ГУАП

КАФЕДРА № 41

ОТЧЕТ   
ЗАЩИЩЕН С ОЦЕНКОЙ

ПРЕПОДАВАТЕЛЬ

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

|  |
| --- |
| ОТЧЕТ О ПРАКТИЧЕСКОЙ РАБОТЕ №4 |
| «Локальное хранение данных» |
| по курсу: «Методы объектно-ориентированного проектирования» |
|  |
|  |

РАБОТУ ВЫПОЛНИЛ

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

Санкт-Петербург 2024

1. **Цель работы**

Реализовать для клиентского приложения из третьей практической работы локальное хранение и кеширование ряда данных, получаемых по сети.

1. **Процесс выполнения работы**

На основе разработанного дизайна и спецификации OpenApi, будут сохраняться следующие данные:

1. email – shared preferences (локальное хранилище данных);
2. Сохранение password – flutter secure storage (защищенное локальное хранилище данных);
3. sessionId – кешируется на время работы приложения.
4. userId – кешируется на время работы приложения.

Для того, чтобы реализовать данный функционал добавим библиотеки shared\_preferences и flutter\_secure\_storage в список зависимостей в файле pubspec.yaml – листинг 1.

Листинг 1 – Добавление зависимостей

|  |
| --- |
| shared\_preferences: ^2.0.6  flutter\_secure\_storage: ^9.0.0 |

Листинг 2 – Полный код pubspec.yaml

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

Теперь перейдем в файл network\_service.dart. Импортируем библиотеки – листинг 3.

Листинг 3 – импорт библиотек

|  |
| --- |
| import 'package:shared\_preferences/shared\_preferences.dart';  import 'package:flutter\_secure\_storage/flutter\_secure\_storage.dart'; |

Создадим приватную функцию \_saveSessionData(), которая будет принимать в себя почту и пароль и выполнять сохранение данных в локальное хранилище данных. Инициализируем объект SharedPreferences в переменную prefs, после чего вызовем у prefs метод setString(‘ключ’, ‘значение’) для сохранения email в локальное хранилище данных. В классе создадим объект FlutterSecureStorage в переменную \_secureStorage, после чего в функции \_saveSessionData() обратимся к \_secureStorage и вызовем его метод write(key: ‘ключ’, value: ‘значение’) для сохранения пароля в защищенное хранилище – листинг 4.

Листинг 4 – функция \_saveSessionData, создание переменных \_sessionId, \_userId, создание Flutter Secure Storage

|  |
| --- |
| class NetworkService {  final FlutterSecureStorage \_secureStorage = const FlutterSecureStorage();  // Функции  Future<void> \_saveSessionData(String email, String password) async {  final prefs = await SharedPreferences.getInstance();  await prefs.setString('email', email);  await \_secureStorage.write(key: 'password', value: password);  }  } |

В функции login – при успешной авторизации, http ответ 200, вызываем \_saveSessionData() и передаем туда email и password введенные пользователем – листинг 5.

Листинг 5 – изменения в функции login

|  |
| --- |
| if (response.statusCode == 200) {  final responseData = jsonDecode(response.body);  \_sessionId = responseData['sessionId'];  \_userId = responseData['userId'];  \_saveSessionData(email, password);  } |

Листинг 6 – Полный код login

|  |
| --- |
| 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'];  \_saveSessionData(email, password);  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.'  };  }  } |

В функции register повторяем аналогичные действия, как в функции login – листинг 7.

Листинг 7 – код функции register

|  |
| --- |
| 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'];  \_saveSessionData(email, password);  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.'  };  }  } |

Создадим функцию autoLogin. Внутри нее объявим объект SharedPreferences как переменную prefs, создадим переменную email, в которую при помощи getString(‘ключ’) установим данные из локального хранилища , создадим переменную password и при помощи обращения к secureStrorage.read(‘ключ’) установим данные из защищенного хранилища данных. После чего сделаем проверку, что данные в email и password существуют, если да, то вызываем функцию login – листинг 8.

Листинг 8 – Код autologin

|  |
| --- |
| Future<Map<String, dynamic>> autoLogin() async {  final prefs = await SharedPreferences.getInstance();  final email = prefs.getString('email');  final password = await \_secureStorage.read(key: 'password');  if (email != null && password != null) {  return await login(email, password);  } else {  return {'success': false, 'error': 'No saved credentials'};  }  } |

Обновим функцию updateUser. При успешном обновлении – если http ответ 200, email не пустой – то сохраним новое значение email в локальное хранилище – листинг 9.

Листинг 9 – Обновленная часть updateUser

|  |
| --- |
| if (response.statusCode == 200) {  if (email != null) {  final prefs = await SharedPreferences.getInstance();  await prefs.setString('email', email);  }  } |

Листинг 10 – Полный код updateUser

|  |
| --- |
| Future<Map<String, dynamic>> updateUser({  String? fullName,  String? email,  dynamic photo,  }) async {  final sessionId = await \_getSessionId();  if (sessionId == null) {  return {'success': false, 'error': 'No session ID found'};  }  final url = Uri.parse('$baseUrl/updateUser');  final request = http.MultipartRequest('POST', url);  request.headers['Authorization'] = 'Bearer $sessionId';  if (fullName != null) {  request.fields['fullName'] = fullName;  }  if (email != null) {  request.fields['email'] = email;  }  if (photo != null) {  if (photo is File) {  final mimeType = lookupMimeType(photo.path);  final mimeTypeData =  mimeType != null ? mimeType.split('/') : ['image', 'jpeg'];  request.files.add(  await http.MultipartFile.fromPath(  'photo',  photo.path,  contentType: MediaType(mimeTypeData[0], mimeTypeData[1]),  ),  );  } else if (photo is Uint8List) {  final mimeTypeData = ['image', 'jpeg'];  request.files.add(  http.MultipartFile.fromBytes(  'photo',  photo,  contentType: MediaType(mimeTypeData[0], mimeTypeData[1]),  filename: 'uploaded\_image.jpg',  ),  );  }  }  try {  final response = await request.send();  final responseBody = await response.stream.bytesToString();  if (response.statusCode == 200) {  if (email != null) {  final prefs = await SharedPreferences.getInstance();  await prefs.setString('email', email);  }  return {  'success': true,  'message': jsonDecode(responseBody)['message']  };  } else {  return \_handleErrorResponse(  http.Response(responseBody, response.statusCode));  }  } catch (e) {  return {  'success': false,  'error': 'Something went wrong. Please try again later.'  };  }  } |

Листинг 11 – полный код network.service

|  |
| --- |
| 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';  import 'package:shared\_preferences/shared\_preferences.dart';  import 'package:flutter\_secure\_storage/flutter\_secure\_storage.dart';  class NetworkService {  final String baseUrl = 'http://localhost:8080';  final FlutterSecureStorage \_secureStorage = const FlutterSecureStorage();  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>> autoLogin() async {  final prefs = await SharedPreferences.getInstance();  final email = prefs.getString('email');  final password = await \_secureStorage.read(key: 'password');  if (email != null && password != null) {  return await login(email, password);  } else {  return {'success': false, 'error': 'No saved credentials'};  }  }  Future<Map<String, dynamic>> login(String email, String password) async {  final url = Uri.parse('$baseUrl/login');  try {  final response = await http.post(  url,  headers: {'Content-Type': 'application/x-www-form-urlencoded'},  body: {'email': email, 'password': password},  );  if (response.statusCode == 200) {  final responseData = jsonDecode(response.body);  \_sessionId = responseData['sessionId'];  \_userId = responseData['userId'];  \_saveSessionData(email, password);  return {  'success': true,  'message': responseData['message'],  'userId': \_userId,  'fullName': responseData['fullName'],  'sessionId': \_sessionId,  };  } else {  return \_handleErrorResponse(response);  }  } catch (e) {  return {  'success': false,  'error': 'Something went wrong. Please try again later.'  };  }  }  Future<Map<String, dynamic>> register(  String fullName,  String email,  String password,  String confirmPassword,  ) async {  final url = Uri.parse('$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'];  \_saveSessionData(email, password);  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 $sessionId';  if (fullName != null) {  request.fields['fullName'] = fullName;  }  if (email != null) {  request.fields['email'] = email;  }  if (photo != null) {  if (photo is File) {  final mimeType = lookupMimeType(photo.path);  final mimeTypeData =  mimeType != null ? mimeType.split('/') : ['image', 'jpeg'];  request.files.add(  await http.MultipartFile.fromPath(  'photo',  photo.path,  contentType: MediaType(mimeTypeData[0], mimeTypeData[1]),  ),  );  } else if (photo is Uint8List) {  final mimeTypeData = ['image', 'jpeg'];  request.files.add(  http.MultipartFile.fromBytes(  'photo',  photo,  contentType: MediaType(mimeTypeData[0], mimeTypeData[1]),  filename: 'uploaded\_image.jpg',  ),  );  }  }  try {  final response = await request.send();  final responseBody = await response.stream.bytesToString();  if (response.statusCode == 200) {  if (email != null) {  final prefs = await SharedPreferences.getInstance();  await prefs.setString('email', email);  }  return {  'success': true,  'message': jsonDecode(responseBody)['message']  };  } else {  return \_handleErrorResponse(  http.Response(responseBody, response.statusCode));  }  } catch (e) {  return {  'success': false,  'error': 'Something went wrong. Please try again later.'  };  }  }  Future<List<Map<String, dynamic>>> getPopularCars() async {  return \_getWithSession('$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> \_saveSessionData(String email, String password) async {  final prefs = await SharedPreferences.getInstance();  await prefs.setString('email', email);  await \_secureStorage.write(key: 'password', value: password);  }  Future<void> \_clearSessionData() async {  final prefs = await SharedPreferences.getInstance();  await prefs.remove('email');  await \_secureStorage.delete(key: 'password');  \_sessionId = null;  \_userId = null;  }  Map<String, dynamic> \_handleErrorResponse(http.Response response) {  final errorData = jsonDecode(response.body);  return {'success': false, 'error': errorData['error'] ?? 'Request failed'};  }  } |

На экране loadingScreen добавим функцию \_attempAutoLogin() которая будет выполнять авторизацию пользователя, если есть данные в локальном хранилище и переводить пользователя сразу на главную страницу – иначе на экран авторизации – листинг 12.

Листинг 12 – функция \_attempAutoLogin

|  |
| --- |
| Future<void> \_attemptAutoLogin() async {  final result = await networkService.autoLogin();  Future.delayed(const Duration(seconds: 3), () {  if (result['success']) {  Navigator.pushReplacement(  context,  MaterialPageRoute(  builder: (context) => const CustomBottomNavigationBar()),  );  } else {  Navigator.pushReplacement(  context,  MaterialPageRoute(builder: (context) => const LoginPage()),  );  }  });  } |

Листинг 13 – Полный код LoadingScreen

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