ПРАВИТЕЛЬСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»

Факультет компьютерных наук
Департамент программной инженерии

СОГЛАСОВАНО
Научный руководитель,
старший преподаватель департамента
программной инженерии

_____ М.К. Горденко
«___» _____ 2021 г.

УТВЕРЖДАЮ
Академический руководитель
образовательной программы
«Программная инженерия»

_____ В. В. Шилов
«___» _____ 2021 г.

Инв. № подл	Подп. и дата	Взам. инв. №	Инв. № дубл.	Подп. и дата
RU.17701729.04.05 -01 81				

«HSE COFFEE» - КЛИЕНТ-СЕРВЕРНОЕ ПРИЛОЖЕНИЕ ДЛЯ ЗНАКОМСТВ НА ANDROID & IOS

Пояснительная записка

ЛИСТ УТВЕРЖДЕНИЯ

RU.17701729.04.05-01 81 01-1-ЛУ

Исполнитель:
студент группы БПИ194
_____ / А. С. Романюк /
«____» _____ 2021 г.

Москва 2021

УТВЕРЖДЕН
RU.17701729.04.05-01 81 01-1-ЛУ

Пояснительная записка

RU.17701729.04.05-01 81 01–1

Листов 57

Инв. № подл	Подп. и дата	Взам. инв. №	Инв. № дубл.	Подп. и дата
RU.17701729.04.05 -01 81				

Москва 2021

СОДЕРЖАНИЕ

АННОТАЦИЯ

В данном программном документе приведена пояснительная записка к программе ««HSE Coffee» – клиент-серверное приложение для знакомств на Android & iOS»

В разделе «Введение» указано наименование программы, краткое наименование программы и документы, на основании которых ведется разработка.

В разделе «Назначение и область применения» указано функциональное назначение программы, эксплуатационное назначение программы и краткая характеристика области применения программы.

В разделе «Технические характеристики» содержатся следующие подразделы:

1) постановка задачи на разработку программы;
2) описание алгоритма и функционирования программы с обоснованием выбора схемы алгоритма решения задачи и возможные взаимодействия программы с другими программами;
3) описание и обоснование выбора метода организации входных и выходных данных;
4) описание и обоснование выбора состава технических и программных средств.

В разделе «Ожидаемые технико-экономические показатели» указана предполагаемая потребность и экономические преимущества разработки по сравнению с отечественными и зарубежными образцами или аналогами.

Настоящий документ разработан в соответствии с требованиями:

1) ГОСТ 19.101–77 Виды программ и программных документов [1];
2) ГОСТ 19.102–77 Стадии разработки [2];
3) ГОСТ 19.103–77 Обозначения программ и программных документов [3];
4) ГОСТ 19.104–78 Основные надписи [4];
5) ГОСТ 19.105–78 Общие требования к программным документам [5];
6) ГОСТ 19.106–78 Требования к программным документам, выполненным печатным способом [6];
7) ГОСТ 19.404–79 Пояснительная записка. Требования к содержанию и оформлению [7].

Изменения к Пояснительной записке оформляются согласно ГОСТ 19.603–78 [8], ГОСТ 19.604–78 [9].

1. ВВЕДЕНИЕ

1.1. Наименование программы

Наименование: ««HSE Coffee» – клиент-серверное приложение для знакомств на Android & iOS».

Наименование на английском языке: ««HSE Coffee» – The Client-Server Application For Acquaintances On Android & iOS».

1.2. Документы, на основании которых ведется разработка

Основанием для разработки является учебный план подготовки бакалавров по направлению 09.03.04 "Программная инженерия" и утвержденная академическим руководителем тема курсового проекта.

2. НАЗНАЧЕНИЕ И ОБЛАСТЬ ПРИМЕНЕНИЯ

2.1. Назначение программы

2.1.1. Функциональное назначение

Функциональным назначением программы является случайный поиск собеседников (других пользователей ПО), которые являются студентами НИУ ВШЭ, и обмен с ними контактными данными.

Основная цель приложения – предоставлять пользователю подбор подходящего под выбранные критерии собеседника, с которым он может обменяться контактами для назначения личных или дистанционных встреч или переговоров.

2.1.2. Эксплуатационное назначение

Решение поиска новых знакомств в рамках университета на сегодняшний день является наиболее востребованным связи с эпидемиологической ситуацией, когда студенты проводят свой учебный процесс дистанционно, что является следствием сужения круга общения со студентами с других образовательных программ.

Мобильное приложение поможет пользователям найти новые знакомства среди студентов других образовательных программ университета.

Конечными пользователями могут быть только студенты и преподаватели НИУ ВШЭ.

2.2. Краткая характеристика области применения

Краткая характеристика области применения: ««HSE Coffee» – The Client-Server Application For Acquaintances On Android & iOS» - мобильное приложение на iOS и Android, предназначающееся для поиска новых знакомств среди студентов «НИУ ВШЭ».

3. ТЕХНИЧЕСКИЕ ХАРАКТЕРИСТИКИ

3.1. Постановка задачи на разработку программы

Необходимо разработать кроссплатформенное клиент-серверное приложение на Android и iOS.

3.1.1. Задачи работы (Android и iOS клиент)

1) Реализовать взаимодействие с серверной частью посредством HTTP-запросов;
2) Реализовать получение, хранение и обновление JavaScript Web Tokens;
3) Реализовать корректное отображение полученных данных;
4) Реализовать выбор изображения с локального хранилища и его редактирование: обрезка и масштабирование;
5) Реализовать загрузку изображений на сервер посредством POST запроса.
6) Реализовать возможность ввода текстовых данных с последующей валидацией.

3.1.2. Задачи работы (серверная часть)

1) Реализовать алгоритм случайного нахождения собеседника с применением фильтрации;
2) Реализовать взаимодействие с базой данных для получения и хранения пользователей, встреч, токенов;
3) Реализовать отправку email-сообщений на электронные адреса с помощью протокола SMTP;
4) Реализовать генерацию и валидацию JavaScript Web Tokens;
5) Реализовать получение, а также хранение на локальном хранилище изображений в формате PNG, JPG, JPEG.

3.2. Описание алгоритма и функционирования программы

3.2.1. Описание построения мобильного приложения

3.2.1.1. Структура мобильного приложения

1) Загрузочная страница

Рисунок 1 – Старотовая страница.

При открытии приложения отображается стартовая, загрузочная страница. В это время приложение проверяет наличие refresh-токена и email адреса в локальном защищённом хранилище.

В случае, если refresh-токен или email отсутствуют, приложение переводит пользователя на страницу №2.

В случае, если необходимые параметры присутствует, приложение выполняет POST запрос (“/api/refresh”) с типом содержимого “application/x-www-form/urlencoded”, где в формате ключ-значение передаются следующие параметры: refresh-токен, email адрес, уникальный идентификатор устройства. Дальше в случае, если код ответа от сервера равен 200 – приложение переходит на страницу ввода информации, если у пользователя есть незаполненные данные, если их нет – на главную страницу. В противное случае – на страницу авторизации.

2) Авторизация

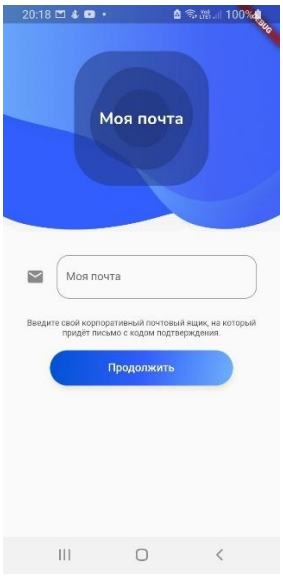


Рисунок 2 – Страница ввода Email.

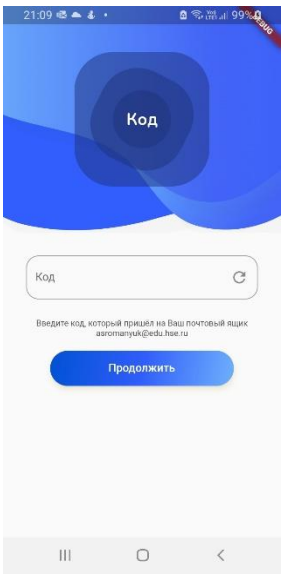


Рисунок 3 – Ввод кода подтверждения.

На странице №2 после нажатия кнопки «Продолжить» происходит валидация текстового поля. Валидация происходит по простому шаблону: строка должна заканчиваться на “@edu.hse.ru”, либо на “@hse.ru” и содержать более трёх символов до последнего символа “@”.

В случае успешной валидации текстового поля – отправляется POST запрос (“api/code?email=$email”), где вместо “$email” прописывается email-адрес, указанный в текстовом поле. А дальше в случае полученного от сервера ответа с кодом 200 – страница переключается на страницу с вводом кода подтверждения.

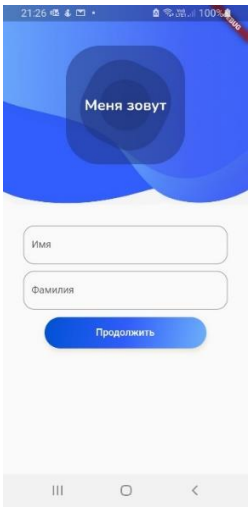
На странице №3 содержимое текстового поля также проходит валидацию: так как предполагается ввод шестизначного кода, то валидация считается выполненной, если содержимое – это строка, состоящая из 6-ти цифр.

Приложение совершает POST запрос (“api/confirm”) с телом запроса в формате ключ-значение, где указываются три значения: email, код и уникальный идентификатор устройства.

В случае успешного ответа – от сервера поступит сообщение в формате JSON, содержащее данные в формате ключ-значение из access-токена и refresh-токена.

После получения токенов устройство сохраняет их и email пользователя в защищённое хранилище.

3) Ввод необходимой информации



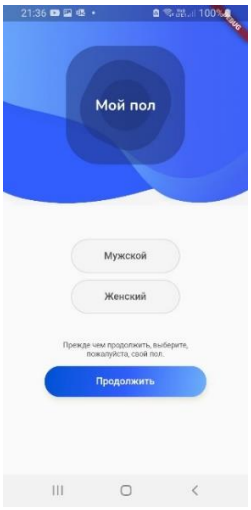


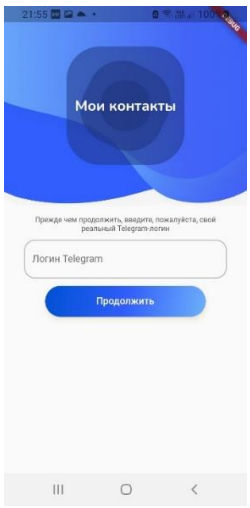
Рисунок 4 – Страница ввода имени и фамилии. *Рисунок 5 – Страница выбора пола.*

После подтверждения кода устройство запрашивает данные о пользователе с помощью HTTP-GET запроса к серверу (“/api/user/settings/{token}”), где вместо “{token}” указывается access-токен. От сервера ожидается получение ответа в формате JSON, в котором содержится информация о пользователе.

На основе полученной информации приложение отображает только те страницы заполнения информации, которые нуждаются в заполнении. Так, например, если у пользователя уже ранее были указаны имя и фамилия, вместо страницы №4 сразу отобразится страница №5.

```
 if (user.firstName == null || user.lastName == null || user.firstName.isEmpty ||
     user.lastName.isEmpty) {
     Navigator.of(context).pushReplacementNamed(AuthNameScreen.routeName);
    }
    // Если не выбран пол
else if(user.gender == null || user.gender == Gender.NONE){
     Navigator.of(context).pushReplacementNamed(AuthGenderScreen.routeName);
    }
…
```

Листинг 1. Код на языке Dart.Часть кода, демонстрирующая перевод пользователя на нужную страницу в соответствии с данными.



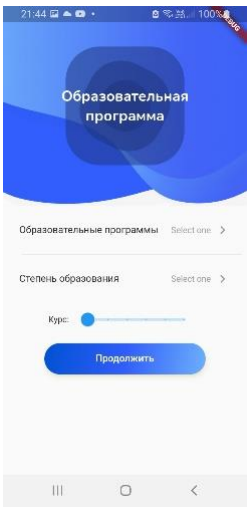


Рисунок 6 – Страница ввода Telegram – логина. *Рисунок 7 – Страница ввода обр. программы.*

После заполнения данных при нажатии на кнопку «Продолжить» происходит локальная валидация данных, если данные валидны, то приложение совершает HTTP PUT запрос к серверу (“/api/user/settings/{token}”), где вместо “{token}” указывается access-токен, а в теле запроса передаётся десериализованный в JSON формат экземпляра класса User. Следующая страничка отображается только в том случае, если от код ответа от сервера равен 200.

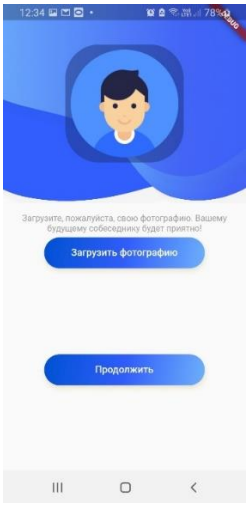


Рисунок 8 – Страница загрузки изображения.

После выбора фотографии пользователем и при нажатии на кнопку «Продолжить» будет выполнен POST запрос (“api/user/image/{token}”), где вместо “{token}” указывается access-токен. В теле запроса с содержимым в формате “multipart/form-data” указывается выбранная пользователем фотография с ключом “image”.

```
dio.FormData formData = new dio.FormData.fromMap(
   {"image": await dio.MultipartFile.fromFile(image.path)});

final response = await dio.Dio()
   .post("$_Ip/api/user/image/$accessToken", data: formData);
```

Листинг 2. Код на языке Dart.

Для выполнения запроса использовалась библиотека Dio (https://pub.dev/packages/dio) для удобного взаимодействия с Multipart (https://tools.ietf.org/html/rfc7578) форматом, которого не предоставляет стандартная библиотека Dart Http (https://pub.dev/packages/http).

4) Главный экран

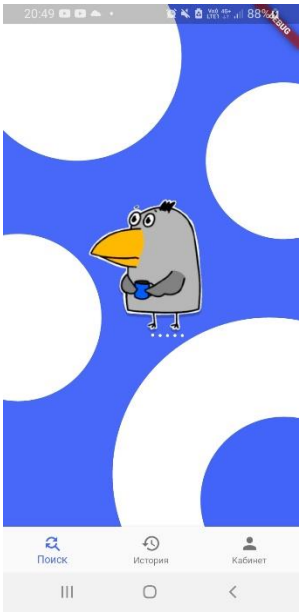
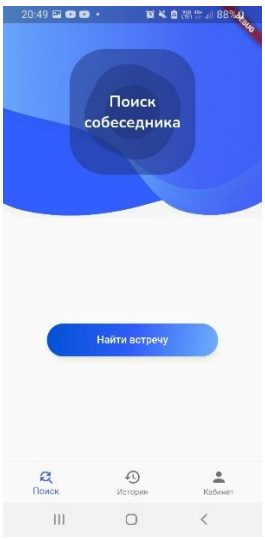


Рисунок 9 - Страница загрузки встречи.

При открытии страницы «Поиск» будет выполнен HTTP-GET запрос (“api/meet/{token}”), ответом от сервера послужит ответ в текстовом формате JSON, содержащий данные о текущей встрече. В соответствии с этими данными отображается нужный экран: экран для начала поиска, экран ожидания поиска, экран встречи.



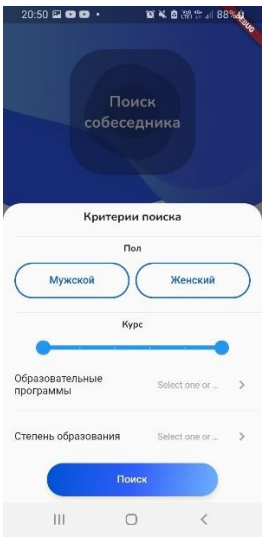


Рисунок 10 – Страница поиска встреч. *Рисунок 11 – Страница выбора критериев поиска.*

Страницы №10–№11 – страницы поиска встречи. При нажатии на кнопку «Найти встречу» происходит открытие компонента BottomSheet, предоставляемый библиотекой Flutter Material (https://flutter.dev/docs/development/ui/widgets/material).

При нажатии на кнопку «Поиск» происходит валидация на выбранные элементы фильтрации: образовательные программы и степень образования. Валидация считается успешной в случае, если выбрана хотя бы одна образовательная программа и хотя бы одна степень образования.

После успешной валидации приложение совершает HTTP-POST запрос (/api/search/{token}) с телом, в котором в формате «ключ-значение» представлены данные для фильтрации: список полов, список факультетов, список степеней образования, целое значение минимального курса, целое значение максимального курса.

Ответом от сервера служит информация о встрече, эта встреча может быть активной или быть в состоянии поиска. Активной встреча может быть в том случае, если пользователь начал поиск встречи и ему сразу же подобрался оппонент, в этом случае ему сразу отобразится информация о встрече. В противном случае – отобразится страница с ожиданием встрече.

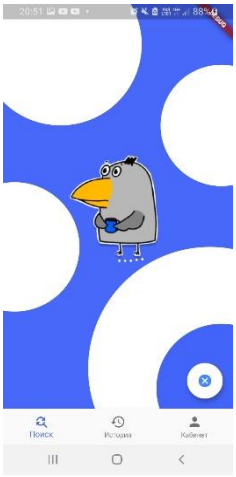


Рисунок 12 – Страница ожидания поиска встречи.

При ожидании встречи каждые 15 секунд приложение выполняет HTTP-GET запросы о состоянии встречи (описание этого запроса описаны ранее). Так как пользователю не требуется получить данную информацию быстро, то технология «long-polling» и технология «сокетного соединения» избыточны и поэтому не были реализованы.

При нажатии на кнопку «Отмена встречи» приложение выполнит HTTP-DELETE запрос (/api/meet/{token}). В случае если код ответа от сервера будет равен 200 – поиск завершится и вернётся страница с поиском встречи.

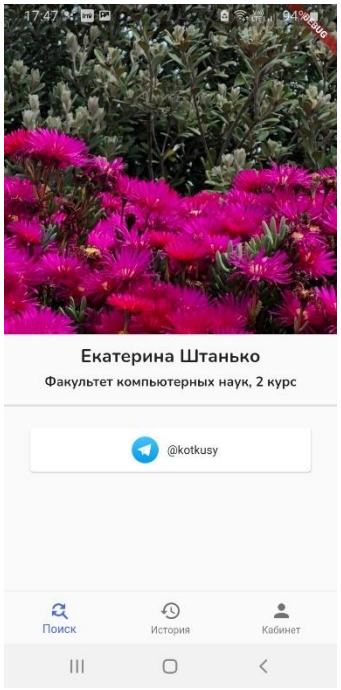
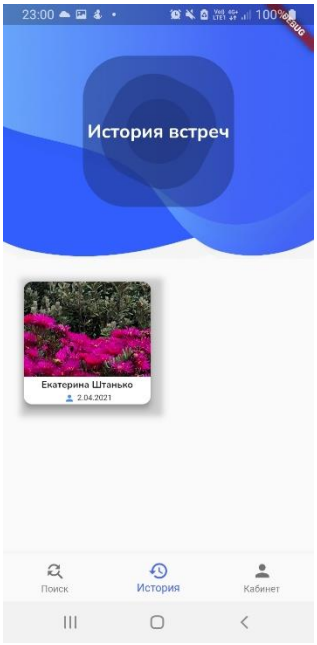


Рисунок 13 – Страница просмотра активной встречи.

Состояние встречи содержит и дату её окончания, поэтому при просмотре встречи запросы на сервер выполнятся не будут. Устанавливается таймер на разницу текущего времени и дату окончания встречи в миллисекундах, и после завершения таймера страница просмотра встречи сменится на страницу для поиска новой встречи.

```
if (value.getData().meetStatus == MeetStatus.ACTIVE)
{
 Timer(
    Duration(
       milliseconds:
          value.getData().expiresDate.microsecond -
             DateTime.now().millisecond), () {
     _updByMeet()
 })
}
```

Листинг 3. Код на языке Dart.



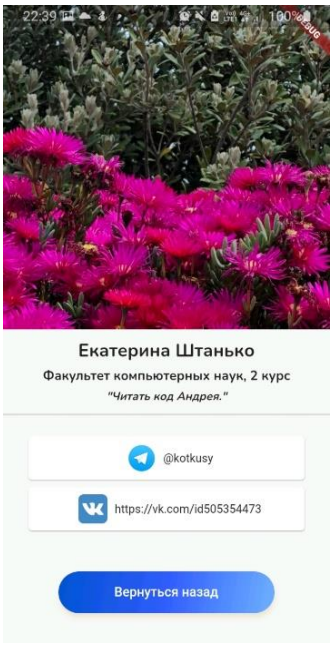


Рисунок 14 – Страница истории встреч. *Рисунок 15 – Страница просмотра встречи.*

Для получения оконченных встреч выполняется HTTP-GET запрос (“/api/meets/{token}”), ответом от сервера послужит ответ в текстовом формате JSON, содержащий данные о каждой законченной встрече. При открытии встречи по нажатию на карточку дополнительный запрос не выполняется.

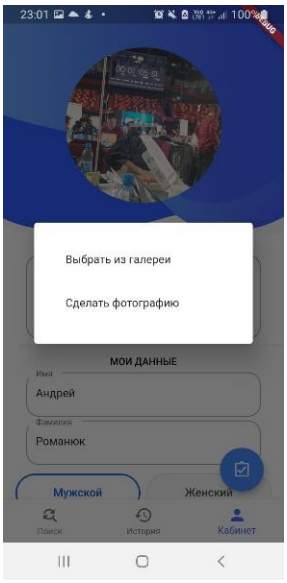


Рисунок 16 – Страница редактирования информации. *Рисунок 17 – Диалоговое окно о загрузке.*

Страница редактирования настроек пользователя идентична с обновлением информацией при первом заполнении. При нажатии пользователем на кнопку «Подтвердить изменения» выполняется HTTP-PUT запрос (“/api/user/settings/{token}”) для обновления информации о пользователе.

При загрузке новой фотографии выполняется запрос HTTP-POST (“api/user/image/{token}”). Более подробная информация об этих запросах описана ранее.

3.2.1.2. Получение изображения из локального хранилища и его обработка

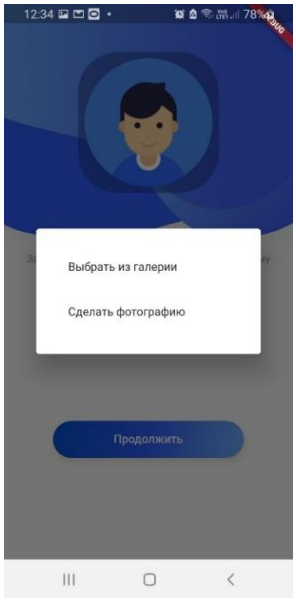


Рисунок 18 – Диалоговое окно о выборе метода загрузки. *Рисунок 19 – Экран редактирования изображения.*

Получение изображения происходит прямиком из локального хранилища, но также дополнительно поддерживается получение изображения средствами камеры при наличие аппаратной поддержки.

Для получения изображения используется библиотека Image Picker. Библиотека в зависимости от платформы обращается к внутренним компонентам системы, таких как галерея или камера, и запрашивает у них результат выполнения.

```
final picker = ImagePicker();
final pickedFile = await picker.getImage(source: source);
```

Листинг 4. Код на языке Dart.

В качестве принимаемого аргумента source требуется указать источник, для получения изображения с локального хранилища – ImageSource.gallery, а для получения изображения с камеры – ImageSource.camera.

Обработка фотографии поддерживает три метода редактирования: обрезка по соотношению сторон, поворот и масштабирование. Данный функционал предоставляет библиотека Image Cropper.

```
_image = await ImageCropper.cropImage(
        sourcePath: pickedFile.path,
        aspectRatioPresets: [
          CropAspectRatioPreset.square,
          CropAspectRatioPreset.ratio4x3,
          CropAspectRatioPreset.original
        ],
        maxWidth: 800);
```

Листинг 5. Код на языке Dart.

Метод принимает в качестве параметра sourcePath путь до фотографии, в качестве aspectRatioPresets принимает параметры уже ранее заготовленных шаблонов для выставления отношения длины к высоте фотографии, а также максимальную ширину фотографии в пикселях.

3.2.1.3. Взаимодействие с защищённым хранилищем

Взаимодействие с защищённым хранилищем производится с помощью библиотеки Flutter Secure Storage.

В хранилище данные хранятся в формате ключ-значение. В процессе работы приложения сохранению и обновлению подвергаются только три ключа: refreshToken, accessToken, email, значения которых хранятся в строковом формате.

1) refreshToken. Ключ для хранения значения refresh-токена.
2) accessToken. Ключ для хранения значения access-токена.
3) email. Ключ для хранения значения email.

3.2.1.4. Получение уникального идентификатора устройства

Получение уникального идентификатора устройства выполняется с помощью библиотеки Flutter Device Info (https://pub.dev/packages/device_info).

```
static Future<String> getFingerprint() async {
  var deviceInfo = DeviceInfoPlugin();
  if (Platform.isIOS) {
    var iosDeviceInfo = await deviceInfo.iosInfo;
    return iosDeviceInfo.identifierForVendor; // unique ID on iOS
  } else {
    var androidDeviceInfo = await deviceInfo.androidInfo;
    return androidDeviceInfo.androidId; // unique ID on Android
  }
}
```

Листинг 6. Код на языке Dart.

В соответствие с платформой (Android или iOS) у библиотеки запрашивается информация об устройстве, и от этой информации забирается уникальный идентификатор. Для iOS это identifierForVendor (UIDevice), а для Android androidId.

3.2.1.5. Обновление Access-токена.

При обращении к серверу приложение может получить код ответа равным 403 (Forbidden). Этот код ошибки указывает на истечение срока действия access-токена. В этом случае приложение на основе сохранённых в хранилище данных: email адреса и refresh-токена, а также уникального идентификатора устройства выполняет POST запрос (“api/refresh”) с телом в формате «ключ-значение».

При условии корректных данных, от сервера поступит сообщение в формате JSON, содержащее данные в формате ключ-значение из access-токена и refresh-токена. Приложение выполняет десериализацию данных и сохраняет новые токены в локальном защищенном хранилище.

3.2.2. Описание построения серверной части

3.2.2.1. Структура серверной части

Структура серверной части разделяется на три слоя:

1) Слой представления данных: Controllers;
2) Слой бизнес-логики: Service;
3) Слой доступа к данным: Repository.
4) Слой данных: Model

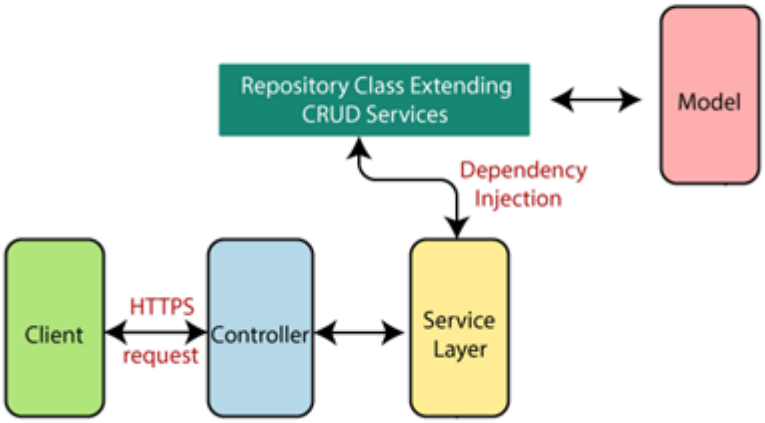


Рисунок 20. Серверная архитектура.

3.2.2.2. Генерация и валидация JavaScript Web Tokens

Генерация и валидация JavaScript Web Tokens выполняется с помощью библиотеки “io.jsonwebtoken:jjwt”. Структура JWT состоит из трёх частей: заголовка, полезной нагрузки, и подписи.

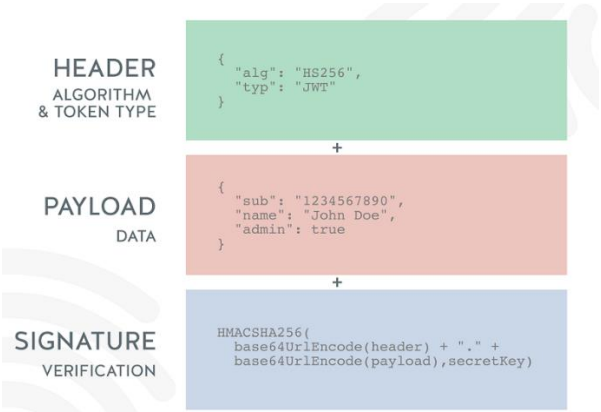


Рисунок 21. Пример структуры JWT с алгоритмом шифрования SHA256.

Технология JWT позволяет использовать алгоритмы шифрования разных семейств: HS, PS, ES, RS. Так как генерация и валидация токенов должна происходить только на сервере, семейства PS, ES, RS не подходят в силу их избыточности, так как при генерации токена для них требуются public-key и private key.

В силу этого, был выбран алгоритм шифрования HS512, он же HMAC SHA512. SHA512 – безопасный алгоритм шифрования, генерирующий хэш-сумму в 512 бит. Это даёт преимущество в безопасности по сравнению, например, с алгоритмом SHA256, однако накладывает дополнительные расходы для передачи данных, так как токен генерируется в 2 раза больше по размеру по сравнению с тем же SHA256.

```
@Value("\${jwt.key}") private val secretKey: String,
@Value("\${jwt.access.min}") private  val minutes: Long

private const val ISSUER = "HSE Coffee"

    fun createAccessToken(user: User): String {
        return Jwts.builder()
            .signWith(Keys.hmacShaKeyFor(secretKey.toByteArray()),
SignatureAlgorithm.HS512)
            .setSubject(user.email)
            .setIssuer(ISSUER)

.setExpiration(Date.from(Instant.now().plus(Duration.ofMinutes(minutes))))
            .setIssuedAt(Date.from(Instant.now()))
            .compact()
    }
```

Листинг 7. Код на языке Kotlin.

В “Payload” указывается email-адрес пользователя, дата окончания жизнеспособности токена и некая подпись. На основе этих данных и секретного ключа шифрования SHA512 библиотека генерирует токен.

При валидации токена такой информации будет достаточно: по email идентифицируется пользователь, по дате окончания определяется жизнеспособность токена. Валидация токена происходит благодаря сравнению Signature части пришедшего токена и новой подписи, сгенерированной на основе Header и Payload с тем же секретным ключом.

3.2.2.3. Отправка email-сообщений с помощью протокола SMTP

Отправка email-сообщений основывается на протоколе SMTP, взаимодействие с почтовым сервисом по данному протоколу предоставляется библиотекой “org.springframework.boot:spring-boot-starter-mail”.

Для подключения к почтовому сервису необходимо указать хост, порт, имя пользователя и пароль.

```
spring.mail.host=smtp.yandex.ru
spring.mail.port=465
spring.mail.username=admin@hsecoffee.ru
spring.mail.password=qRuaG67l1YjtMxD
spring.mail.properties.mail.smtp.auth=true
spring.mail.properties.mail.smtp.ssl.enable=true
spring.mail.default-encoding=UTF-8
```

Листинг 8. Конфигурация подключения к почтовому сервису.

Отправка email-сообщения происходит благодаря взаимодействию с экземпляром класса JavaMailSender, который Spring Boot создаёт на основе конфигурационных данных.

```
val message = SimpleMailMessage().apply{
    setSubject(subject)
    setText(text.replace("{code}", code.toString()))
    setFrom(from)
    setTo(receiver)
}

javaMailSender.send(message)
```

Листинг 9. Код на языке Kotlin.

Формирование письма также происходит на основе конфигурационных данных:

```
mail.lifetime.ms=300000
mail.from=info@hsecoffee.ru
mail.text=Hi, your HSE Coffee app code is {code}
mail.subject=Auth in HSE Coffee app
```

Листинг 10. Конфигурация контента для письма.

3.2.2.4. Взаимодействие и хранение данных в базе данных

Взаимодействие с базой данных происходит благодаря стандарту JDBC, который предоставляет абстрактный слой взаимодействия с базой данных. В качестве самой базы данных была выбрана реляционная база данных – MySQL, которая является производительной, безопасной, масштабируемой и кроссплатформенной БД.

Для взаимодействия со стандартом JDBC использовалась спецификация JPA, которая предоставляет возможность сохранять в удобном виде Java-объекты в базе данных. Для реализации этого стандарта использовалась библиотека Hibernate. Данная библиотека не только устанавливает связь объектов с базой данных, но также автоматически генерирует интерфейсы взаимодействия с ней. Это не только ускоряет процесс разработки, но и делает код безопасным.

```
@EnableJpaRepositories
@Repository("userRepository")
interface UserRepository : CrudRepository<User, Long> {
    fun findByEmail(email: String): User?

    fun existsUserById(id : Long): Boolean
}
```

Листинг 11. Код на языке Kotlin. Интерфейс взаимодействия с таблицей User.

Таким образом, всё взаимодействие с базой данных основывается на описании интерфейсов взаимодействия и описании структуры объектов, хранимых в виде таблицы.

```
@Entity
data class RefreshToken(
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "id")
    val id: Long,

    @OneToOne(targetEntity = User::class, fetch = FetchType.EAGER)
    @JoinColumn(nullable = false, name = "user_id")
    val user: User,
    @Column(name = "uuid")
    val uuid: UUID = UUID.randomUUID(),
    @Column(name = "fingerprint")
    val fingerprint: String,
…
```

Листинг 12. Код на языке Kotlin. Объект, хранимый в БД в формате таблицы.

3.2.2.5. Получение и хранение изображений

Хранение изображений осуществляется на локальном хранилище. Такое решение было принято для того, чтобы можно было в любое время получить доступ к ресурсам с помощью одного только веб-сервера.

После получения через REST контроллер объекта типа MultipartFile, он проходит валидацию и сохраняется на локальном хранилище с помощью библиотеки Java NIO, которая предназначена для высокопроизводительных операций ввода-вывода.

Для организации просмотра изображений требуется конфигурация веб-сервера Nginx, который перенаправляет запросы извне на ресурсы локальной системы, в частости при запросе к /api по HTTP соединению веб сервер перенаправляет запрос на сам сервер, работающий на порту 8080, а при запросе /uploads веб сервер перенаправляет запрос на локальное хранилище /home/server/uploads/, где и хранятся изображения.

```
server {
  listen       80;
    server_name   example.com *.example.com;
    location /api {
        proxy_pass         http://localhost:8080;
        proxy_http_version 1.1;
        proxy_set_header   Upgrade $http_upgrade;
        proxy_set_header   Connection keep-alive;
        proxy_set_header   Host $host;
        proxy_cache_bypass $http_upgrade;
        proxy_set_header   X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header   X-Forwarded-Proto $scheme;
    }
    location /uploads {
        include  /etc/nginx/mime.types;
        root /home/server;
    }
}
```

Листинг 13. /etc/nginx/sites-available/default. Файл конфигурации Nginx.

3.2.2.6. Алгоритм случайного нахождения собеседника с применением фильтрации

Пользователь, который выполнил запрос на нахождение собеседника, передал необходимые фильтры. Фильтром называются следующие элементы:

1) Список образовательных программ;
2) Список степеней образования;
3) Минимальный курс;

4) Максимальный курс;
5) Список полов.

Алгоритм получает всех пользователей в базе данных, которые ранее выполнили запрос на нахождения собеседника, но ещё не нашли его. Данный список пользователей отсортирован в порядке выполнения запроса на поиск, то есть чем раньше пользователь сделал такой запрос, тем ближе он находится к началу такого списка.

Обозначим пользователя, который совершает запрос на поиск как Пользователь 1, а пользователя, который ранее выполнил поиск собеседника, но ещё не нашёл его – Пользователь 2.

Тогда алгоритм проходится по отсортированному списку и находит первого такого Пользователя 2, который выполняет следующие критерии:

1) Элементы фильтрации Пользователя 1 подходят под Пользователя 2
2) Элементы фильтрации Пользователя 2 подходят под Пользователя 1

То есть подбирается такой Пользователь 2, который не только удовлетворяет фильтрам Пользователя 1, но ещё и сам Пользователь 1 удовлетворяет фильтрам Пользователя 2.

Если такой Пользователь 2 нашёлся, то он удаляется из таблицы пользователей, которые находятся на стадии поиска встречи.

Если такого Пользователя 2 не нашлось, пользовать 1 помещается в конец таблицы пользователей, которые находятся на стадии поиска встречи.

В целом алгоритм не является случайным, однако с точки зрения пользователя, который совершил запрос на поиск собеседника, его можно таковым назвать, так как пользователь не может знать о всех других пользователях, которые также выполняют поиск.

3.3. Описание и обоснование выбора метода взаимодействия клиента и сервера

3.3.1. Описание метода организации взаимодействия клиента и сервера

Взаимодействие между клиентской и серверной частями осуществляется посредством HTTP-запросов. При получении GET, POST, PUT, DELETE запросов от клиента сервер отвечает сообщением в формате JSON (JavaScript Object Notation). К содержимому может относиться: информация о пользователе, встречи, параметрах фильтрации, статуса встречи.

Также к взаимодействию относится передача изображений методом POST протокола HTTP с типом содержимого Multipart/form-data.

3.3.2. Обоснования выбора метода организации взаимодействия клиента и сервера

Форматом общения между клиентом и сервером был выбран формат JSON, однако возможно было реализовать обмен текстовыми форматом XML, либо бинарным форматом.

Формат общения между клиентом и сервером был выбран JSON, а не его похожий аналог – XML по следующей причине:

1) Размер документа формата JSON значительно меньше, а значит повышается скорость передачи данных.

Также формат общения между клиентом и сервером был выбран текстовый, а не бинарный по следующей причине:

1) Текстовый формат более читаемый, что ускоряет процесс разработки.

3.4. Описание и обоснование выбора состава технических и программных средств

3.4.1. Состав технических и программных средств

Технические требования сервера:

1) Операционная система
    1. Windows
        1. Windows 10 (8u51 и выше).
        2. Windows 8.x.
        3. Windows 7.
        4. Windows Vista.
        5. Windows Server 2008 R2 x64.
        6. Windows Server 2012 и 2012 R2.
    2. Mac OS X
        1. Mac OS X 10.8.3+, 10.9 +.
    3. Linux
        1. Oracle Linux 5.5+.
        2. Oracle Linux 6.x.
        3. Oracle Linux 7.x.
        4. Red Hat Enterprise Linux 5.5+, 6.x.
        5. Red Hat Enterprise Linux 7.x.
        6. Suse Hat Enterprise Server 10 SP2+, 11.x.

7. Suse Hat Enterprise Server 12.x.
8. Ubuntu Linux 12.04 LTS, 13.x.
9. Ubuntu Linux 14.x.
10. Ubuntu Linux 15.04.
11. Ubuntu Linux 15.10

2) Процессор: Одноядерный процессор с тактовой частотой от 2.4 ГГц.
3) Пространство на диске: от 12 ГБ.
4) RAM: от 1536 МБ.
5) Наличие Java JDK 11, MySQL Server 8.0 и выше, Nginx 1.19 и выше.

Технические требования мобильного приложения:

1) Операционная система мобильного устройства должна бывать Android версии 6.0 «Marshmallow» и выше или iOS версии 10.3.3 и выше.
2) Постоянное подключение к сети интернет;
3) Для использования кнопок и полей для ввода требуется сенсорный экран;
4) Память устройства должна быть не менее 80 Мб (рекомендуется более 120 Мб).

3.4.2. Обоснование выбора технических и программных средств

1. Серверная часть:

Чтобы поддерживать высокую производительность и стабильность серверной части – используется последняя версия Spring Boot на платформе JVM.

Наличие MySQL Server необходимо, чтобы создавать и работать с базой данных. При этом выбор MySQL аргументирован так:

1) Поддержка SQL
2) Поддержка многих операционных систем**.**
3) Высокая скорость и производительность**.**

Наличие веб-сервера необходимо, чтобы ретранслировать запросы пользователей из внешней сети на внутреннюю сеть.

2. Android приложение:

Приложение разрабатывалось под минимальную версию операционной системы Android версии 6.0 «Marshmallow», потому что согласно официальному отчёту компании Google от 11 апреля 2021 года устройств с операционной системой 6.0 и выше – 84.9% от общего количества.

3. iOS приложение:

Приложение разрабатывалось под минимальную версию операционной системы iOS версии 10.3.3, потому что согласно официальному отчёту издания David Smith от 11 апреля 2021 года устройств с операционной системой 10.3.0 и выше – 98% от общего количества.

4. ОЖИДАЕМЫЕ ТЕХНИКО-ЭКОНОМИЧЕСКИЕ ПОКАЗАТЕЛИ

4.1. Ориентировочная экономическая эффективность

В рамках данной работы расчет экономической эффективности не предусмотрен.

4.2. Предполагаемая потребность

Данное приложение будет интересно студентам «НИУ ВШЭ», которые желают приобрести новые знакомства.

4.3. Экономические преимущества разработки по сравнению с отечественными и зарубежными образцами или аналогами

Существует как отечественные, так и зарубежные аналоги, однако данное приложение имеет следующие преимущества:

1) Распространяется бесплатно;
2) Не требует вложения денежных средств во время использования;
3) Не имеет рекламных баннеров;
4) Имеет неограниченный срок службы
5) Является частью внутриуниверситетской жизни.

5. СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1) ГОСТ 19.101-77 Виды программ и программных документов. //Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.
2) ГОСТ 19.102-77 Стадии разработки. //Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.
3) ГОСТ 19.103-77 Обозначения программ и программных документов. //Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.
4) ГОСТ 19.104-78 Основные надписи. //Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.
5) ГОСТ 19.105-78 Общие требования к программным документам. //Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.
6) ГОСТ 19.106-78 Требования к программным документам, выполненным печатным способом. //Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.
7) ГОСТ 19.404-79 Пояснительная записка. Требования к содержанию и оформлению. //Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.
8) ГОСТ 19.603-78 Общие правила внесения изменений. //Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.
9) ГОСТ 19.604-78 Правила внесения изменений в программные документы, выполненные печатным способом. //Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.
10) Системные требования Java [Электронный ресурс]// URL https://www.java.com/ru/download/help/sysreq.html (Дата обращения: 02.04.2021, режим доступа: свободный).
11) Системные требования MySQL [Электронный ресурс]// URL https://www.cs-cart.ru/docs/4.1.x/manager/install/requirements/ (Дата обращения: 02.04.2021, режим доступа: свободный).
12) Flutter. System requirements [Электронный ресурс]// URL https://flutter.dev/docs/get-started/install/windows (Дата обращения: 04.04.2021, режим доступа: свободный).
13) Системные требования для iOS устройств [Электронный ресурс]// URL https://support.apple.com/ru-ru/HT204230 (Дата обращения: 09.11.2020, режим доступа: свободный).
14) Требования к системе для Spring [Электронный ресурс]// URL: https://java-ru-blog.blogspot.com/2020/02/spring-boot-features.html (Дата обращения: 09.11.2020, режим доступа: свободный).
15) Wikipedia. Flutter (SDK) [Электронный ресурс]// URL: https://ru.wikipedia.org/wiki/Flutter_(SDK) (Дата обращения: 04.04.2021, режим доступа: свободный).
16) Wikipedia. JAR [Электронный ресурс]// URL: https://ru.wikipedia.org/wiki/JAR (Дата обращения: 04.04.2021, режим доступа: свободный).

17) Wikipedia. SMTP [Электронный ресурс]// URL: https://ru.wikipedia.org/wiki/SMTP (Дата обращения: 04.04.2021, режим доступа: свободный).
18) Multipart/form-data [Электронный ресурс]// URL: https://ru.wikipedia.org/wiki/Multipart/form-data (Дата обращения: 04.04.2021, режим доступа: свободный).
19) Сокетное соединение по протоколу TCP/IP [Электронный ресурс]// URL: http://crypto.pp.ua/2010/06/soketnye-soedineniya-po-protokolu-tcpip-v-java/ (Дата обращения 11.04.2021, режим доступа свободный).
20) iOS Version Stats [Электронный ресурс]// URL: https://david-smith.org/iosversionstats/ (Дата обращения 11.04.2021, режим доступа свободный).

ПРИЛОЖЕНИЕ 1
ТЕРМИНОЛОГИЯ

1. **Access token –** кратковременный токен для идентификации подлинности пользователя.
2. **Fingerprint –** уникальный идентификатор устройства.
3. **Flutter** - SDK с открытым исходным кодом для создания мобильных приложений
4. **HMAC** (англ. *hash-based message authentication code* – «код аутентификации») – механизм проверки целостности информации.
5. **HTTP** (англ. *HyperText Transfer Protocol* — «протокол передачи гипертекста») — протокол прикладного уровня передачи данных, изначально — в виде гипертекстовых документов в формате HTML, в настоящее время используется для передачи произвольных данных.
6. **JAR** (англ. *Java Archive)* – это Java-архив. Представляет собой ZIP-архив, в котором содержится часть программы на языке Java.
7. **JDBC –** платформенно-независимый стандарт взаимодействия Java-приложений с различными СУБД.
8. **JPA –** технология, обеспечивающая объектно-реляционное отображение простых Java объектов и предоставляющая API для сохранения, получения и управления такими объектами.
9. **JSON (**англ. *JavaScript Object Notation*) — Текстовый формат обмена данными.
10. **JWT** (англ. *JSON Web Token*) — это открытый стандарт (RFC 7519) для создания токенов доступа, основанный на формате JSON.
11. **Long polling** – технология, которая позволяет получать данные о новых событиях с помощью «длинных запросов». Сервер получает запрос, но отправляет ответ на него не сразу, а лишь тогда, когда произойдёт какое-либо событие.
12. **Multipart/form-data** - это составной тип содержимого, чаще всего использующийся для отправки HTML-форм с бинарными (не-ASCII) данными методом POST протокола HTTP.
13. **Refresh Token –** долговременный токен, используется для получения новой пары access/refresh токенов.
14. **SMTP** (англ. *Simple Mail Transfer Protocol* — простой протокол передачи почты) — это широко используемый сетевой протокол, предназначенный для передачи электронной почты в сетях TCP/IP
15. **Spring Framework** (или коротко **Spring**) — универсальный серверный фреймворк с открытым исходным кодом для Java-платформы.
16. **База данных** (сокр. БД) — совокупность данных, хранимых в соответствии со схемой данных, манипулирование которыми выполняют в соответствии с правилами средств моделирования данных.

17. **Веб-сервер** — сервер, принимающий HTTP-запросы от клиентов и выдающий им HTTP-ответы.
18. **Десериализация –** восстановление первоначального состояния структуры данных из текстовой/битовой последовательности.
19. **Сокет** (англ. *socket* — разъём) - название программного интерфейса для обеспечения непрерывного обмена данными между процессами.

ПРИЛОЖЕНИЕ 2
ФОРМАТЫ HTTP-ЗАПРОСОВ

Запрос	Тип запроса	Параметры	Код ответа	Ответ	Назначение
api/code	POST	email – Email адрес пользователя	200, 400	Строковый результат об операции.	Отправка кода подтверждения на указанный адрес
api/confirm	POST	email – Email адрес пользователя code – Код подтверждения fingerprint – Уникальный идентификатор устройства.	200,400, 500	Для 200: JSON из двух значений refreshToken и accessToken. Для 400, 500: сообщение об ошибке.	Подтверждение кода Ошибка 400 – неверный код подтверждения Ошибка 500 – серверная ошибка.
api/refresh	POST	email – Email адрес пользователя refreshToken – Refresh-токен fingerprint – Уникальный идентификатор	200, 400, 500	Для 200: JSON из двух значений refreshToken и accessToken. Для 400, 500: сообщение об ошибке.	Обновление токенов. Ошибка 400 – невозможно обновить токен для пользователя. Ошибка 500 – серверная ошибка.

api/meet/{token}	GET	token – access-токен	200, 401, 403, 500	Для 200: информация о встрече в формате JSON Для 401, 403, 500: сообщение об ошибке.	Получение информации о встрече. 403 – Срок действия токена истёк. 401 – Неверный токен. 500 – Серверная ошибка.
api/search/{token}	POST	token – access-токен, searchParams – параметры поиска	200, 401, 403, 500	Строков представление статуса о встрече: ACTIVE – встреча сразу нашлась, ERROR – произошла ошибка, SEARCH – поиск встречи начался.	Начать поиск встречи. 403 – Срок действия токена истёк. 401 – Неверный токен. 500 – Серверная ошибка.

api/meet/{token}	DELETE	token – access-токен	200, 401, 403, 500	Строков представление статуса о встрече: NONE – встречу отменена, ERROR – произошла ошибка, SEARCH – поиск встречи начался.	Отметить поиск встречи. 403 – Срок действия токена истёк. 401 – Неверный токен. 500 – Серверная ошибка.
api/meets/{token}	GET	token – access-токен	200, 401, 403, 500	200: Строковое представление списка из встреч в формате JSON. 401, 403, 500: Описание ошибки.	Получить список законченных встреч. 403 – Срок действия токена истёк. 401 – Неверный токен. 500 – Серверная ошибка.

api/user/settings/{token}	PUT	token – access-токен, user – текстовое представление пользователя в JSON формате.	200, 401, 403, 500	Результат выполнения запрос в текстовом формате.	Обновление данных о пользователей. 403 – Срок действия токена истёк. 401 – Неверный токен. 500 – Серверная ошибка.
api/user/settings/{token}	GET	token – access-токен	200, 401, 403, 500	200: Текстовое представление пользователя в формате JSON 401, 403, 500: Описание ошибки.	Получение текущих настроек пользователя. 403 – Срок действия токена истёк. 401 – Неверный токен. 500 – Серверная ошибка.
/api/user/image/{token}	POST	token – access-токен, image – MultipartFile	200, 401, 403, 500	200: Текстовое представление пользователя в формате JSON 401,403,500: Описание ошибки.	Загрузка изображения для пользователя. 403 – Срок действия токена истёк. 401 – Неверный токен. 500 – Серверная ошибка.

ПРИЛОЖЕНИЕ 3
ОПИСАНИЕ И ФУНКЦИОНАЛЬНОЕ НАЗНАЧЕНИЕ КЛАССОВ

Таблица 3.1

Описание и функциональное назначение классов серверной части

Класс	Назначение
AuthController	Контроллер для авторизации пользователя.
AuthService	Сервис для работы с авторизацией пользователей.
CancelStatus	Enum class. Перечисление статус-кодов для отмены встреч.
ConfirmationCode	Data-class. Email-код для подтверждения аккаунта.
ConfirmationCodeRepository	Интерфейс для описания операций для взаимодействия с таблицей хранения кодов для подтверждения.
Contact	Data-class. Контакт.
Degree	Enum class. Степень образования.
EmailService	Сервис для работы с Email, в частности с протоколом SMTP для отправки Email-сообщений.
Faculty	Enum class. Перечисление доступных факультетов НИУ ВШЭ.
Gender	Enum class. Пол пользователя.
HseCoffeeApplication	Application-сущность всего приложения.
ImageStorageRepository	Интерфейс для описания операций для взаимодействия с изображениями пользователей.
ImageStorageService	Сервис для загрузки изображений для пользователя.
JwtResponseWrapper	Data class. Предназначается для JWT, чтобы отдать контроллеру уже готовый ответ в случае ошибки.
JwtService	Сервис для работы с JavaScript Web Token.
LoginWrapper	Data class. Обёртка над результатом ответа после выполнения авторизации.
Meet	Data-class. Встреча.
MeetController	Контроллер для управления встречами авторизованных пользователей.
MeetRepository	Интерфейс для описания операций для взаимодействия с таблицей встреч.
MeetService	Сервис для работы с встречами.
MeetStatus	Enum class. Перечисление статус-кодов для состояния встречи.
RefreshToken	Data-class. Токен для обновления сессии.
RefreshTokenRepository	Интерфейс для описания операций для взаимодействия с таблицей для хранения Refresh токенов.
RefreshTokenService	Сервис для работы с Refresh токенами.
Search	Data-class. Поисковые данные.
SearchParams	Data-class. Параметры поиска встреч.
SearchRepository	Интерфейс для описания операций для взаимодействия с таблицей поиска встреч.
User	Data-class. Пользователь.
UserController	Контроллер для управления аккаунтом пользователя.
UserRepository	Интерфейс для описания операций для взаимодействия с таблицей пользователей.

Класс	Назначение
UserService	Сервис для работы с пользователями.
UserStatus	Enum class. Перечисление статус-кодов для состояния пользователя.

Таблица 3.2

Описание и функциональное назначение классов мобильной части

Класс	Назначение
Api	Класс, предоставляющий интерфейс взаимодействия с Api.
Auth	Класс, предоставляющий интерфейс для взаимодействия с локальным хранилищем и методами для работы с авторизацией.
AuthCodeScreen	Widget. Авторизация. Экран ввода кода подтверждения.
AuthContactsScreen	Widget. Авторизация. Экран ввода контактных данных.
AuthEmailScreen	Widget. Авторизация. Экран ввода email – адреса.
AuthFacultyScreen	Widget. Авторизация. Экран ввода факультета и степени образования.
AuthGenderScreen	Widget. Авторизация. Экран выбора пола.
AuthNameScreen	Widget. Авторизация. Экран ввода имени и фамилии.
AuthPhotoScreen	Widget. Авторизация. Экран для загрузки фотографии.
ButtonContinue	Widget. Кнопка «Продолжить»
CapsuleWidget	Widget. Разделитель.
Contact	Класс, описывающий сущность – Контакт.
Degree	Enum class. Степень образования.
DialogLoading	Диалоговое окно.
EducationFields	Widget. Выпадающий список.
EventWrapper	Class Wrapper. Обёртка под ответы сервера.
Faculty	Enum class. Перечисление доступных факультетов НИУ ВШЭ.
Gender	Enum class. Пол пользователя.
Header	Widget. Авторизация. Заголовочный экран – шапка.
HomeCabinetScreen	Widget. Главный экран. Экран личного кабинета.
HomeFindScreen	Widget. Главный экран. Экран поиска встречи.
HomeLoadingScreen	Widget. Главный экран. Экран загрузки поиска встречи.
HomeMeetsScreen	Widget. Главный экран. Экран встреч.
HomePersonScreen	Widget. Главный экран. Экран просмотра пользователя.
HomeScreen	Widget. Главный экран.
HseCoffeeApp	Основной класс. Точка запуска приложения.
Meet	Класс, описывающий сущность – Встреча.
MeetStatus	Enum class. Перечисление статус-кодов для состояния встречи.
RouterHelper	Вспомогательный класс-маршрутизатор.
SearchParams	Класс, описывающий сущность – Параметры встречи.
SplashPainter	Фон для SplashScreen.
SplashScreen	Widget. Загрузочный экран при старте приложения.
TextFieldWrapper	Widget. Текстовое поле с дополнительными параметрами.
ToggleButtonGender	Widget. Виджет для выбора пола.
User	Класс, описывающий сущность – Пользователь.
UserStorage	Класс предназначается для хранения экземпляра пользователя во время работы приложения.

ПРИЛОЖЕНИЕ 4
ОПИСАНИЕ И ФУНКЦИОНАЛЬНОЕ НАЗНАЧЕНИЕ ПОЛЕЙ МЕТОДОВ И СВОЙСТВ

Таблица 4.1

Описание методов и свойств класса AuthController.kt

Методы				
Имя	Модификатор доступа	Тип	Аргументы	Назначение
sendCode	public	Метод	email: String	Метод для отправки Email-кода на почту пользователя.
confirmCode	public	Метод	email: String, code: Int, fingerprint: String	Метод для подтверждения кода, присланного на почту пользователя.
refreshToken	public	Метод	email: String, refreshToken: UUID, fingerprint: String	Метод для обновления пары Refresh - Access токенов.

Свойства				
Имя	Модификатор доступа	Тип	Доступ	Назначение
logger	private	Logger	get, set	Логгер.
emailService	private	EmailService	get, set	Сервис для работы с SMTP.
userService	private	UserService	get, set	Сервис для работы с пользователями.
jwtService	private	JwtService	get, set	Сервис для работы с JWT.
refreshTokenService	private	RefreshTokenService	get, set	Сервис для работы с RefreshToken

Таблица 4.2

Описание методов и свойств класса MeetController.kt

Методы				
Имя	Модификатор доступа	Тип	Аргументы	Назначение
getMeet	public	Метод	token: String	Метод для получения текущей встречи, если её нет возвращается неизвестная встреча с [MeetStatus.NONE]
findMeet	public	Метод	token: String, searchParams: SearchParams	Метод для начала поиска встречи. Если встреча была

				уже начата - возвращается [MeetStatus.ACTIVE],
cancelSearch	public	Метод	token: String	Метод для прерывания поиска встречи.
getMeets	public	Метод	token: String	Метод для получения списка из законченный встреч. Законченной встречей считается такая встреча, у которой [MeetStatus.FINISHED]
Свойства				
Имя	Модификатор доступа	Тип	Доступ	Назначение
logger	private	Logger	get, set	Логгер.
authService	private	AuthService	get, set	Сервис для работы с авторизацией
meetService	private	MeetService	get, set	Сервис для работы с встречами.

Таблица 4.3

Описание методов и свойств класса UserController.kt

Методы				
Имя	Модификатор доступа	Тип	Аргументы	Назначение
setSettings	public	Метод	token: String, newUser: User	Метод для выставления настроек для пользователя.
getSettings	public	Метод	token: String	Метод получения текущих настроек пользователя.
setImage	public	Метод	token: String, image: MultipartFile	Метод для загрузки фотографии пользователя.
Свойства				
Имя	Модификатор доступа	Тип	Доступ	Назначение
logger	private	Logger	get, set	Логгер.

authService	private	AuthService	get, set	Сервис для работы с авторизацией
meetService	private	MeetService	get, set	Сервис для работы с встречами.
imageStorageService	private	ImageStorageService	get, set	Сервис для работы с загрузкой фотографий.

Таблица 4.4

Описание методов и свойств класса AuthService.kt

Методы				
Имя	Модификатор доступа	Тип	Аргументы	Назначение
logByToken	public	Метод	token: String	Провести авторизацию по токену.
Свойства				
Имя	Модификатор доступа	Тип	Доступ	Назначение
logger	private	Logger	get, set	Логгер.
userService	private	UserService	get, set	Сервис для работы с пользователями.
jwtService	private	JwtService	get, set	Сервис для работы с JWT.

Таблица 4.5

Описание методов и свойств класса EmailService.kt

Методы				
Имя	Модификатор доступа	Тип	Аргументы	Назначение
EmailService	public	Конструктор	private val javaMailSender: JavaMailSender, private val text: String, private val subject: String, private val from: String, private var lifeTime: Int, private val domains: String	Сервис для работы с Email, в частности с протоколом SMTP для отправки EMAIL-сообщений.
isValidCode	public	Метод	receiver: String, code: Int	Проверка на валидность кода относительно Email-адреса.
isValidMail	public	Метод	email: String?	Проверка на валидность Email-адреса.
trySendCode	public	Метод	receiver: String	Метод для генерации, создания и отправки кода на Email-адрес.

setLifeTime	public	Метод	lifeTime: Int	Ручное выставление времени.
sendCode	public	Метод	receiver: String, code: Int	Метод для отправки целочисленного кода на Email-адрес.
Свойства				
Имя	Модификатор доступа	Тип	Доступ	Назначение
logger	private	Logger	get, set	Логгер.
confirmationCodeRepository	private	ConfirmationCodeRepository	get, set	Репозиторий для кодов подтверждения.

Таблица 4.6

Описание методов и свойств класса ImageStorageService.kt

Методы				
Имя	Модификатор доступа	Тип	Аргументы	Назначение
correctFile	public	Метод	file: MultipartFile	Метод для проверки корректности изображения.
isSupportedContentType	public	Метод	contentType: String	Метод для провеки корректности типа файла.
store	public	Метод	file: MultipartFile, user: User	Метод для загрузки фотографии в корневую папку.
Свойства				
Имя	Модификатор доступа	Тип	Доступ	Назначение
logger	private	Logger	get, set	Логгер.
userService	private	UserService	get, set	Сервис для работы с пользователями.

Таблица 4.7

Описание методов и свойств класса JwtService.kt

Методы				
Имя	Модификатор доступа	Тип	Аргументы	Назначение
JwtService	public	Конструктор	val secretKey: String, val minutes: Long	Сервис для работы с Javascript Web Token.
createAccessToken	public	Метод	user: User	Метод для создания JWT токена.

validateToken	public	Метод	token: String	Метод для валидации токена средствами
getIncorrectTokenResponse	private	Метод	-	Получение Response для некорректного токена.
parseAccessToken	public	Метод	token: String	Преобразование токена из [String] в [Jws] спомощью [Jwts.parserBuilder]
getJsonTokens	public	Метод	user: User, fingerPrint: String	Получение на основе пользователя и отпечатка пару из Access и Refresh токенов в JSON формате.

Свойства				
Имя	Модификатор доступа	Тип	Доступ	Назначение
logger	private	Logger	get, set	Логгер.
refreshTokenService	private	RefreshTokenService	get, set	Сервис работы с Refresh токенами.

Таблица 4.8

Описание методов и свойств класса MeetService.kt

Методы				
Имя	Модификатор доступа	Тип	Аргументы	Назначение
getMeet	public	Метод	user: User	Метод для получения текущей встречи пользователя.
getMeets	public	Метод	user: User	Метод, который дыдаёт коллекцию из всех законченных встреч пользователя.
cancelSearch	public	Метод	user: User	Метод для отмены пользователем встречи.
searchMeet	public	Метод	user: User, searchParams: SearchParams	Метод для поиска встречи.
updateMeetStatus	private	Метод	meet: Meet	Метод для проверки законченности

				встречи. Если время действия встречи [Meet.expiresDate] вышло, то она перейдёт в статус [MeetStatus.FINISHED]
Свойства				
Имя	Модификатор доступа	Тип	Доступ	Назначение
logger	private	Logger	get, set	Логгер.
userService	private	UserService	get, set	Сервис для работы с пользователями.
meetRepository	private	MeetRepository	get, set	Репозиторий встреч.
searchRepository	private	SearchRepository	get, set	Репозиторий поиска.

Таблица 4.9

Описание методов и свойств класса RefreshTokenService.kt

Методы				
Имя	Модификатор доступа	Тип	Аргументы	Назначение
createByUser	public	Метод	user: User, fingerprint: String	Создание Refresh токена [RefreshToken] на основе пользователя и отпечатка.
isValid	public	Метод	user: User, token: UUID, fingerprint: String	Проверка на корректность Refresh токена [RefreshToken].
Свойства				
Имя	Модификатор доступа	Тип	Доступ	Назначение
logger	private	Logger	get, set	Логгер.
refreshTokenRepository	private	RefreshTokenRepository	get, set	Репозиторий для работы с Refresh токенами.

Таблица 4.10

Описание методов и свойств класса UserService.kt

Методы				
Имя	Модификатор доступа	Тип	Аргументы	Назначение

UserService	public	Конструктор	val folder: String	Сервис для работы с пользователями.
getUserByEmailOrCreate	public	Метод	email: String	Метод для получения пользователя по email. Если такого пользователя нет в базе данных, то он создаётся и сохраняется в ней.
getUserByEmail	public	Метод	email: String	Метод для поиска пользователей с помощью email адреса.
createUserByEmail	public	Метод	email: String	Создание в БД пользователя с email = [email].
changeFolderAndSave	public	Метод	user: User	Метод для смена у пользователя пути с его фотографией.
setSettings	public	Метод	oldUser: User, newUser: User	Установить настройки пользователю
save	public	Метод	user: User	Сохранение или обновление записи о пользователе в БД.
Свойства				
Имя	Модификатор доступа	Тип	Доступ	Назначение
logger	private	Logger	get, set	Логгер.
userRepository	private	UserRepository	get, set	Репозиторий для работы с пользователями.

Таблица 4.11

Описание методов и свойств интерфейса ConfirmationCodeRepository.kt

Методы				
Имя	Модификатор доступа	Тип	Аргументы	Назначение
removeConfirmationTokenByEmail	public	Метод	email: String	Удаление Confirmation Code из базы данных.
existsByEmail	public	Метод	email: String	Проверка на содержание почты в базе данных.
findByEmail	public	Метод	email: String	Поиск пользователя по почте.

Таблица 4.12

Описание методов и свойств интерфейса ImageStorageRepository.kt

Методы				
Имя	Модификатор доступа	Тип	Аргументы	Назначение
correctFile	public	Метод	file: MultipartFile	Метод для проверки корректности файла.
store	public	Метод	file: MultipartFile, user: User	Метод для сохранения файла для юзера.

Таблица 4.13

Описание методов и свойств интерфейса MeetRepository.kt

Методы				
Имя	Модификатор доступа	Тип	Аргументы	Назначение
findAllByUser1OrUser2	public	Метод	user1: User, user2: User	Поиск списка встреч по двум пользователям.
existsMeetById	public	Метод	id: Long	Проверка на наличие встречи по Id.

Таблица 4.14

Описание методов и свойств интерфейса RefreshTokenRepository.kt

Методы				
Имя	Модификатор доступа	Тип	Аргументы	Назначение
findRefreshTokenByUser	public	Метод	user: User	Поиск токена по пользователю.

Таблица 4.15

Описание методов и свойств интерфейса SearchRepository.kt

Методы				
Имя	Модификатор доступа	Тип	Аргументы	Назначение
findSearchByFinder	public	Метод	finder: User	Поиск поискового запроса по пользователю.

Таблица 4.16

Описание методов и свойств интерфейса UserRepository.kt

Методы				
Имя	Модификатор доступа	Тип	Аргументы	Назначение
findByEmail	public	Метод	email: String	Поиск пользователя по email – адресу.

existsUserById	public	Метод	id: Long	Проверка на наличие пользователя по Id.

Таблица 4.17

Описание методов и свойств класса ConfirmationCode.kt

Свойства				
Имя	Модификатор доступа	Тип	Доступ	Назначение
id	public	Long	get, set	Id.
code	public	Int	get, set	Код.
createdDate	public	Date	get, set	Дата создания
email	public	String	get, set	Email - адрес

Таблица 4.18

Описание методов и свойств класса Contact.kt

Свойства				
Имя	Модификатор доступа	Тип	Доступ	Назначение
id	public	Long	get, set	Id.
user	public	User	get, set	Пользователь.
name	public	String	get, set	Название контакта.
value	public	String	get, set	Значение контакта.

Таблица 4.19

Описание методов и свойств класса Meet.kt

Свойства				
Имя	Модификатор доступа	Тип	Доступ	Назначение
id	public	Long	get, set	Id.
user1	public	User	get, set	Пользователь 1.
user2	public	User	get, set	Пользователь 2.
meetStatus	public	MeetStatus	get, set	Статус встречи.
createdDate	public	Date	get, set	Дата создания.
expiresDate	public	Date	get, set	Дата истечения срока службы.

Таблица 4.20

Описание методов и свойств класса RefreshToken.kt

Свойства				
Имя	Модификатор доступа	Тип	Доступ	Назначение
id	public	Long	get, set	Id.
user	public	User	get, set	Пользователь.
uuid	public	UUID	get, set	UUID.
fingerprint	public	String	get, set	Уникальный идентификатор.
createdDate	public	Date	get, set	Дата создания.
expiresDate	public	Date	get, set	Дата истечения срока службы.

Таблица 4.21

Описание методов и свойств класса Search.kt

Свойства				
Имя	Модификатор доступа	Тип	Доступ	Назначение
id	public	Long	get, set	Id.
finder	public	User	get, set	Пользователь – поисковик.
searchParams	public	SearchParams	get, set	Параметры поиска.
createdDate	public	Date	get, set	Дата создания.

Таблица 4.22

Описание методов и свойств класса SearchParams.kt

Свойства				
Имя	Модификатор доступа	Тип	Доступ	Назначение
id	public	Long	get, set	Id.
genders	public	MutableSet< Gender>	get, set	Множество полов.
minCourse	public	Int	get, set	Минимальный курс.
maxCourse	public	Int	get, set	Максимальный курс.
degrees	public	MutableSet< Degree>	get, set	Множество степеней образования.
faculties	public	MutableSet< Faculty>	get, set	Множество факультетов.

Таблица 4.22

Описание методов и свойств класса User.kt

Свойства				
Имя	Модификатор доступа	Тип	Доступ	Назначение
id	public	Long	get, set	Id.
email	public	String	get, set	Email.
firstName	public	String	get, set	Имя.
lastName	public	String	get, set	Фамилия.
gender	public	Gender	get, set	Пол.
createdDate	public	Date	get, set	Дата создания.
faculty	public	Faculty	get, set	Факультет.
degree	public	Degree	get, set	Степень образования.
contacts	public	MutableList< Contact>	get, set	Контакты.
course	public	Int	get, set	Номер курса.
userStatus	public	UserStatus	get, set	Статус пользователя.
aboutMe	public public	String	get, set	О себе
photoUri		String	get, set	Путь до фото.

Таблица 4.23

Описание методов и свойств класса Api.dart

Методы				
Имя	Модификатор доступа	Тип	Аргументы	Назначение
_getMeetByResponse	private	Метод	http.Response response	Получить статус встречи от ответа сервера.
_updateTokens	private	Метод	-	Обновить токены.
cancelSearch	public	Метод	-	Отменить поиск встречи.
confirmCode	public	Метод	String code, String email	Подтвердить код подтверждения.
getImageUrlByUser	public	Метод	User user	Получить адрес изображения у пользователя.
getMeet	public	Метод	-	Получить встречу.
getMeets	public	Метод	-	Получить встречи.
getUser	public	Метод	-	Получить пользователя.
search	public	Метод	SearchParams searchParams	Начать поиск встречи.
sendCode	public	Метод	String email	Отправить код подтверждения на email.
setPhoto	public	Метод	User user, File file	Загрузить фотографию пользователю.
setUser	public	Метод	User user	Выставить настройки пользователю.
updateImage	public	Метод	-	Обновление адреса для получения фотографии.

Таблица 4.24

Описание методов и свойств класса Auth.dart

Методы				
Имя	Модификатор доступа	Тип	Аргументы	Назначение
getAccessToken	public	Метод	-	Получить access токен.
getData	public	Метод	-	Получить данные из хранилища.
getEmail	public	Метод	-	Получить email – адрес.

getFingerprint	public	Метод	-	Получить уникальный идентификатор.
getRefreshToken	public	Метод	-	Получить refresh токен.
saveDataByJson	public	Метод	String email, String json	Сохранить данные по полученной JSON.

Таблица 4.25

Описание методов и свойств класса Meet.dart

Методы				
Имя	Модификатор доступа	Тип	Аргументы	Назначение
Meet	public	Конструктор	this.user1, this.user2, meetStatus, createdDate, expiresDate	Конструктор.
fromJson	public	Метод	Map<String, dynamic> json	Получить контакты по JSON
toJson	public	Метод	-	Перевести объект в JSON.

Таблица 4.26

Описание методов и свойств класса SearchParams.dart

Методы				
Имя	Модификатор доступа	Тип	Аргументы	Назначение
SearchParams	public	Конструктор	genders, faculties, degrees, minCourse, maxCourse	Конструктор.
fromJson	public	Метод	Map<String, dynamic> json	Получить контакты по JSON
toJson	public	Метод	-	Перевести объект в JSON.

Таблица 4.27

Описание методов и свойств класса User.dart

Методы				
Имя	Модификатор доступа	Тип	Аргументы	Назначение
User	public	Конструктор	this.email, this.firstName, this.lastName, this.gender, this.degree, this.faculty, this.contacts, this.course, this.photoUri, this.aboutMe	Конструктор.

fromJson	public	Метод	Map<String, dynamic> json	Получить контакты по JSON
toJson	public	Метод	-	Перевести объект в JSON.
getFullName	public	Метод	-	Получить полное имя.
getVk	public	Метод	-	Получить ВК.
getInst	public	Метод	-	Получить Инстраграм.
getTelegram	public	Метод	-	Проучить Телеграм.
_getValueContact	private	Метод	String name	Получить контакт по имени.

Таблица 4.28

Описание методов и свойств класса AuthCodeScreen.dart

Методы				
Имя	Модификатор доступа	Тип	Аргументы	Назначение
initState	public	Метод	-	Инициализация состояния.
build	public	Метод	context	Построение виджета.

Таблица 4.29

Описание методов и свойств класса AuthContactsScreen.dart

Методы				
Имя	Модификатор доступа	Тип	Аргументы	Назначение
initState	public	Метод	-	Инициализация состояния.
build	public	Метод	BuildContext context	Построение виджета.
callSnackBar	public	Метод	String text	Вызвать SnackBar об ошибке.
dispose	public	Метод	-	Произвести чистку объектов.
tgClean	public	Метод	String tg	Выполнить обработку текстового поля для Telegram

Таблица 4.30

Описание методов и свойств класса AuthEmailScreen.dart

Методы				
Имя	Модификатор доступа	Тип	Аргументы	Назначение
initState	public	Метод	-	Инициализация состояния.

build	public	Метод	BuildContext context	Построение виджета.

Таблица 4.31

Описание методов и свойств класса AuthFacultyScreen.dart

Методы				
Имя	Модификатор доступа	Тип	Аргументы	Назначение
initState	public	Метод	-	Инициализация состояния.
build	public	Метод	BuildContext context	Построение виджета.
callSnackBar	public	Метод	String text	Вызвать SnackBar об ошибке.
dispose	public	Метод	-	Произвести чистку объектов.

Таблица 4.32

Описание методов и свойств класса AuthGenderScreen.dart

Методы				
Имя	Модификатор доступа	Тип	Аргументы	Назначение
initState	public	Метод	-	Инициализация состояния.
build	public	Метод	BuildContext context	Построение виджета.
callSnackBar	public	Метод	String text	Вызвать SnackBar об ошибке.
dispose	public	Метод	-	Произвести чистку объектов.

Таблица 4.33

Описание методов и свойств класса AuthNameScreen.dart

Методы				
Имя	Модификатор доступа	Тип	Аргументы	Назначение
initState	public	Метод	-	Инициализация состояния.
build	public	Метод	BuildContext context	Построение виджета.
callSnackBar	public	Метод	String text	Вызвать SnackBar об ошибке.
dispose	public	Метод	-	Произвести чистку объектов.

Таблица 4.34

Описание методов и свойств класса AuthPhotoScreen.dart

Методы				
Имя	Модификатор доступа	Тип	Аргументы	Назначение

initState	public	Метод	-	Инициализация состояния.
build	public	Метод	BuildContext context	Построение виджета.
callSnackBar	public	Метод	String text	Вызвать SnackBar об ошибке.
dispose	public	Метод	-	Произвести чистку объектов.
_sendImage	private	Метод	-	Отправить изображение.
_ getImage	private	Метод	ImageSource source	Получить изображение.
_showPickOptionsDialog	private	Метод	BuildContext context	Показать окно выбора способа загрузки изображения.

Таблица 4.35

Описание методов и свойств класса Header.dart

Методы				
Имя	Модификатор доступа	Тип	Аргументы	Назначение
build	public	Метод	BuildContext context	Построение виджета.

Таблица 4.36

Описание методов и свойств класса HomeScreen.dart

Методы				
Имя	Модификатор доступа	Тип	Аргументы	Назначение
_navigateByMeet	private	Метод	Meet meet	Навигация по встрече.
_navigateByMeetStatus	private	Метод	MeetStatus meetStatus	Навигация по статусу встречи.
_updByMeet	private	Метод	Timer timer	Обновить экран по встречи.
build	public	Метод	BuildContext context	Построение виджета.
callSnackBar	public	Метод	String text	Вызвать SnackBar об ошибке.
dispose	public	Метод	-	Произвести чистку объектов.
getScreenByMeet	public	Метод	Meet meet, MeetStatus meetStatus	Получить экран по встрече.

Таблица 4.37

Описание методов и свойств класса HomeCabinetScreen.dart

Методы

Имя	Модификатор доступа	Тип	Аргументы	Назначение
build	public	Метод	BuildContext context	Построение виджета.
initState	public	Метод	-	Инициализация состояния.

Таблица 4.38

Описание методов и свойств класса HomeFindScreen.dart

Методы				
Имя	Модификатор доступа	Тип	Аргументы	Назначение
build	public	Метод	BuildContext context	Построение виджета.
initState	public	Метод	-	Инициализация состояния.
_settingModalBottomSheet	private	Метод	BuildContext context	Выставление настроек для нижнего экрана.
callSnackBar	public	Метод	String text	Вызвать SnackBar об ошибке.
startFind	public	Метод	BuildContext context	Начать поиск встречи.
showModalBottomSheet	public	Метод	-	Открыть экран с установкой параметров поиска.

Таблица 4.39

Описание методов и свойств класса HomeLoadingScreen.dart

Методы				
Имя	Модификатор доступа	Тип	Аргументы	Назначение
build	public	Метод	BuildContext context	Построение виджета.
initState	public	Метод	-	Инициализация состояния.
_cancelMeet	private	Метод	-	Отменить поиск встречи.
callSnackBar	public	Метод	String text	Вызвать SnackBar об ошибке.

Таблица 4.40

Описание методов и свойств класса HomeMeetsScreen.dart

Методы				
Имя	Модификатор доступа	Тип	Аргументы	Назначение
build	public	Метод	BuildContext context	Построение виджета.

initState	public	Метод	-	Инициализация состояния.
loadMeets	public	Метод	-	Начать загрузку встреч.

Таблица 4.41

Описание методов и свойств класса HomePersonScreen.dart

Методы				
Имя	Модификатор доступа	Тип	Аргументы	Назначение
build	public	Метод	BuildContext context	Построение виджета.
initState	public	Метод	-	Инициализация состояния.

Таблица 4.42

Описание методов и свойств класса SplashScreen.dart

Методы				
Имя	Модификатор доступа	Тип	Аргументы	Назначение
build	public	Метод	BuildContext context	Построение виджета.
initState	public	Метод	-	Инициализация состояния.

Таблица 4.43

Описание методов и свойств класса SplashPainter.dart

Методы				
Имя	Модификатор доступа	Тип	Аргументы	Назначение
paint	public	Метод	Canvas canvas, Size size	Отрисовка на холсте фигур.
drawRandomOvals	public	Метод	Canvas canvas, Paint paint, int n, double width, double height	Отрисовка случайного овала.
shouldRepaint	public	Метод	CustomPainter oldDelegate	Метод, возвращающий true, если нужна перерисовка.

Таблица 4.44

Описание методов и свойств класса ButtonContinue.dart

Методы				
Имя	Модификатор доступа	Тип	Аргументы	Назначение
build	public	Метод	BuildContext context	Построение виджета.

Таблица 4.45

Описание методов и свойств класса capsule_widget.dart

Методы				
Имя	Модификатор доступа	Тип	Аргументы	Назначение
build	public	Метод	BuildContext context	Построение виджета.

Таблица 4.46

Описание методов и свойств класса dialog_loading.dart

Методы				
Имя	Модификатор доступа	Тип	Аргументы	Назначение
build	public	Метод	BuildContext context	Построение виджета.

Таблица 4.47

Описание методов и свойств класса edu_fields.dart

Методы				
Имя	Модификатор доступа	Тип	Аргументы	Назначение
build	public	Метод	BuildContext context	Построение виджета.

Таблица 4.48

Описание методов и свойств класса toggle_button_gender.dart

Методы				
Имя	Модификатор доступа	Тип	Аргументы	Назначение
build	public	Метод	BuildContext context	Построение виджета.
getGenders	public	Метод	-	Получить список гендеров.
onClicked	public	Метод	Int index	Обработчик нажатия при клике.

Таблица 4.49

Описание методов и свойств класса main.dart

Методы				
Имя	Модификатор доступа	Тип	Аргументы	Назначение
build	public	Метод	BuildContext context	Построение виджета.

Таблица 4.50

Описание методов и свойств класса router_auth.dart

Методы				
Имя	Модификатор доступа	Тип	Аргументы	Назначение

routeByUser	public	Метод	BuildContext context, User user	Маршрутизация относительно пользователя.

ЛИСТ РЕГИСТРАЦИИ ИЗМЕНЕНИЙ

Лист регистрации изменений									
Номера листов (страниц)					Всего листов (страниц в докум.)	№ документа	Входящий № сопроводительного докум. и дата	Подп.	Дата
Изм.	Измененных	Замененных	Новых	Аннулирован					

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729.04.05 —01 81				
Инв. № подл.	Подп. и дата	Взам. инв. №	Инв. № дубл.	Подп. и дата