

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ТЕЛЕКОММУНИКАЦИЙ ИМ. ПРОФ. М.А. БОНЧ-БРУЕВИЧА»
(СПбГУТ)

АРХАНГЕЛЬСКИЙ КОЛЛЕДЖ ТЕЛЕКОММУНИКАЦИЙ
ИМ. Б.Л. РОЗИНГА (ФИЛИАЛ) СПбГУТ
(АКТ (ф) СПбГУТ)

КУРСОВОЙ ПРОЕКТ

НА ТЕМУ

РАЗРАБОТКА МОБИЛЬНОГО ПРИЛОЖЕНИЯ

«ПОДБОР ПЕРСОНАЛА ДЛЯ ВЫПОЛНЕНИЯ

ЗАДАЧ»

Л109. 24КП01. 020 ПЗ

(Обозначение документа)

МДК.02.01 Технология разработки

программного обеспечения

Студент	ИСПП-11	05.12.2024	В.С Стрельцов
	(Группа)	(Подпись)	(И.О. Фамилия)
Преподаватель		05.12.2024	Ю.С. Маломан
		(Подпись)	(И.О. Фамилия)

Архангельск 2024

СОДЕРЖАНИЕ

Перечень сокращений и обозначений.....	3
Введение.....	4
1 Анализ и разработка требований.....	6
1.1 Назначение и область применения.....	?
1.2 Постановка задачи	?
1.3 Описание алгоритма функционирования системы	?
1.4 Выбор состава программных и технических средств	?
2 Проектирование программного обеспечения	?
2.1 Проектирование интерфейса пользователя.....	?
2.2 Разработка архитектуры программного обеспечения.....	?
2.3 Проектирование базы данных	?
3 Разработка и интеграция модулей программного обеспечения.....	?
3.1 Разработка программных модулей.....	?
3.2 Реализация интерфейса пользователя.....	?
3.3 Разграничение прав доступа пользователей	?
3.4 Экспорт и импорт данных.....	4
4 Тестирование и отладка программного обеспечения.....	?
4.1 Структурное тестирование.....	?
4.2 Функциональное тестирование	?
5 Инструкция по эксплуатации программного обеспечения	?
5.1 Установка программного обеспечения.....	?
5.2 Инструкция по работе	?
Заключение	?
Список использованных источников	30
Приложение	?

ПЕРЕЧЕНЬ СОКРАЩЕНИЙ И ОБОЗНАЧЕНИЙ

В настоящем курсовом проекте применяют следующие сокращения и обозначения.

ЗНФ – третья нормальная форма

БД – база данных

ИС – информационная система

ОЗУ – оперативное запоминающее устройство

ПК – персональный компьютер

ПО – программное обеспечение

СУБД – система управления базами данных

ER - entity-relationship model, модель «сущность — связь»

IDE - Integrated Development Environment, интегрированная среда разработки

MS – Microsoft

DVD – формат оптических носителей последнего поколения

CD – оптический носитель информации в виде пластикового диска с отверстием в центре, процесс записи и считывания информации которого осуществляется при помощи лазера

ВВЕДЕНИЕ

Актуальность данного проекта заключается в том, что он предоставляет решение актуальной проблемы в области фриланса - неэффективного взаимодействия между заказчиками и соискателям.

Современный мир труда претерпевает значительные изменения. Рост популярности удаленной работы и фриланса формирует новую реальность, где заказчики ищут квалифицированных специалистов, а соискатели ищут интересные проекты.

По причине неэффективного взаимодействия между заказчиками и соискателям, заказчики нуждаются в более эффективных способах взаимодействия с потенциальными кандидатами. Разработка мобильного приложения сделает поиск заказчиков более эффективным и простым, упростит коммуникацию между заказчиками и соискателями, улучшит взаимодействие и приведет к более успешному поиску подходящих кандидатов на их задачу.

Целью курсового проектирования является разработка мобильного приложения для подбора персонала на выполнение задач, которое позволит просматривать, редактировать и размещать задачи, находить специалистов по областям, вести диалог с заказчиками и экспертами.

Для достижения поставленной цели требуется решить следующие задачи:

- выполнить сбор требований целевой аудитории,
- проанализировать информационные источники по предметной области,
- спроектировать архитектуру приложения,
- выбрать состав программных и технических средств, используемых для реализации информационно-поисковой системы,
- спроектировать диаграмму вариантов использования приложения,
- спроектировать физическую схему БД,

- разработать API для мобильного приложения,
- реализовать разграничение прав доступа пользователей,
- реализовать защиту данных,
- разработать мобильное приложение,
- реализовать работу с сервером при помощи REST API,
- выполнить тестирование и отладку ПО и проанализировать результаты тестирования,
- разработать программную и эксплуатационную документацию

1 Анализ и разработка требований

1.1 Назначение и область применения

Данное мобильное приложение предназначено для физических и юридических лиц, желающих оптимизировать процесс поиска экспертов на свои задачи. Мобильное приложение упрощает процесс поиска экспертов для своих задач, просмотр информации об эксперте, позволяет создавать задачи, редактировать информацию о своих задачах, а также приглашать экспертов на задачи.

1.2 Постановка задачи

Необходимо разработать мобильное приложение, которое будет выполнять следующие функции:

- авторизация,
- регистрация,
- просмотр информации об эксперте,
- просмотр и фильтрация информации про опыт эксперта,
- создание, редактирование информации о своих задачах,
- приглашение экспертов на задачи,
- общение с экспертами,
- отправка файлов и изображений в общении с экспертами,
- выставление оценки экспертам, после закрытия задачи.

1.3 Описание алгоритма функционирования системы

При запуске приложения отображается начальная страница, на которой присутствует описание функционала мобильного приложения и в панели приложения находится кнопка аутентификации, после нажатия на кнопку,

открывается модальное окно аутентификации с возможностью выбора роли заказчика и эксперта.

На рисунке 1 изображена UML диаграмма вариантов использования приложения для категорий пользователя.

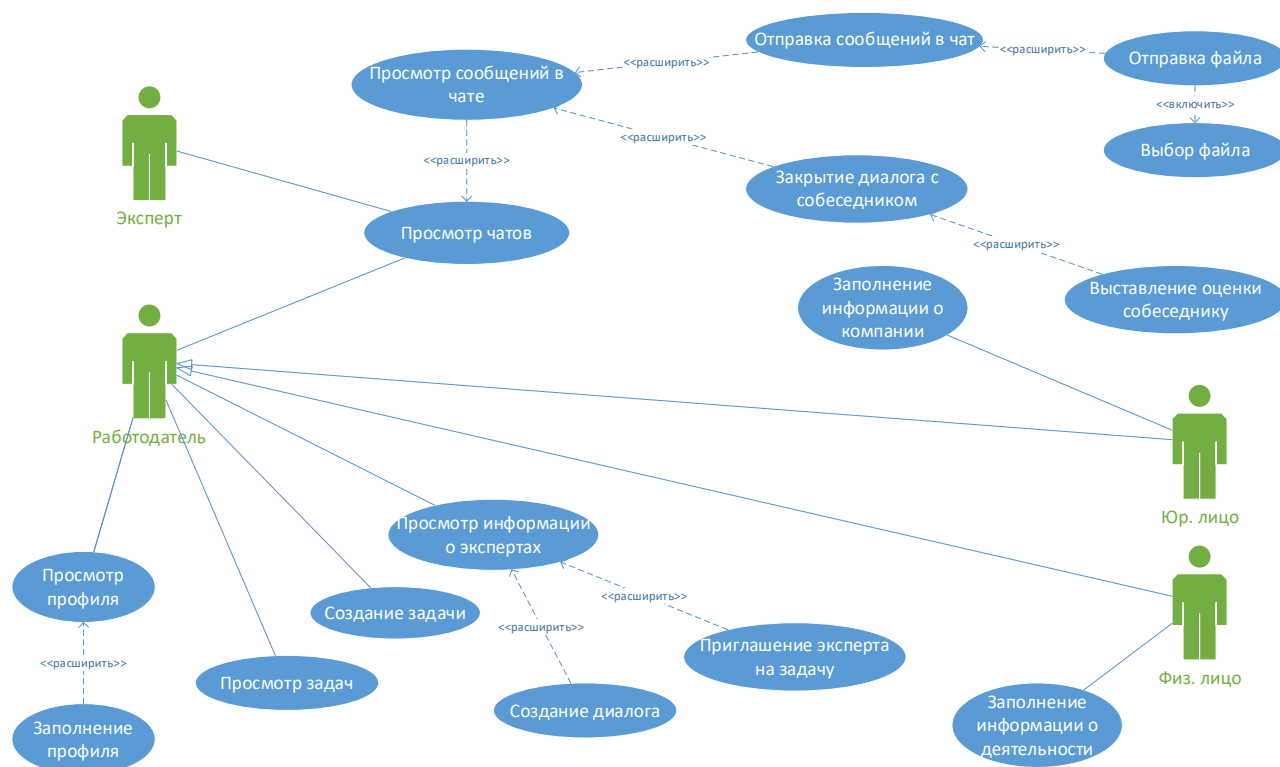


Рисунок 1 – Диаграмма вариантов использования приложения

1.4 Выбор состава программных и технических средств

Согласно цели проекта требуется создать мобильное приложение для подбора персонала на задачу.

Работа с мобильным приложением будет осуществляться на мобильных устройствах с установленной операционной системой Android 11 или IOS 18, с интернет-подключением.

В качестве системы управления базами данных выбрана СУБД PostgreSQL 15, так как данная СУБД позволяет определять собственные типы данных, функции и операторы, обеспечивает высокую производительность

благодаря оптимизации запросов, индексации, поддерживает различные типы данных, включая JSON.

Приложение будет написано на языке программирования Dart, с использованием фреймворка Flutter, так как данный фреймворк поддерживает кроссплатформенность, компилируется в нативный код и выдает высокую производительность.

Для разработки приложения будет использоваться интегрированная среда разработки программ Android Studio 2023.1.1, так как данная IDE имеет функцию быстрой перезагрузки, встроенные эмуляторы и поддерживает разные языки программирования.

Для функционирования системы на стороне сервера достаточны следующие программные и технические средства:

- операционная система Ubuntu,
- сервер БД: PostgreSQL не ниже 12.0,
- процессор с частотой 2 ГГц,
- свободная оперативная память объемом 1 ГБ,
- программное обеспечение для конфигурирования, управления и администрирования сервера БД: pgAdmin,
- программное обеспечение для работы API: Python версией не ниже 3.6, Django версией не ниже 3.2, Django REST Framework.

Для функционирования системы на стороне клиента достаточны следующие программные и технические средства:

- операционная система Android версией не ниже 11 или IOS версией не ниже 18,
- процессор с частотой 2 ГГц,
- оперативная память в объеме 2 ГБ,
- свободное место в хранилище 200 МБ,
- постоянное интернет-подключение.

2 Проектирование программного обеспечения

2.1 Проектирование интерфейса пользователя

В рамках разработки мобильного приложения «Подбор персонала для выполнения задач» был создан интерфейс пользователя в виде wireframe и мокапах при помощи сайта Figma. Эти визуальные представления позволяют наглядно увидеть структуру приложения, его основные элементы и функциональность.

Некоторые wireframe интерфейса мобильного приложения можно увидеть на рисунке 2.

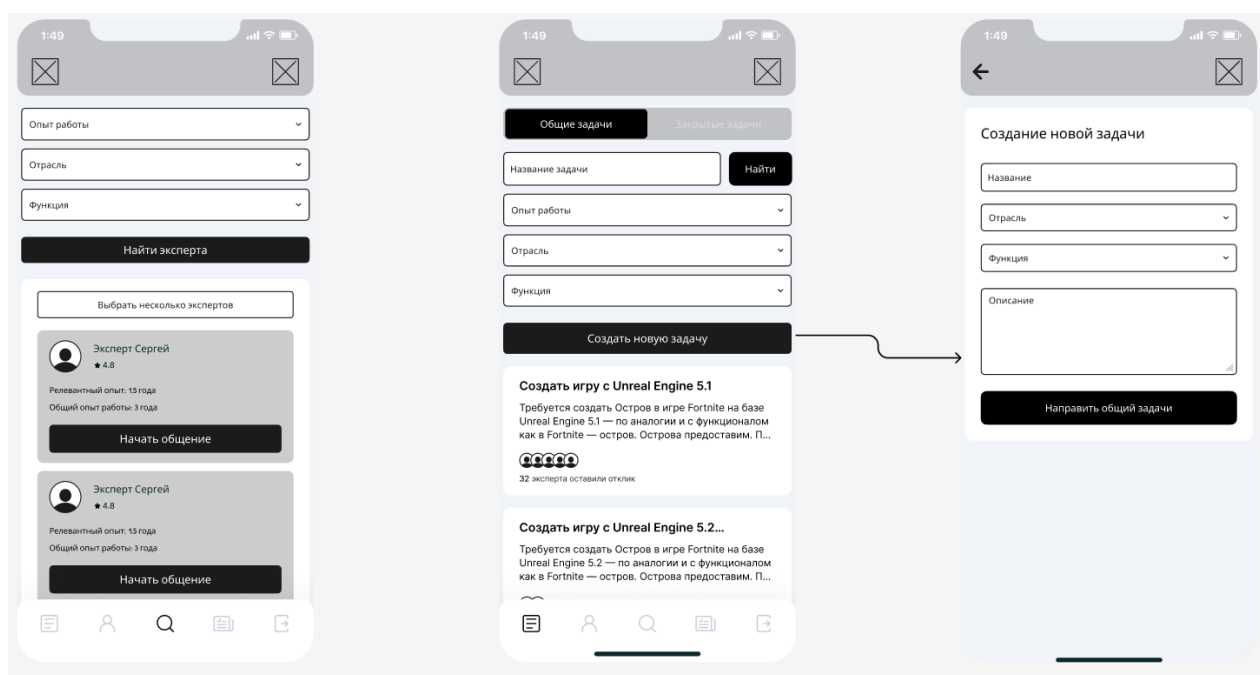


Рисунок 2 – Figma. Wireframe интерфейс пользователя мобильного приложения

Также был создан логотип для мобильного приложения, который размещен в панели навигации. Часть некоторых мокапов интерфейса и логотип мобильного приложения можно увидеть на рисунке 3.

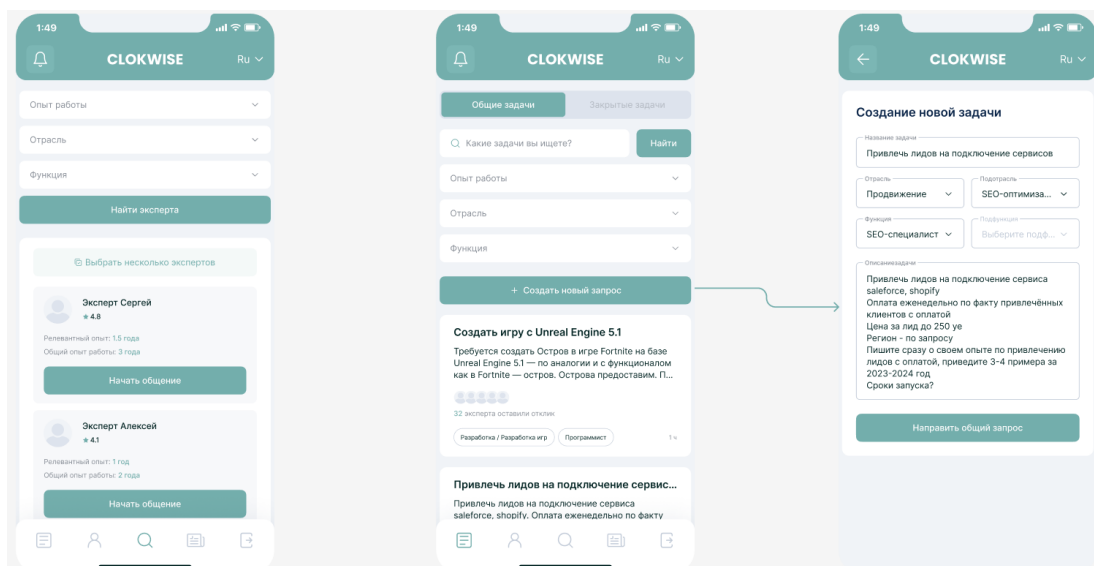


Рисунок 3 – Figma. Мокапы интерфейс пользователя мобильного приложения

2.2 Разработка архитектуры программного обеспечения

Приложение для поиска специалистов предназначено для соединения заказчиков и исполнителей услуг. Архитектура приложения построена на основе клиент-серверной модели и включает в себя несколько ключевых компонентов: backend, мобильное приложение, БД.

Backend будет использовать фреймворк Django с библиотекой Django REST Framework для создания API, позволяющая клиенту взаимодействовать с сервером. Аутентификация будет реализована при помощи Json Web Tokens.

БД будет использовать СУБД PostgreSQL. БД должна хранить в себе информацию о пользователях, задачах, отзывах, чатах и сообщениях.

Мобильное приложение будет написано на фреймворке Flutter для поддержки кроссплатформенности. Интерфейс пользователя разработан с учетом удобства и простоты использования, включая страницы для регистрации, поиска специалистов, создания задач и оставления отзывов. Взаимодействие с сервером будет происходить при помощи HTTP-запросов к RESTful API, ответы будут приходить в формате JSON. Для отправки запросов

будет использоваться веб-клиент Dio. Токен авторизации будет храниться в зашифрованном виде.

2.3 Проектирование базы данных

Требуется разработать БД для системы подбора персонала. Система будет использоваться рекрутерами в виде физических или юридических лиц.

Эксперт может просматривать, откликаться на задачи, вести диалоги с работодателями, просматривать профиль работодателя.

Работодатель может создавать задачи, просматривать экспертов по областям специальности, приглашать экспертов на задачи, оставлять отзывы экспертам, вести диалоги с экспертами.

Модели БД созданы при помощи DBDiagram.io. На рисунке 4 показана часть физической модели предметной области, связанной с поиском экспертов при помощи задачи, в виде ERD.

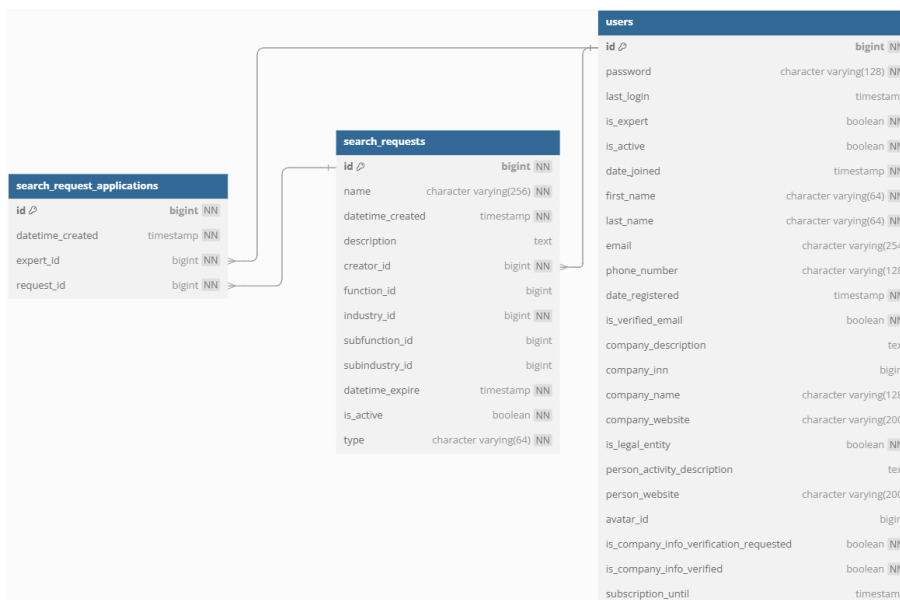


Рисунок 4 – DBDiagram.io – Часть физической модели БД

3 Разработка и интеграция модулей программного обеспечения

3.1 Разработка программных модулей

Для курсового проекта было разработано мобильное приложение на Dart с использованием фреймворка Flutter в Android Studio [4]. В приложении реализована авторизация через ввод логина и пароля в модальном окне, которая открывается кнопкой на начальной странице, с ошибкой при неверной комбинацией, и регистрация новых пользователей. Данные страницы видны на рисунке 5.

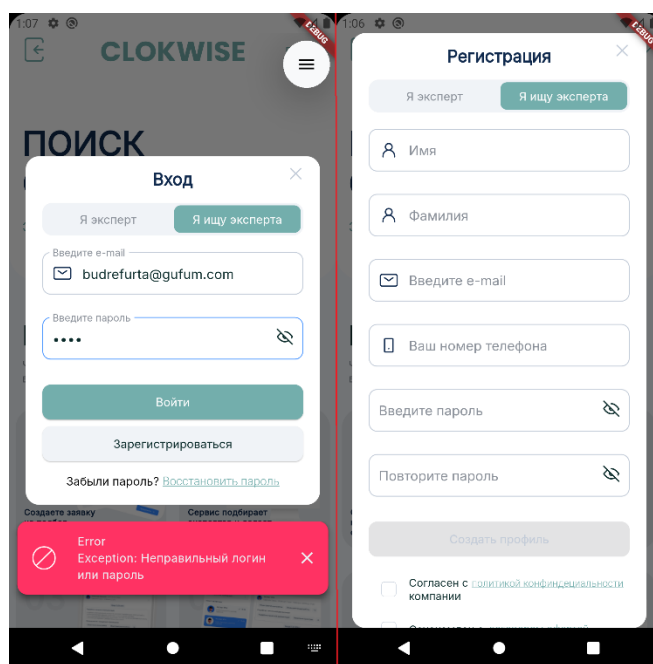


Рисунок 5 – Clokwise. Вид страниц с модальными окнами
«Авторизация» и «Регистрация»

После успешной авторизации пользователь попадает на страницу с профилем, при помощи нижней навигационной панели можно попасть на

страницу поиска экспертов. Страницы с профилем и поиском экспертов на рисунке 6.

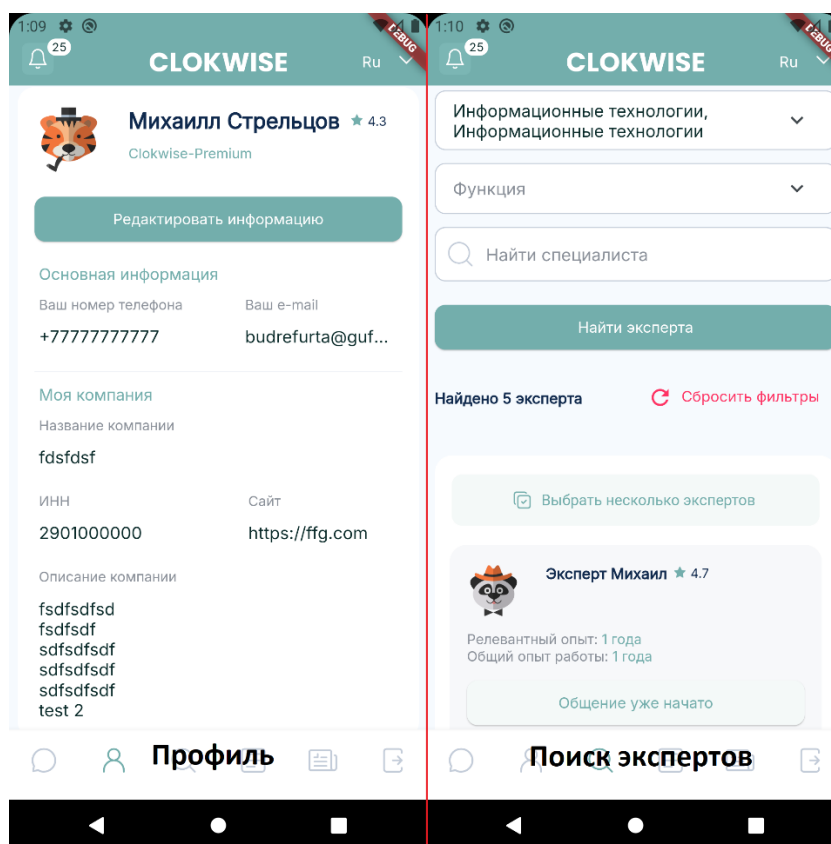


Рисунок 6 – Clokwise. Вид страниц «Профиль» и «Поиск экспертов»

Для реализации получения данных из БД использован сетевой клиент из плагина Dio для отправления POST-запроса на сервер с данными для фильтрации. Код метода репозитория search для получения данных экспертов представлен в листинге 1.

Листинг 1 – Код метода для отправки POST-запроса на сервер

```
/// Отправляет запрос на сервер для получения результатов  
поиска.  
/// Принимает объект [searchRequest], который содержит  
параметры поиска.  
/// Возвращает список результатов поиска в виде
```

```

///[List<ExpertResultModel>].
///Исключение будет выброшено, если сервер вернет статус-код,
///отличный от 200.
Future<List<ExpertResultModel>> postSearchResult({
    required SearchRequest searchRequest}) async {
    // Выполняем POST-запрос к серверу с данными запроса в
    формате JSON.
    final response = await _mainNetworkClient.client.post(
        '$_path/',
        data: searchRequest.toJson());
    // Проверяем, был ли ответ успешным (статус-код 200).
    if (response.statusCode != 200) {
        throw Exception('Ошибка при выполнении запроса:
        ${response.statusCode}'); // Выбрасываем исключение при ошибке}

    // Преобразуем данные из ответа в список объектов
    final experts = response.data.map<ExpertResultModel>(
        (message) => ExpertResultModel.fromJson(message)).toList();
    return experts; // Возвращаем список результатов поиска}

```

Код представления в API серверной части для поиска экспертов представлен ниже в листинге 2.

Листинг 2 – Код представления для поиска экспертов в API

```

class ExpertSearchView(APIView):
    """
    API View для поиска экспертов на основе заданных критериев.

    Этот класс обрабатывает POST-запросы для поиска экспертов,
    используя различные параметры,
    такие как индустрия, подиндустрия, функция, подфункция,
    годы опыта и текстовый запрос.

    Параметры запроса:
    - industry: ID индустрии (опционально)
    - subindustry: ID подиндустрии (опционально)
    - function: ID функции (опционально)
    - subfunction: ID подфункции (опционально)
    - experience: Количество лет опыта (опционально)
    - prompt: Текстовый запрос для поиска (опционально)

    Ответ:
    Возвращает список найденных экспертов в формате JSON.
    """

    permission_classes = [AllowAny]

```

```

def post(self, request):
    """
    Обрабатывает POST-запрос для поиска экспертов.
    Аргументы:
    - request: объект запроса, содержащий данные для
поиска.
    Возвращает:
    - Response: объект ответа с данными о найденных
экспертах или статус ошибки.
    """

    serializer = ExpertSearchSerializer(data=request.data)
    if not serializer.is_valid():
        return Response(status=status.HTTP_400_BAD_REQUEST)

    data = serializer.validated_data
    relevant_segments_conditions = Q()

    # Формирование условий фильтрации сегментов опыта
    if industry_id := data.get("industry"):
relevant_segments_conditions.add(Q(industry=industry_id),
Q.AND)
        if subindustry_id := data.get("subindustry"):
relevant_segments_conditions.add(Q(subindustry=subindustry_id),
Q.AND)
        if function_id := data.get("function"):
relevant_segments_conditions.add(Q(function=function_id),
Q.AND)
        if subfunction_id := data.get("subfunction"):
relevant_segments_conditions.add(Q(subfunction=subfunction_id),
Q.AND)

    if not relevant_segments_conditions.children:
        return Response(status=status.HTTP_400_BAD_REQUEST)

    # Извлечение сегментов опыта экспертов
    segments =
ExpertExperienceSegment.objects.select_related(
        "experience_record", "experience_record__user"
    ).annotate(
        duration=ExpressionWrapper(F("date_end") -
F("date_start"), output_field=DurationField())
    )

    relevant_segments =
segments.filter(relevant_segments_conditions)

    # Суммирование релевантного и общего опыта
    relevant_experience = relevant_segments.values(
        "experience_record__user"
    ).order_by(
        "experience_record__user"

```

```

        ).annotate(
            relevant_experience=Sum("duration")
        )

    total_experience = segments.values(
        "experience_record__user"
    ).order_by(
        "experience_record__user"
    ).annotate(
        total_experience=Sum("duration")
    )

    # Фильтрация экспертов на основе релевантных сегментов
    experts_conditions =
Q(pk__in=relevant_segments.values("experience_record__user"))

    experts = User.objects.filter(
        experts_conditions
    ).annotate(
        relevant_experience=Subquery(
            relevant_experience.filter(
                experience_record__user=OuterRef("pk")
            ).values(
                "relevant_experience"
            )
        ),
        total_experience=Subquery(
            total_experience.filter(
                experience_record__user=OuterRef("pk")
            ).values(
                "total_experience"
            )
        ),
        chat_id=Subquery(
            Chat.objects.filter(
                search_request__isnull=True,
                expert=OuterRef("pk"),
                initiator__id__in=[OuterRef("pk"),
request.user.id],
                is_closed=False
            ).values("id")[:1]
        )
    )

    # Фильтрация по годам опыта
    if experience_years := data.get("experience"):
        experts =
experts.filter(relevant_experience__gte=timedelta(days=experience_years * 365))

    # Поиск по текстовому запросу
    if prompt := data.get("prompt"):
        vector = SearchVector(

```



```

        "company_name",
        "experience_records__segments__description",
        "experience_records__segments__industry__name",
        "experience_records__
segments__subindustry__name",
        "experience_records__segments__function__name",
        "experience_records__segments__subfunction__name",
    )
    query = SearchQuery(prompt)
    experts = experts.annotate(
        search=vector, rank=SearchRank(vector, query)
    ).filter(
        search=query
    ).order_by("id", "-rank", "-
relevant_experience").distinct("id")
    else:
        experts = experts.order_by("-relevant_experience")

    # Формирование ответа
    json = FoundExpertSerializer(instance=experts,
many=True).data
    return Response(data=json, status=status.HTTP_200_OK)
```python
Примечания:
- Убедитесь, что все необходимые импорты и зависимости
присутствуют в вашем проекте.
- Проверьте, что сериализаторы (например,
ExpertSearchSerializer и FoundExpertSerializer) корректно
определены и работают.
- Обратите внимание на обработку ошибок и валидацию данных,
чтобы обеспечить надежность API.

```

## 3.2 Реализация интерфейса пользователя

Интерфейс разработан с использованием постраничной навигации, в приложении разработаны различные элементы управления, стили и виджеты для упрощения работы. Навигация в приложении реализована с помощью виджета Navigator, который управляет стеком объектов Route, представляющих страницы в приложении []. Для упрощения работы с маршрутами использован пакет go\_router.

Для отображения информации об экспертах разработан виджет SearchExpertItem, который представлен в листинге 3.

### Листинг 3 – Код виджета для SearchExpertItem

```
/// Виджет для отображения информации о эксперте в результате
поиска.
class SearchExpertItem extends StatelessWidget {
 final int userId; // Идентификатор пользователя
 final int? chatId; // Идентификатор чата
 final bool isChosen; // Флаг, указывающий, выбран ли эксперт
 final String name; // Имя эксперта
 final String? avatar; // URL аватара эксперта
 final double? rating; // Рейтинг эксперта
 final String relevantExperience; // Релевантный опыт эксперта
 final String totalExperience; // Общий опыт работы эксперта
 final bool isPaidForView; // Флаг, указывающий, оплачен ли
//просмотр
 final VoidCallback onTap; // Обработчик нажатия на элемент

 const SearchExpertItem({
 required this.userId, required this.chatId, required
this.name, required this.relevantExperience, required
this.totalExperience, required this.isChosen, required
this.onTap, required this.avatar, required this.rating,
required this.isPaidForView, super.key});

 @override
 Widget build(BuildContext context) {
 return Column(children: [
 InkWell(
 onTap: onTap,
 child: Ink(
 child: Container(
 width: double.infinity,
 child: Column(
 crossAxisAlignment: CrossAxisAlignment.start,
 children: [
 // Виджет для отображения аватара и информации об эксперте
 AvatarSearchWidget(
 image: avatar,
 fullName: 'Эксперт $name',
 rating: rating,
 isPaidForView: isPaidForView),
 // Отображение релевантного опыта
 RichText(
 text: TextSpan('Релевантный опыт: ',
 children: [
 TextSpan(
 text: relevantExperience),
 // Отображение общего опыта работы
 RichText(text: TextSpan(
 text: 'Общий опыт работы: ',
 children: [
 TextSpan(totalExperience)
```

```

 chatId == null ? CwElevatedButton(
 text: 'Начать общение',
 onTap: () { // Запуск события начала чата
 BlocProvider.of<SearchBloc>(context).
 add(SearchEvent.startChat(userId: userId));
 }
) : CwElevatedButton(
 text: 'Общение уже начато',
 onTap: () {
 // Показать детали чата, если чат уже начат
 CustomNavigator.showModal(context: context,
 child: ChatDetailsPage(id: chatId!));
 }
));
 }
}

```

### 3.3 Разграничение прав доступа пользователей

В мобильном приложении было разработано разграничение прав доступа пользователей. Для этого в приложении были реализованы авторизация и регистрация. Пользователь с ролью заказчика под названием «Ищу эксперта», может производить поиск экспертов, создавать задачи, а также заполнять информацию в профиле про компанию или свою деятельность. Пользователь с ролью «Эксперт», заполняет в профиле информацию про образование и опыт. Пример разграничения прав доступа пользователей можно увидеть на примере заполнения информации на странице «Редактирование профиля» на рисунке 7.

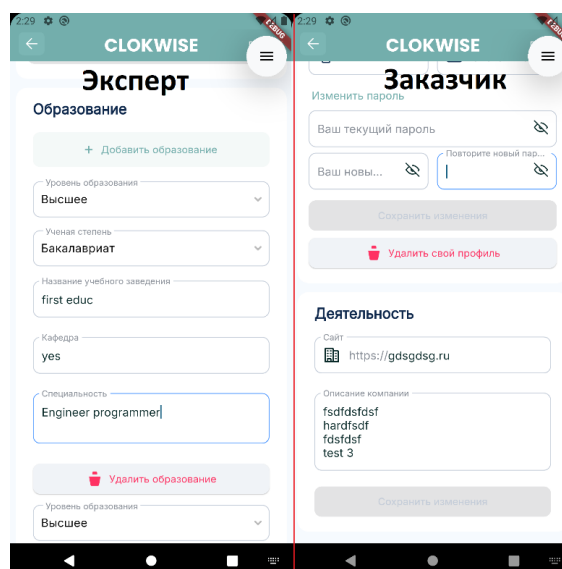


Рисунок 7 – Clokwise. Вид страницы «Редактирование профиля»

### 3.4 Экспорт и импорт данных

В мобильном приложении реализован экспорт и импорт данных при помощи библиотек `flutter_downloader` и `file_picker`.

Пример экспорта данных можно увидеть на листинге 4.

Листинг 4 – Код экспорта данных

```
final data = dio.FormData.fromMap({
 'file': dio.MultipartFile.fromBytes(
 image.bytes,
 filename: 'image.${image.fileTypeInfo.extension}',
 contentType: MediaType('image',
image.fileTypeInfo.extension),
),
});

final response = await _mainNetworkClient.client.post(
 '$_path/upload-attachment/$chatId/',
 data: data,
);
```

Пример импорта данных можно увидеть на листинге 5.

Листинг 5 – Код импорта данных

```
final taskId = await FlutterDownloader.enqueue(
 url: '${AppUrls.path}/$fileId/',
 headers: {'Authorization': 'Bearer $authToken'},
 savedDir: directory.path,
 saveInPublicStorage: true,
 showNotification: true,
 openFileFromNotification: true, // click on notification
to open downloaded file (for Android)
);

FlutterDownloader.registerCallback(downloadCallback);
final tasks = await FlutterDownloader.loadTasks();
```

## 4 Анализ и разработка требований

### 4.1 Структурное тестирование

Во время курсового проектирования было проведено структурное тестирование для `search_repository`. Была использована библиотека `mockito` для создания имитации поведения зависимостей для репозитория и библиотека `flutter_test`. В листинге 3 приведен код unit-теста для поиска экспертов.

Листинг 6 – Код unit-теста для `search_repository`

```
void main() {
 late MockSearchRepository mockSearchRepository;
 late MockSecureStorageManager mockSecureStorageManager;
 late MockMainNetworkClient mockMainNetworkClient;

 TestWidgetsFlutterBinding.ensureInitialized();

 setUp(() {
 mockSecureStorageManager = MockSecureStorageManager();
 mockMainNetworkClient = MockMainNetworkClient();
 mockSearchRepository = MockSearchRepository();
 });

 group('SearchRepository', () {
 test('postSearchResult returns list of experts on success',
 () async {
 final searchRequest = SearchRequest(industry: 1,
experience: 0);
 final result = await
mockSearchRepository.postSearchResultMock(searchRequest);

 expect(result, isA<List<ExpertResultModel>>());
 expect(result.length, 2);
 expect(result[0].id, 1);
 expect(result[0].firstName, 'Expert 1');
 expect(result[1].id, 2);
 expect(result[1].firstName, 'Expert 2');
 });

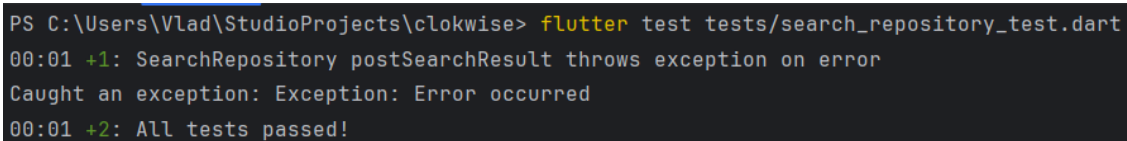
 test('postSearchResult throws exception on error', () async
{
```

```

 final searchRequest = SearchRequest(prompt: 'invalid',
industry: 1, experience: 0);
 try {
 // Используем метод мока для выбрасывания исключения
 await
mockSearchRepository.throwExceptionOnPostSearchResult(searchReq
uest);
 } catch (e) {
 // Выводим информацию об исключении
 print('Caught an exception: $e');
 // Проверяем, что исключение действительно произошло
 expect(e, isA<Exception>());
 }
 });
});
}

```

На рисунке 8 изображена консоль с результатами тестирования, где присутствуют отловленные исключения и успешное тестирование.



```

PS C:\Users\Vlad\StudioProjects\clockwise> flutter test tests/search_repository_test.dart
00:01 +1: SearchRepository postSearchResult throws exception on error
Caught an exception: Exception: Error occurred
00:01 +2: All tests passed!

```

Рисунок 8 – Android Studio. Результаты unit-тестирования

## 4.2 Функциональное тестирование

Во время курсового проектирования было проведено функциональное тестирование при помощи виджет-тестирования из библиотеки `integration_test`.

Листинг 7 – Код интеграционного тестирования для страницы с авторизацией

```

import 'package:flutter/material.dart';
import 'package:flutter_test/flutter_test.dart';
import 'package:integration_test/integration_test.dart';
import 'package:clockwise/main.dart' as app;

void main() {

```

```

// Инициализация интеграционного тестирования.
IntegrationTestWidgetsFlutterBinding.ensureInitialized();
/// Тест для проверки функциональности входа в систему.
///
/// Этот тест выполняет следующие шаги:
/// 1. Запускает приложение.
/// 2. Находит элементы интерфейса для ввода имени
пользователя и пароля.
/// 3. Вводит тестовые данные (имя пользователя и пароль).
/// 4. Нажимает кнопку входа.
/// 5. Проверяет, что после успешного входа отображается имя
пользователя.
testWidgets("Login Test", (WidgetTester tester) async {
 // Запускаем основное приложение.
 app.main();
 // Дожидаемся завершения анимаций и обновления интерфейса.
 await tester.pumpAndSettle();

 // Находим кнопку входа по ключу.
 final menuButton = find.byKey(ValueKey('enterButton'));
 // Проверяем, что кнопка входа отображается на экране.
 expect(menuButton, findsOneWidget);

 // Находим поле для ввода имени пользователя и пароля по
ключам.
 final usernameField =
find.byKey(ValueKey('emailTextField'));
 final passwordField =
find.byKey(ValueKey('passwordTextField'));

 // Вводим тестовый адрес электронной почты.
 await tester.enterText(usernameField,
'budrefurta@gufum.com');
 await tester.pumpAndSettle(); // Обновляем интерфейс.

 // Вводим тестовый пароль.
 await tester.enterText(passwordField, 'test123456');
 await tester.pumpAndSettle(); // Обновляем интерфейс.

 // Находим кнопку входа по ключу и нажимаем на нее.
 final enterButton = find.byKey(ValueKey('enterButton'));
 await tester.tap(enterButton);

 await tester.pumpAndSettle(); // Дожидаемся завершения
анимаций.

 // Проверяем, что после успешного входа отображается имя
пользователя.
 expect(find.text('Михаилл'), findsOneWidget);
});
}

```

## 5 Анализ и разработка требований

### 5.1 Инструкция по эксплуатации программного обеспечения

Минимальные системные требования для серверной части:

- Операционная система - Linux(Ubuntu, Debian);
- Процессор - 2 ГГц;
- Оперативная память - 2 ГБ;
- Свободное место на диске – 10 ГБ;
- Дополнительные компоненты: Python 3.6 (или выше), PostgreSQL, Django REST Framework, Nginx (или Apache).

Добавление объектов БД происходит через миграции от Django REST Framework.

Для установки серверной части требуется перейти в терминале в корневую папку API, и в терминале использовать команду `python manage.py runserver`.

Минимальные системные требования для мобильного приложения:

- Операционная система - Android 11 (IOS 18);
- Процессор - 2 ГГц;
- Оперативная память - 2 ГБ;
- Свободное место в хранилище – 200 МБ;
- Дополнительное - постоянное интернет-подключение.

Для инсталляции мобильного приложения требуется выбрать собранный apk файл с приложением в проводнике и установить его.

В мобильном приложении используются следующие данные для авторизации:

- Электронная почта – [budrefurta@gufum.com](mailto:budrefurta@gufum.com);
- Пароль – test123456.



## 5.2 Инструкция по работе

При запуске приложения, пользователя встречает начальное окно с общей информацией о приложении. Для авторизации требуется нажать на иконку логина в левом верхнем углу страницы. Ввести данные для авторизации и выбрать под какой ролью она происходит в появившемся модальном окне, отображенном на рисунке 9.

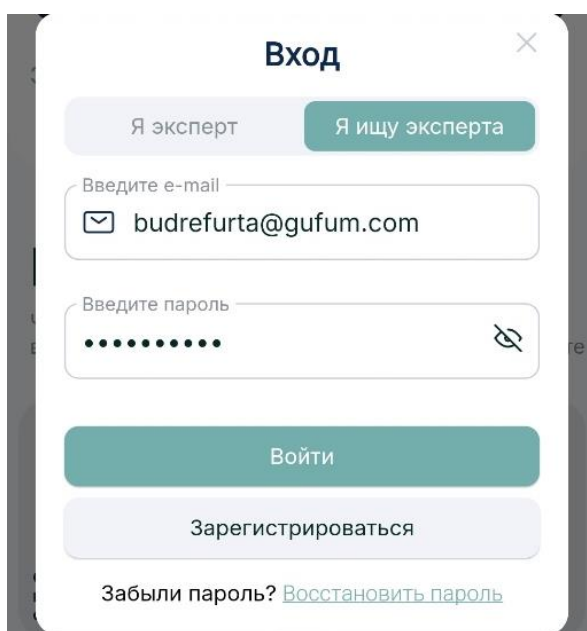


Рисунок 9 – Clokwise. Вид модального окна «Авторизация»

После авторизации пользователь попадает на экран своего профиля. Пользователю в роли «Ищу эксперта» доступны экраны чатов, поиск экспертов и созданные запросы, к которым можно получить доступ через навигационную панель внизу экрана.

Для поиска эксперта требуется перейти на страницу «Поиск экспертов», при помощи навигационной панели, нажав на кнопку с иконкой лупы. После выбрать требуемые опыт, отрасль, подотрасль, функцию и подфункцию, а также если нужно ввести слово для фильтрации. Затем

требуется нажать на кнопку «Найти эксперта». Пример заполненной страницы для поиска экспертов изображен на рисунке 10.

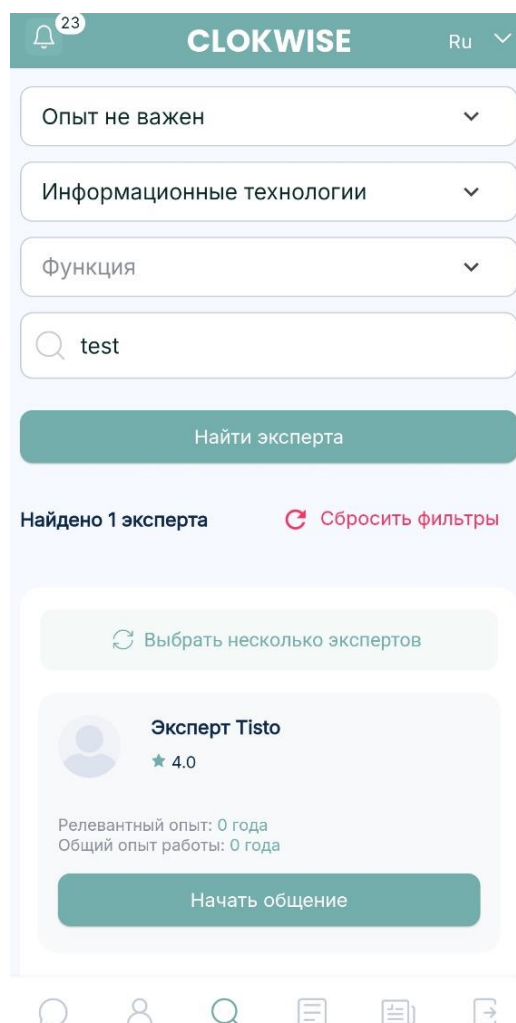


Рисунок 10 – Clokwise. Вид страницы «Поиск эксперта»

Для создания задачи требуется перейти на страницу «Запросы», где можно посмотреть созданные Вами задачи. Для создания требуется нажать на кнопку «Создать новый запрос», после чего заполнить данными запрос.

Пример страниц «Создание нового запроса» с заполненными данными и «Запросы» можно увидеть на рисунке 11.

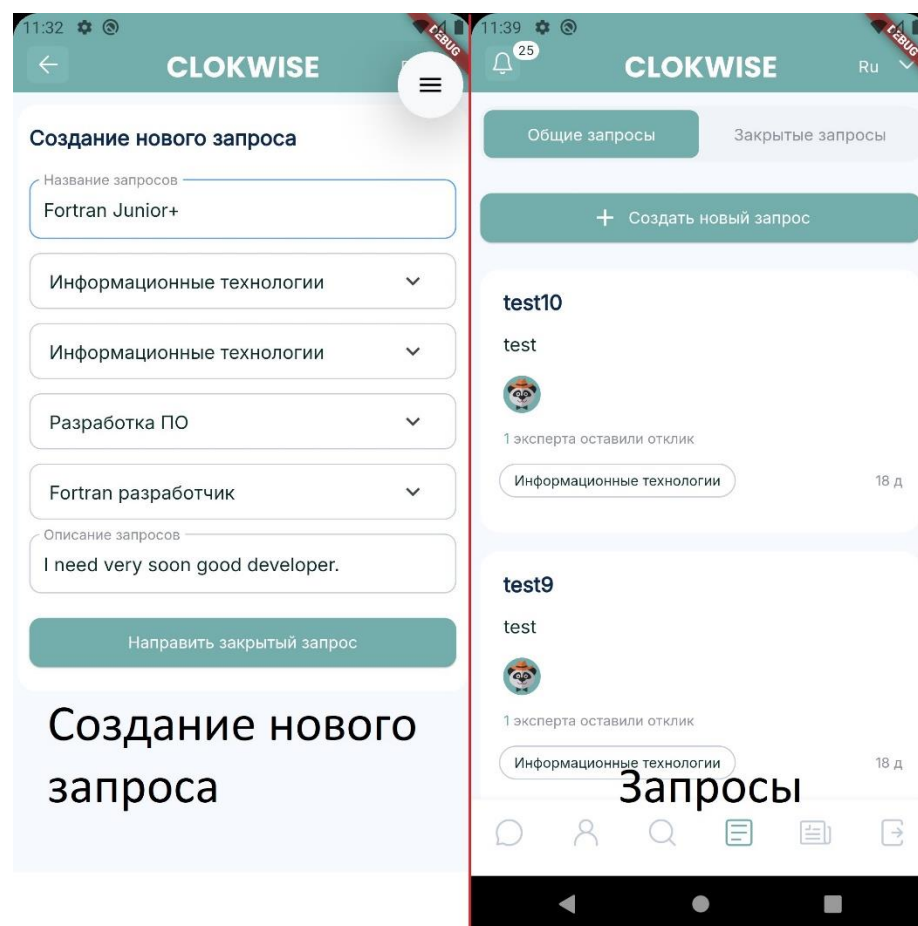


Рисунок 11 – Clokwise. Вид страниц «Создание нового запроса» и «Запрос»

Для перехода на страницу с чатами требуется нажать на навигационной панели на кнопку с иконкой облачка сообщения. На странице с чатами доступны чат с поддержкой, чаты с заказчиками. Чаты сортируются по последнему сообщению. Чат, у которого присутствует галочка и время означает, что чат был прочитан. При нажатии на чат, откроется чат с выбранным пользователем.

Пример страниц с чатами и с чатом пользователя можно увидеть на рисунке 12.

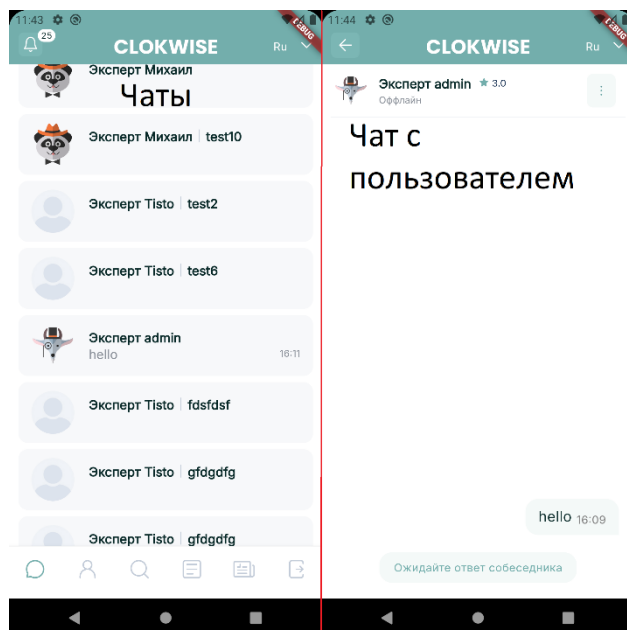


Рисунок 12 – Clokwise. Вид страниц «Чаты» и «Чат с пользователем»

Пользователю в роли «Я эксперт» доступны только экраны чатов и профиля. Для смены роли требуется зайти на страницу профиля и в верхней части страницы выбрать роль. Переключатель ролей можно увидеть на странице «Профиль», изображенной на рисунке 13.

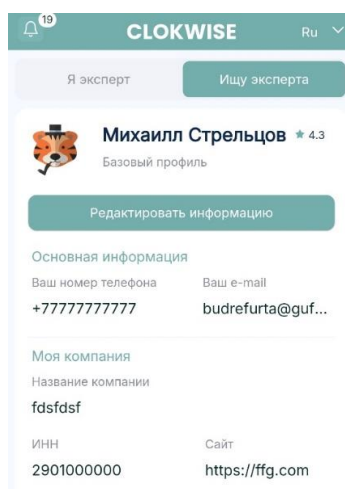


Рисунок 13 – Clokwise. Вид страницы «Профиль»

