добавим в качестве принимаемого параметра – модель `Car`, чтобы можно было из `HomePage` легко передавать выбранный автомобиль – листинг 80.

### Листинг 80 – обновленный DetailPage

```
import 'package:flutter/material.dart';
import 'package:smooth_page_indicator/smooth_page_indicator.dart';
import '../Models/car_model.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(0xFFFFFBD16),
        ),
        const SizedBox(width: 3),
        Text(
          '${widget.car.rating}',
          style: const TextStyle(
            fontSize: 20,
            color: Color(0xFFFFFBD16),
          ),
        ),
        const SizedBox(width: 10),
        Text(
          widget.car.reviewCount >= 100
            ? "(100+ reviews)"
            : "(${widget.car.reviewCount} reviews)",
          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(
      "CAR INFO",

```

```

        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: "Engine", value: widget.car.engine),
        const SizedBox(height: 20),
        InfoRow(label: "Power", value: '${widget.car.power}hp'),
        const SizedBox(height: 20),
        InfoRow(label: "Fuel", value: widget.car.fuel),
        const SizedBox(height: 20),
        InfoRow(label: "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(
  '${widget.car.rentalPricePerDay} P / 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(
      "Rent Car",
      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,
        ),
      ),
    ],
  );
}
}

```

Обновим main.dart. Добавим onGenerateRoute, чтобы можно было передавать в DetailPage модель данных Car – листинг 81.

Листинг 81 – обновленный 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';

void main() {
  runApp(const MyApp());
}

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

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      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 82 – Обновление блока Popular Cars

```

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

Производим аналогичное действие и для блока со всеми автомобилями — листинг 83.

### Листинг 83 – обновленный блок All cars

```

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
          ),
        );
      },
    );
  },
);

```

```
),
```

Обновим carCard, а именно – где происходит добавление изображения, используем NetworkImage.

Листинг 84 – полный код HomePage

```
import 'package:flutter/material.dart';
import 'package:smooth_page_indicator/smooth_page_indicator.dart';
import '../Services/network_service.dart';
import '../Models/car_model.dart';
import 'package:intl/intl.dart';
import 'package:flutter/foundation.dart';
import 'dart:io';

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

  @override
  _HomePageState createState() => _HomePageState();
}

class _HomePageState extends State<HomePage> {

  final PageController _pageController = PageController();
  final 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'].isEmpty ? 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(

```



```

        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(
        "Popular cars",
        style: TextStyle(
          fontSize: 20,
          fontWeight: FontWeight.bold,
        ),
      ),
      Text(
        "see all",
        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(
    "All cars",
    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(

```

        "Your location",
        style: TextStyle(
          color: Color(0xFF848FAC),
          fontSize: 18,
          fontFamily: "Urbanist",
        ),
      ),
      SizedBox(width: 10),
      ImageIcon(AssetImage('assets/images/arrow-down.png')),
    ],
  ),
  Text(
    "Moscow, Russia",
    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 = "(100+ reviews)";
} else {
  formattedReviews = "($numOfReviewsInt reviews)";
}

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(0xFFFFFBD16),
      ),
      const SizedBox(width: 3),
      Text(
        rating,
        style: const TextStyle(
          fontSize: 15,
          color: Color(0xFFFFFBD16),
        ),
      ),
      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(
    '$price ₺ / day',
    style: const TextStyle(
      fontSize: 15,
      color: Color(0xFF192252),
    ),
  ),
),
],
);
}
}

```

Теперь настроим ProfilePage(). Аналогично предыдущим – добавляем NetworkService. Создаем переменные, которые будут хранить FullName, email, profileImageUrl. Добавляем функцию загрузки данных пользователя – листинг 85.

Листинг 85 – Функция загрузки данных

```

Future<void> _loadUserData() async {
  final response = await networkService.getUserData();

  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'),
      ),
    );
  }
}

```

Добавим функцию выхода – листинг 86.

Листинг 86 – функция выхода

```

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')),
    );
  }
}

```

Внесем правки в блок с информацией о пользователе. Если profileImageUrl не null и не пустой, то при помощи Image.network вставляем изображение из ссылки, иначе – просто серый цвет. В поле Text, где fullName – вставляем fullname, если он не null, иначе – Loading... - Листинг 87

Листинг 87 – обновление контейнера с информацией о пользователе

```

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,
),
],
),

```

Теперь обновим EditUserPage. В класс EditUserPage добавим переменные, которые потом будут приниматься при навигации – листинг 88.

Листинг 88 – обновление класса EditUserPage

```

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();
}

```

Обновим навигацию в ProfilePage и сделаем вызов функции получения данных, как только мы возвращаемся с EditUserPage – листинг 89.

Листинг 89 – обновление навигации в ProfilePage

```
onTap: () async {
  await Navigator.push(
    context,
    MaterialPageRoute(
      builder: (context) => EditUserPage(
        initialFullName: fullName,
        initialEmail: email,
        initialProfileImageUrl: profileImageUrl,
      )),
  );
  _loadUserData();
},
```

Добавляем TextEditingController для полей с fullName и email. Добавим две переменных File и Uint8List для создания пикера изображения, в зависимости от типа устройства. Напишем функцию pickImage, которая будет вызывать пикер – листинг 90.

Листинг 90 – функция пикера

```
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;
      });
    }
  }
}
```

Добавим функцию обновления данных пользователя – листинг 91.

Листинг 91 – функция обновления данных пользователя

```
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("User updated successfully")),
  );
} else {
  ScaffoldMessenger.of(context).showSnackBar(
    SnackBar(content: Text('Error: ${response['error']}')),
  );
}
}

```

В CustomTextFields в поле initialText вставляем fullname, если оно есть или текст «Enter your name», делаем аналогичное действие для поля с email. При нажатии на кнопку UpdateInfo – вызываем функцию updateInfo – листинг 92.

Листинг 92 – вызов функции updateUserInfo

```

CustomActionButton(
  label: "Update info",
  isPrimary: true,
  onTap: _updateUserInfo,
),

```

Листинг 93 – полный код EditUserPage

```

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/network_service.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();

    @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("User updated successfully")),
        );
    } 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(
                        "Update user info",
                        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: "Full name",
  initialText: widget.initialFullName ?? "Enter your name",
),
const SizedBox(height: 25),
CustomTextField(
  controller: _emailController,
  labelText: "Email address",
  initialText: widget.initialEmail ?? "Enter your email",
),
isMobile ?
const Spacer() : SizedBox(height: 220),
CustomActionButton(
  label: "Update info",
  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 94 – полный код ProfilePage

```

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:flutter/foundation.dart';
import 'dart:io';
import '../Services/network_service.dart';

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

  @override
  _ProfilePageState createState() => _ProfilePageState();
}

class _ProfilePageState extends State<ProfilePage> {
  final networkService = NetworkService();
  String? fullName;
  String? email;
  String? profileImageUrl;
  bool isLoading = true;

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

```

```

_loadUserData();
}

Future<void> _loadUserData() async {
  final response = await networkService.getUserData();

  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: "Change password",
  onTap: () {},
),
const SizedBox(height: 21),
CustomButton(
  iconPath: 'assets/images/empty-wallet.png',
  label: "Billing information",
  onTap: () {},
),
const SizedBox(height: 21),
CustomButton(
  iconPath: 'assets/images/direct-inbox.png',
  label: "Notifications",
  onTap: () {},
),
isMobile ? const Spacer() : const SizedBox(height: 130),
Column(
  children: [
    CustomActionButton(
      label: "Log Out",
      isPrimary: true,
      onTap: () async {
        await _logout();
        Navigator.pushNamedAndRemoveUntil(
          context,
          '/login',
          (Route<dynamic> route) => false,
        );
      },
    ),
    const SizedBox(height: 20),
    CustomActionButton(
      label: "Delete account",
      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(
        "Profile",
        style: TextStyle(
          color: Color(0xFF192252),
          fontSize: 25,
          fontWeight: FontWeight.bold,
        ),
      ),
    ),
    centerTitle: false,
  ),
),
],
),
);
}
}

```

## Листинг 95 – 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/Core
# FoundationKeys.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
  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
  file_picker: ^6.1.1

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/bmw5.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/applcon.png" # Укажите путь к вашей иконке
flutter_intl:
```

enabled: true

Произведем проверку работы на мобильном устройстве. Результат представлен на рисунках 11 – 15

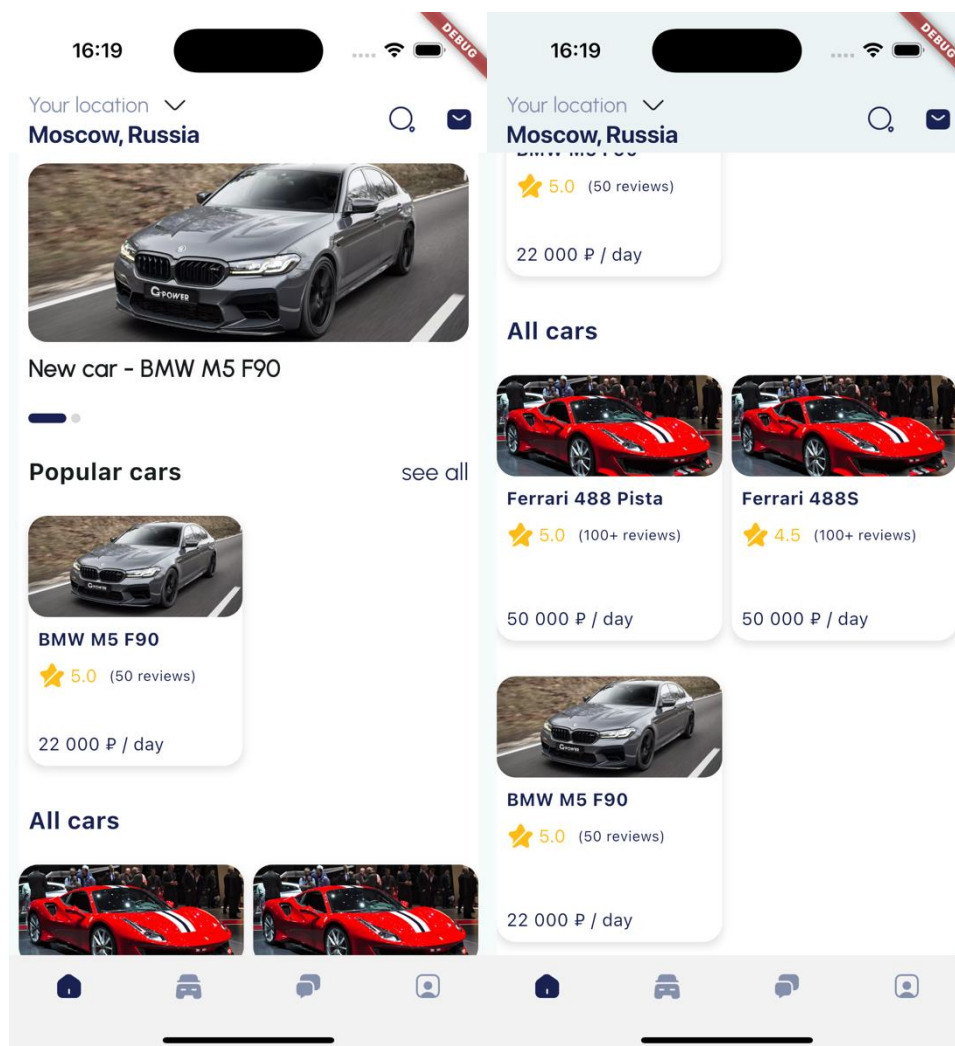


Рисунок 11 – Главный экран

16:19

16:19

BMW M5 F90

★ 5.0 (50 reviews)

BMW M5 F90 super Description

CAR INFO

Engine

V8 4.4L Twin-Turbo

Power

625 hp

Fuel

Petrol

Color

Grey

Drivetrain

AWD

BMW M5 F90

22 000 ₰ / day

Rent Car

16:19

16:19

Ferrari 488 Pista

★ 5.0 (100+ reviews)

Ferrari 488 Pista description

CAR INFO

Engine

V12 6.0

Power

800 hp

Fuel

Petrol

Color

Red

Drivetrain

RWD

Ferrari 488 Pista

50 000 ₰ / day

Rent Car

Рисунок 12 – Детальная информация об автомобиле

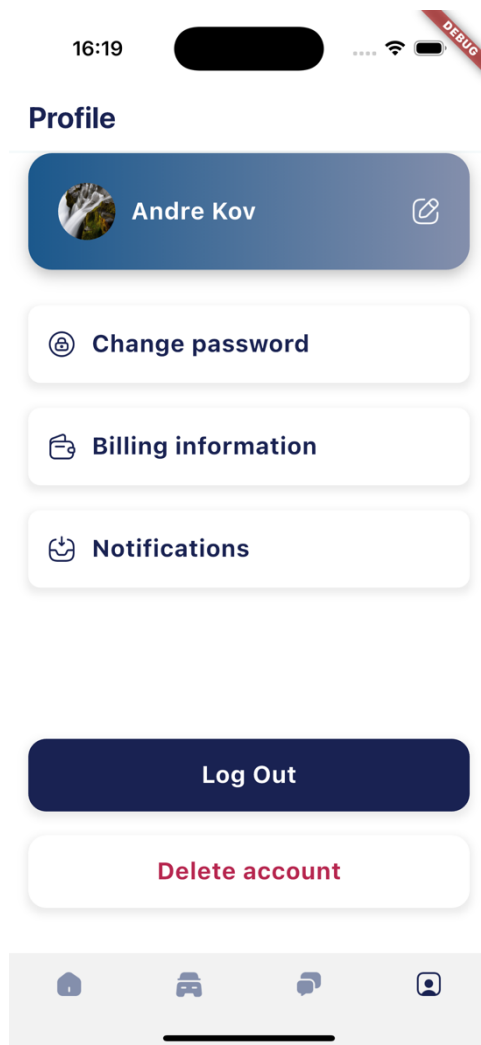


Рисунок 13 – Экран профиля

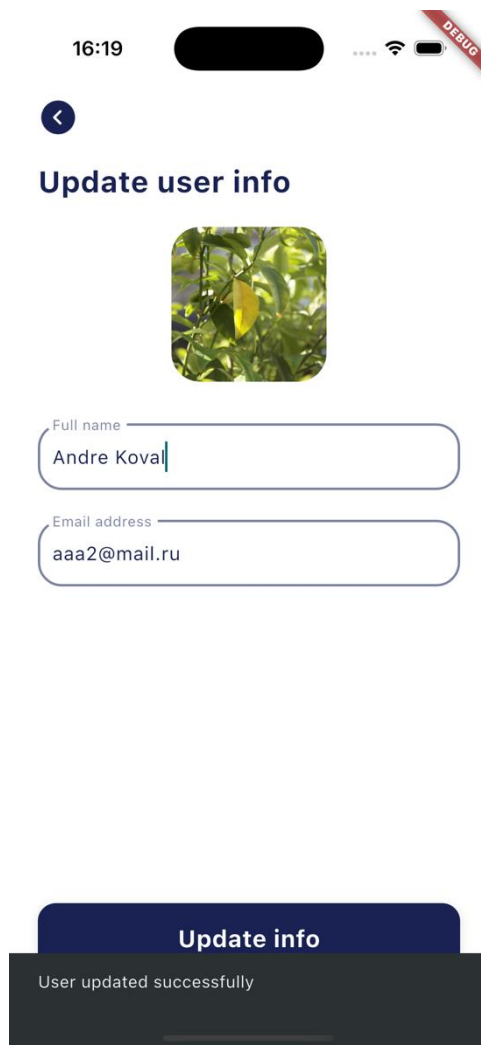


Рисунок 14 – Обновление данных

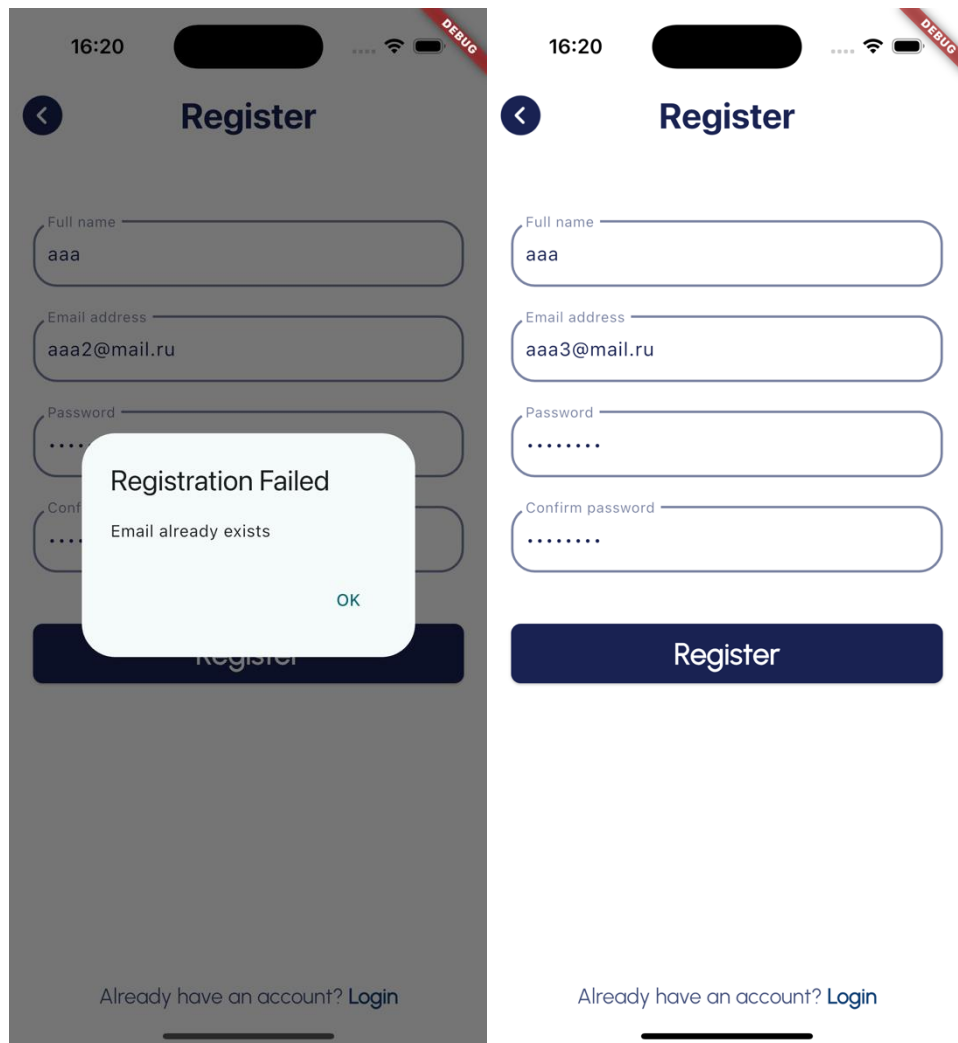


Рисунок 15 – Регистрация пользователя