

Пермский филиал федерального государственного  
автономного образовательного учреждения высшего образования  
«Национальный исследовательский университет  
«Высшая школа экономики»

*Факультет социально-экономических и компьютерных наук*

Князев Арсений Алексеевич

**РАЗРАБОТКА PWA ДЛЯ ПОИСКА ПОПУТЧИКОВ В ПУТЕШЕСТВИЯ**

*Курсовая работа*

студента образовательной программы «Разработка информационных систем  
для бизнеса» по направлению подготовки 09.03.04 *Программная инженерия*

Руководитель:

Старший преподаватель кафедры ИТБ  
Волков С.А.

Пермь, 2024 год

# Содержание

Введение . . . . .	3
1 Анализ предметной области для формирования требований . . . . .	4
1.1 Обоснование актуальности и практической значимости . . . . .	4
1.2 Интервью с пользователями . . . . .	6
1.3 Анализ конкурентов . . . . .	6
1.4 Макетирование интерфейса . . . . .	8
1.5 Выбор средств и технологий разработки . . . . .	9
1.5.1 Клиентская часть . . . . .	9
1.5.2 Серверная часть . . . . .	10
1.5.3 База данных . . . . .	11
2 Проектирование базы данных . . . . .	12
2.1 Связи между атрибутами . . . . .	12
2.2 Приведение к первой нормальной форме . . . . .	12
2.3 Приведение ко второй нормальной форме . . . . .	13
2.4 Приведение к третьей нормальной форме . . . . .	15
3 Разработка приложения . . . . .	16
3.1 Серверная часть . . . . .	17
3.1.1 Создание базы данных . . . . .	17
3.1.2 CRUD-операции . . . . .	17
3.1.3 Авторизация . . . . .	18
3.1.4 Обмен сообщениями . . . . .	19
3.1.5 Запросы для форм . . . . .	20
3.2 Клиентская часть . . . . .	22
3.2.1 Главная страница . . . . .	22
3.2.2 Авторизация . . . . .	26
3.2.3 Чаты . . . . .	28
3.2.4 Чат с пользователем . . . . .	29
3.2.5 Профиль . . . . .	31
3.2.6 Профиль другого пользователя . . . . .	36
Список литературы . . . . .	37
Приложение А. Диаграмма User Flow . . . . .	38
Приложение Б. Диаграмма базы данных . . . . .	39
Приложение В. Создание таблиц . . . . .	40

## **Введение**

В повседневной жизни люди всё больше и больше начали полагаться на общение и знакомства через социальные сети, мессенджеры и другие интернет ресурсы [1]. Поэтому и поиск новых знакомств также перетекает в интернет.

Так-же в России ежегодно растет доля внутреннего туризма [2].

Именно эти тренды были основополагающими при выборе темы моего прикладного проекта. Мной было решено совместить эти два тренда в одну идею. Этой идеей стало веб-приложение для поиска попутчиков для путешествий.

Такое приложение подстегнет людей, у которых нет знакомых, с которыми можно отправиться в путешествие, к путешествиям, а также позволит найти новые знакомства.

Объектом автоматизации являются человеческие отношения.

Цель работы – создание информационной системы обеспечивающей пользователям удобный инструмент для поиска попутчиков, с целью совместных путешествий.

# 1 Анализ предметной области для формирования требований

Анализ решено было решено разбить на 5 частей:

1. Подтверждение практической значимости и надобности в продукте.
2. Интервью с предполагаемыми пользователями и формирование сценариев взаимодействия.
3. Анализ конкурентов.
4. Макетирование интерфейса.
5. Выбор средств и инструментов для разработки.

Все этапы анализа должны проходить поочередно, потому что каждый последующий этап зависит от результата предыдущего.

## 1.1 Обоснование актуальности и практической значимости

Анализ было решено начать с подтверждения надобности пользователей в реализации продукта. Для этого воспользовался сервисами Google Trends и Яндекс ВордСтат, которые предлагают получить статистику поисковых запросов по ключевым словам.

Результаты запроса «Поиск попутчиков на отдых» представлены на рисунке 1.1.

Период ↓	Число запросов
март 2023	5 079
апрель 2023	4 774
май 2023	6 903
июнь 2023	8 752
июль 2023	9 917
август 2023	9 241
сентябрь 2023	6 515
октябрь 2023	4 514
ноябрь 2023	4 113
декабрь 2023	3 833
январь 2024	4 761
февраль 2024	4 155
март 2024	4 493

*Рисунок 1.1 – Результаты Яндекс ВордСтат*

Google Trends не дает доступа к информации о количестве запросов, поэтому его данные не были учтены в анализе, так как не представляется возможным как-либо адекватно интерпретировать данные полученные от этого сервиса. Яндекс ВордСтат в свою очередь предоставляет полные метрики по запросам.

В среднем количество запросов в месяц составляет около 6 тысяч, самые пикиовые значения приходятся на июнь, июль и август, периоды, когда люди больше отправляются в путешествия.

Из полученных данных мы понимаем, что на рынке существует потребность в подобном продукте и люди регулярно ищут подобные сервисы.

Еще одним подтверждением актуальности и практической значимости будущего продукта, являются отзывы пользователей похожих продуктов на площадках Google Play и App Store. Отзывы пользователей не только демонстрируют потребность в подобном продукте, но также показывают, что существующие продукты не закрывают потребности пользователей в связи со своей недоработанностью, малой функциональностью и плохой работоспособностью. Комментарии пользователей приложений конкурентов представлены на рисунках с 1.2 по 1.7.

★★★ 8 марта 2023 г.

Идея хорошая, но задумка сырья. Визуально не очень выглядит интерфейс, и по факту он так же не особо хорошо работает. В тегах о себе мало вопросов и вариантов ответа, рассказ о себе что на сайте, что в приложении виден не до конца и его никак не раскрыть. Поиск попутчиков в приложении отдельная беда. Выставила даты, всех просмотрела. Захотела увидеть в принципе кто куда и когда хочет поехать и тут сюрприз-даты не убираются. Выставила даты с сегодняшнего числа по осень - "ничего не найдено" и тд

*Рисунок 1.2 – Комментарий пользователя*

★★★★★ 24 декабря 2023 г.

Всё очень плохо работает. Геолокацию определяет неправильно, исправить не даёт. Кнопки обратной связи не активны.

*Рисунок 1.3 – Комментарий пользователя*

★★★★★ 28 января 2023 г.

Приложение работает плохо. Каждый день приходится менять пароль в аккаунте. Сколько не писала с службу поддержки все без толку.

*Рисунок 1.4 – Комментарий пользователя*

Требует доработки  
★★☆☆☆

3 года назад  
mihalis3000

Задумка хорошая, исполнение плохое, второй день модератор проверяет объявление.

*Рисунок 1.5 – Комментарий пользователя*

Не работает ни сайт ни приложение!

★★☆☆☆

3 года назад

красотка 🙄

Не работает ни сайт. Ни приложение. Задолбали

*Рисунок 1.6 – Комментарий пользователя*

Хорошее приложение, полезное 4 года назад  
★★★★★ Евгений Гальцман

Очень полезное приложение для тех кто собирает группу, например для гидов или экскурсоводов, яхтсменов. Радует то, что оно бесплатно. Нашёл несколько попутчиков для своих интересных путешествий

*Рисунок 1.7 – Комментарий пользователя*

## 1.2 Интервью с пользователями

Далее были проведены интервью с потенциальными пользователями будущего приложения. На основе проведенных интервью были сформированы сценарии взаимодействия с приложением на основе фреймворка Jobs To Be Done. Сценарии взаимодействия представлены в таблице 1.

Таблица 1 – Jobs To Be Done

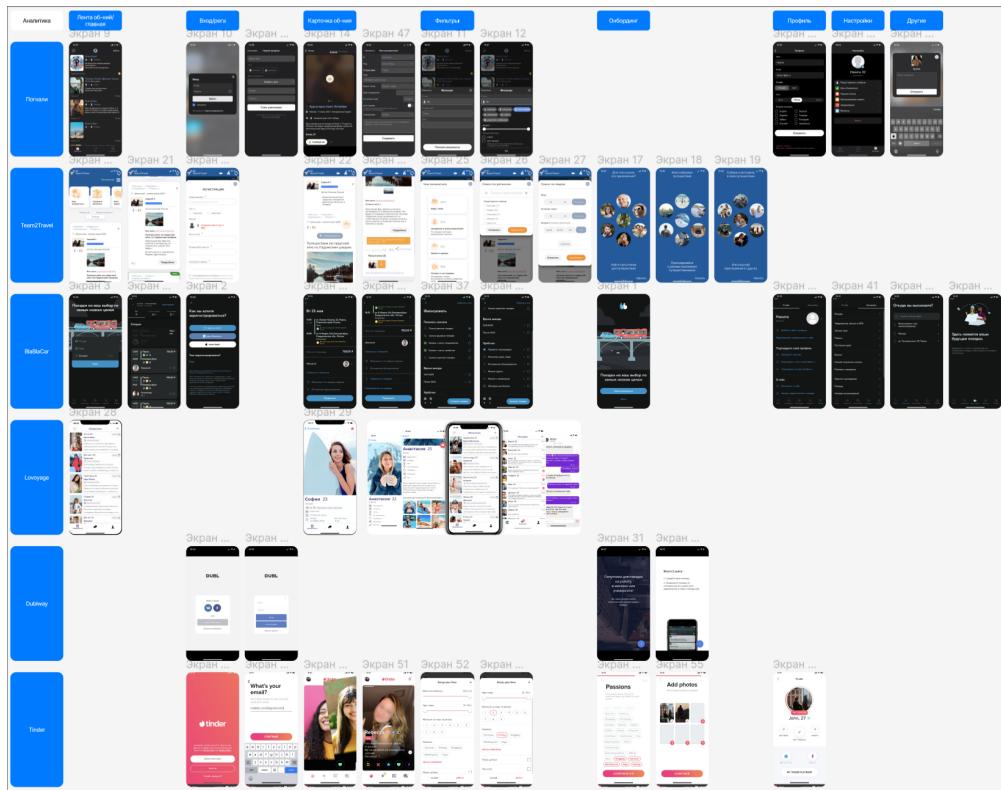
Ситуация	Мотивация	Желаемый результат
Когда я хочу отправиться в путешествие в компании, но не имею человека, который сможет поехать со мной	Я хочу найти человека или компанию, которые бы разделяли мои интересы и ценности	Чтобы поездка была наполнена положительными эмоциями, и я нашел новые знакомства
Когда я хочу отправиться в путешествие в компании, но не имею человека, который сможет поехать со мной	Я хочу найти человека или компанию, которые бы разделяли мои интересы и ценности в современном формате и не просматривать кучу странных объявлений	Чтобы я быстро и весело нашел себе попутчика
Когда я хочу поехать в путешествие, но не имею большого бюджета для него	Я хочу найти человека или компанию, с которыми можно разделить часть расходов	Чтобы путешествие получилось бюджетным, но еще более веселым и запоминающимся

По итогам проведенных интервью были выделены 3 задачи, которые пользователи хотят выполнить при помощи приложения по поиску попутчиков в путешествия. Эти задачи помогут смоделировать интерфейс приложения максимально лаконичным и удобным именно под нужды пользователей.

## 1.3 Анализ конкурентов

Следующим этапом стал анализ конкурентов. Для этого было выбрано 6 приложений прямых и косвенных конкурентов, были выделены все экраны для того, чтобы

проанализировать функционал имеющийся в этих приложениях. Экраны приложений конкурентов представлены на рисунке 1.8.



**Рисунок 1.8 – Экраны прямых и косвенных конкурентов**

Приложения «Погнали», «Team2Travel» и «Lovoyage» имеют схожие экраны главной страницы, она выглядит как доска объявлений со списком всех доступных пользователей. В остальном экраны приложений схожи и обладают похожим функционалом.

Решение с доской объявлений не самое подходящее для такого приложения, так как мы живем в «эпоху рекомендательных алгоритмов», где буквально в каждом приложении или на каждом сайте есть система рекомендаций на основе данных о пользователе, в пример можно привести «TikTok», не только одну из самых популярных, но также самую быстро растущую социальную сеть на данный момент [3], которая поставила рекомендательные алгоритмы во главу приложения, что стало одной из ключевых причин его успеха.

Также в пример можно привести приложение для знакомств «Tinder», которое тоже обладает большой популярностью в своей сфере. Там рекомендательные алгоритмы «решают» какой пользователь попадется вам на основе статистики и метрик.

В нашем же случае мы возьмем модель приложения «Tinder», так как оно доказало эффективность такого решения своей популярностью. Пользователь будет указывать свои интересы и цели на поездку, а уже на основе этих данных пользователю будут предоставляться рекомендуемые кандидаты.

## 1.4 Макетирование интерфейса

Далее была сформирована схема user flow, см. ПРИЛОЖЕНИЕ А, ней изображен путь пользователя по приложению. Данная схема позволит спроектировать дизайн приложения, не забыв учесть какие-либо функции, а также позволит наглядно изобразить архитектуру будущего приложения.

Итогом проведенного анализа, стал макет приложения реализованный при помощи инструмента Figma. Общий вид спроектированных страниц представлен на рисунке 1.9.

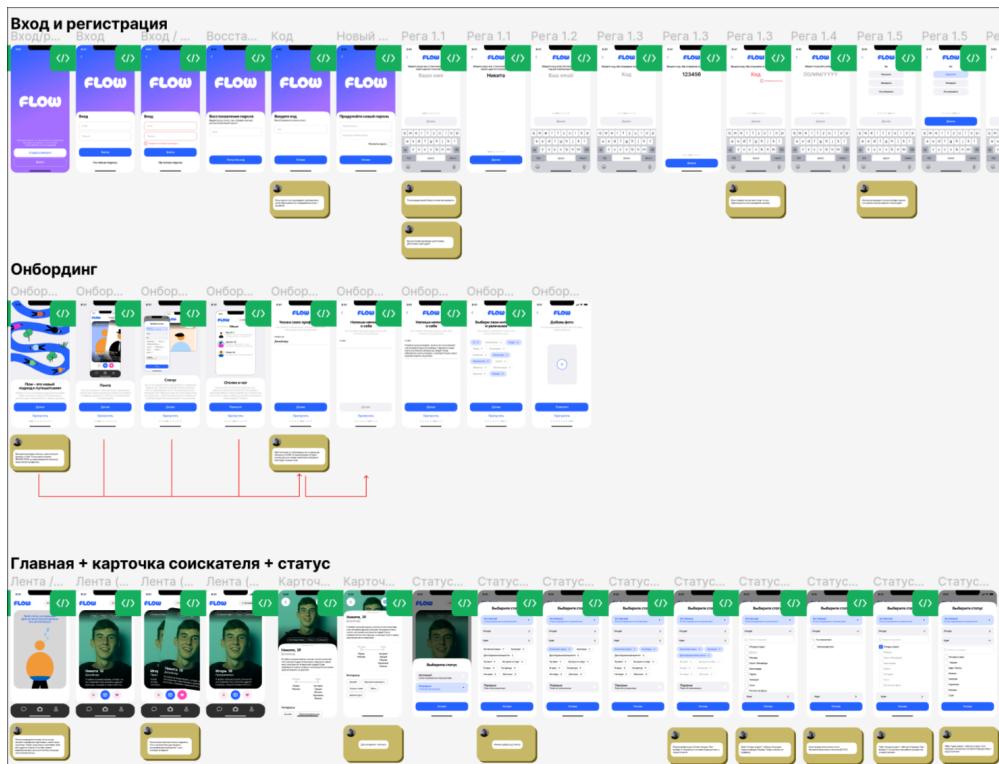


Рисунок 1.9 – Готовые страницы в Figma

Были смоделированы страницы входа и регистрации, главная, чат, профиль пользователя, и модальные окна изменения интересов и целей поездки. Примеры страниц представлены на рисунке 1.10.

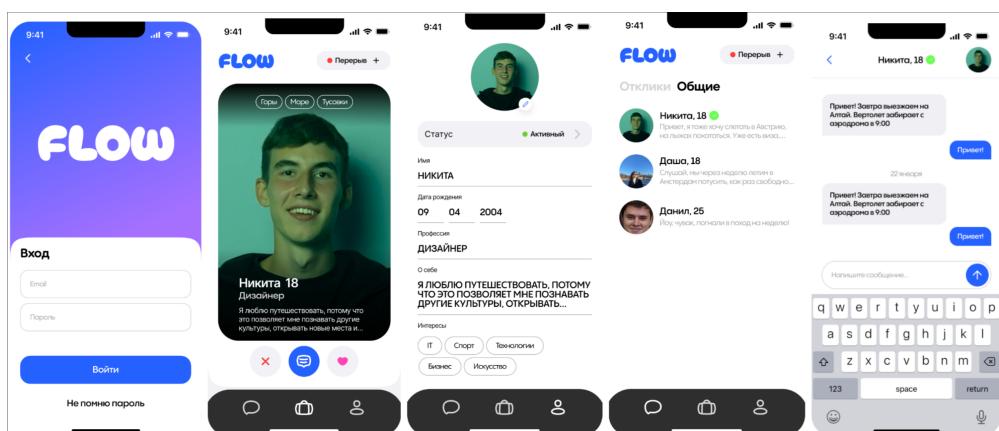


Рисунок 1.10 – Экраны приложения

## 1.5 Выбор средств и технологий разработки

Так как пользователи мобильных устройств составляют 80% времени проведенного в социальных сетях [4], то было принято решение создать веб-приложение.

Для упрощения разработки под все типы мобильных устройств, решено использовать технологию Progressive Web App (далее PWA). Технология была представлена Google для упрощения создания мультиплатформенных приложений. Технология предоставляет набор инструментов, который позволяет превратить сайт, в нативное приложение с некоторыми оговорками [5].

### 1.5.1 Клиентская часть

Для того, чтобы выбрать инструмент для создания клиентской части приложения была составлена сравнительная таблица с 4-мя кандидатами, представленная под номером 2.

Таблица 2 – Сравнение фреймворков

Критерий	React	Vue	Angular	Svelte
Сообщество	Большое	Среднее	Среднее	Маленькое
Производительность	Высокая	Высокая	Умеренная	Очень высокая
Управление состояниями	Стороннее	Встроенное	Стороннее	Встроенное
Стилизация компонентов	Сторонняя	Сторонняя	Сторонняя	Встроенная
Простота изучения	Умеренно	Легко	Сложно	Легко
Количество сторонних библиотек	Огромное	Среднее	Среднее	Малое

Следуя данным из таблицы, одним из самых лучших инструментов оказывается «Svelte», он быстрый, имеет большое количество встроенных функций, которые в других фреймворках требуют сторонние пакеты для их работы. Но, есть небольшая проблема, маленькое сообщество, что может вылиться в невозможность найти решение какой-либо проблемы и малое количество сторонних библиотек, что может привести к проблемам с написанием функционала. Но есть пара нюансов, которые поменяют взгляд на две эти проблемы.

Первое, Svelte имеет интерактивное руководство, которое проводит пользователя «за ручку» и показывает все возможности данного инструмента, начиная с чего-то простого, как переменные и реактивность и заканчивая «Context API» и другим.

Вторая проблема также, по своей сути не является проблемой, Svelte по умолчанию предоставляет все нужные инструменты для разработки: встроенное хранилище состояний, стилизация компонентов, роутинг(при помощи SvelteKIT). Говоря проще, другим фреймворкам требуется сторонние библиотеки для того, чтобы полноценно использоваться в проекте, а Svelte нет [6].

Учитывая всё вышесказанное, в проекте будет использоваться Svelte, а точнее SvelteKIT, фреймворк предоставляющий дополнительные инструменты для Svelte, такие как SSR, Роутинг и многое другое.

### 1.5.2 Серверная часть

Для серверной части было решено ограничиться знакомыми языками программирования, поэтому выбор был из фреймворков на Python и ASP.NET Core. В своей предыдущей работе я использовал ASP.NET Core, поэтому в этот раз было решено попробовать что-то новое, а значит вариантов стало еще меньше.

Самыми популярными Backend фреймворками на Python являются: FastAPI, Flask, Django. Здесь выбор был прост, было решено взять самый быстрый фреймворк, так как скорость работы для backend'а является основным показателем.

На рисунке 1.11 продемонстрировано сравнение скоростей разных фреймворков, на основе на количестве ответов в секунду при 20 запросах на 1 запрос.

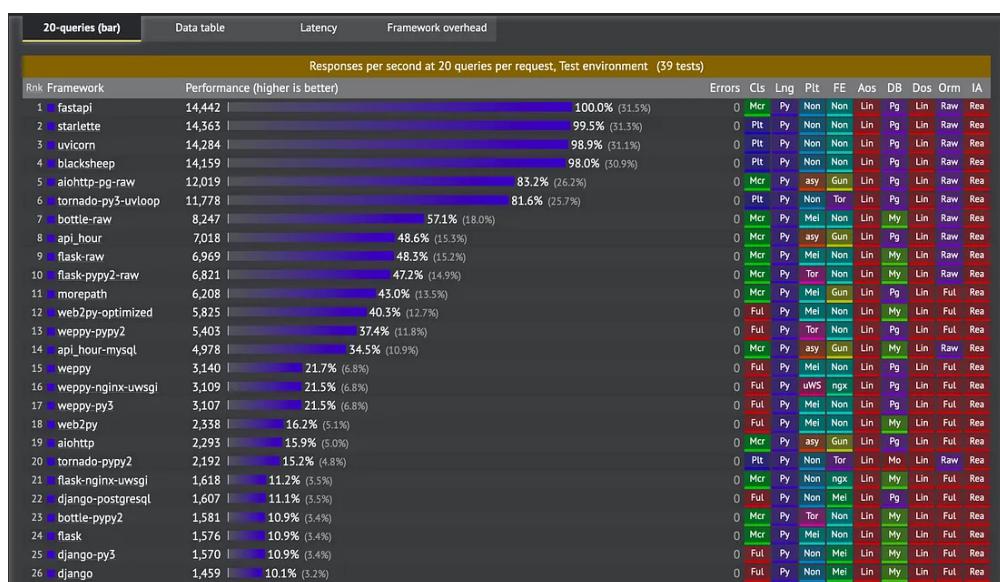


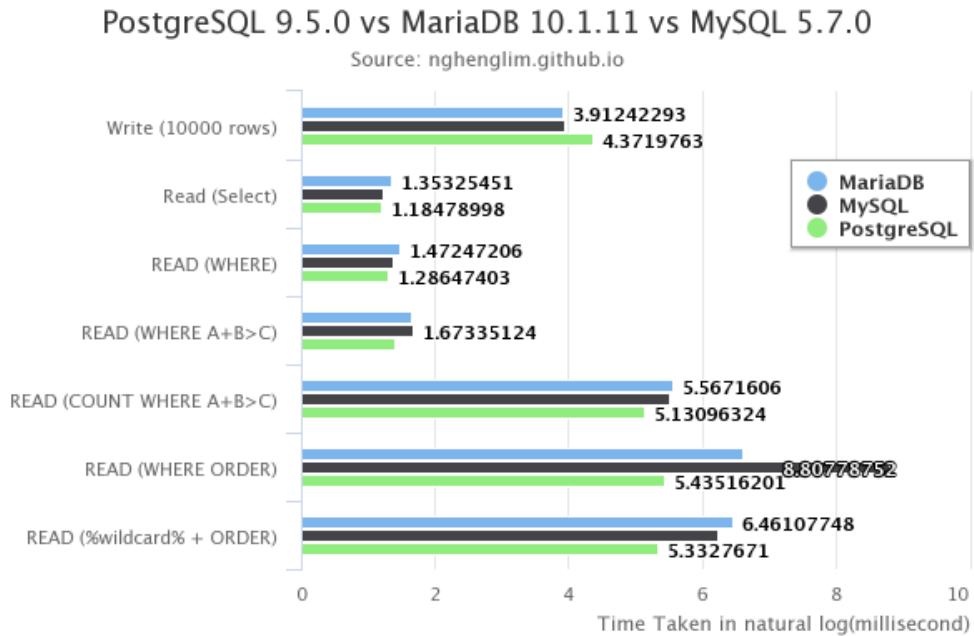
Рисунок 1.11 – Сравнение производительности backend фреймворков

Как мы видим, fastapi является самым быстрым, поэтому в проекте будет использоваться он.

### 1.5.3 База данных

В качестве базы данных будет использована SQL база данных, так в приложении будет большое количество запросов на поиск, а SQL базы данных справляются с этим лучше, так как данные и язык запросов в SQL базе данных структурированы.

Имеется большое количество SQL баз данных, их производительность примерно на одном уровне. Пример тестирования для 3-ех самых популярных SQL баз данных представлен на рисунке 1.12.



*Рисунок 1.12 – Тестирование скорости выполнения запросов*

Из тестирования видно, что какой-то значительной разницы между базами данных, не наблюдается, но в основном быстрее «PostgreSQL», поэтому будем пользоваться ей.

## 2 Проектирование базы данных

Следующим этапом работы стало проектирование базы данных.

### 2.1 Связи между атрибутами

При установлении функциональных зависимостей при проектировании БД необходимо учесть следующие связи между атрибутами:

- 1ФЗ По user\_id можно однозначно определить имя пользователя, почтовый адрес, хэш пароля, пол, текст о пользователе, профессию, дату рождения, названия файла с изображением пользователя, дату регистрации;
- 2ФЗ: По message\_id можно узнать отправителя, conversation\_id, дату отправки, дату редактирования, текст сообщения и изображение;
- 3ФЗ: По conversation\_id можно узнать дату создания диалога, дату обновления диалога, удален ли диалог, участников диалога;
- 4ФЗ: По interest\_id можно будет узнать название интереса;
- 5ФЗ: По trip\_purpose\_id можно будет узнать название цели поездки;
- 6ФЗ: По match\_id можно будет узнать id того пользователя, который заинтересовался кем-то и id пользователя в ком был заинтересован первый пользователь;
- 7ФЗ: По departure\_id можно будет узнать departure\_name;
- 8ФЗ: По arrival\_id можно будет узнать arrival\_name;
- 9ФЗ: По departure\_id и user\_id можно узнать от куда едет пользователь;
- 10ФЗ: По arrival\_id и user\_id можно узнать куда едет пользователь;
- 11ФЗ: По role\_id можно узнать название роли;
- 12ФЗ: По role\_id и user\_id можно узнать роль пользователя.

### 2.2 Приведение к первой нормальной форме

Отношение находится в 1НФ, когд все его атрибуты имеют единственно значение (атомарные атрибуты) и все кортежи уникальны (наличие РК).

В реляционной модели отношение всегда находится в первой нормальной форме по определению понятия отношение.

- user\_id
- username
- mail
- password\_hash
- birthdate
- sex
- registration\_date

- about\_text
- profession
- message\_id
- from\_id
- to\_id
- created\_at
- updated\_at
- content\_id
- message\_body
- message\_image
- match\_id
- interested\_in\_id
- interest\_id
- interest\_name
- departure\_id
- arrival\_id
- departure\_name
- arrival\_name

В соответствии с описанными выше функциональными зависимостями формируем первичный ключ отношения, который включает следующие атрибуты:

- user\_id
- message\_id
- content\_id
- match\_id
- interest\_id
- trip\_purpose\_id
- departure\_id
- arrival\_id
- role\_id

### **2.3 Приведение ко второй нормальной форме**

Отношение находится во 2НФ, если оно находится в 1НФ + отсутствует частичная функциональная зависимость неключевых атрибутов от ключа (не д.б. неключевых полей зависящих от части РК).

- user\_id определяет:
  - username
  - mail

- password\_hash
- birthdate
- sex
- registration\_date
- user\_image
- message\_id определяет:
  - conversation\_id
  - sender\_id
  - created\_at
  - updated\_at
- conversation\_id определяет:
  - is\_delete
  - created\_at
  - updated\_at
  - users
- match\_id определяет:
  - user\_id
  - interested\_in\_id
- interest\_id определяет:
  - interest\_name
- purpose\_id определяет:
  - purpose\_name
- departure\_id определяет:
  - departure\_name
- arrival\_id определяет:
  - arrival\_name
- role\_id определяет:
  - role\_name

Получаем такие отношения (таблицы):

- “users”, которая включает атрибуты: (user\_id, username, mail, password\_hash, birthdate, sex, registration\_date)
- “messages”, которая включает атрибуты: (message\_id, conversation\_id, sender\_id, created\_at, updated\_at)
- “conversations”, которая включает атрибуты: (conversation\_id, created\_at, updated\_at, is\_deleted)
- “conversation\_participant”, которая включает атрибуты: (user\_id, conversation\_id)

- “matches”, которая включает атрибуты: (match\_id, user\_id, interested\_in\_id)
- “interests”, которая включает атрибуты: (interest\_id, interest\_name)
- “user\_interest”, которая включает атрибуты: (user\_id, interest\_id)
- “trip\_purposes”, которая включает атрибуты: (purpose\_id, purpose\_name)
- “user\_trip\_purpose”, которая включает атрибуты: (user\_id, purpose\_id)
- “departures”, которая включает атрибуты: (departure\_id, departure\_name)
- “arrivals”, которая включает атрибуты: (arrival\_id, arrival\_name)
- “user\_departure”, которая включает атрибуты: (user\_id, departure\_id)
- “user\_arrival”, которая включает атрибуты: (user\_id, arrival\_id)
- “roles”, которая включает атрибуты: (role\_id, role\_name)
- “user\_role”, которая включает атрибуты: (user\_id, role\_id)

## **2.4 Приведение к третьей нормальной форме**

Отношение находится в ЗНФ, если оно находится во 2НФ + нет ФЗ между не ключевыми атрибутами (неключевые атрибуты взаимно независимы) + каждый неключевой атрибут нетранзитивно зависит от ключа.

В нашем случае третья нормальная форма уже выполнена.

После приведения ко всем нормальным формам мы формируем диаграмму базы данных см. ПРИЛОЖЕНИЕ Б.

### **3   Разработка приложения**

В этой части работы, мы рассмотрим процесс реализации приложения, начиная серверной частью и заканчивая клиентской.

По итогу мы получим клиент-серверное приложение написанное на языках JavaScript и Python, а так же проведем тестирование разработанного программного обеспечения.

### 3.1 Серверная часть

Серверная часть будет отвечать за работу с базой данных и за работу с клиентской частью – она будет выполнять запросы отправленные с клиента, получать имеющиеся данные из базы данных или добавлять новые данные, а затем отправлять ответ на клиентскую часть.

#### 3.1.1 Создание базы данных

За создание и работу с базой данных будет отвечать ORM sqlalchemy2. Данный позволяет легко создавать таблицы в sql базах данных на основе моделей написанных на python, а также позволяет легко делать запросы при помощи встроенных в него функций.

Код для создания таблиц представлен в ПРИЛОЖЕНИИ В.

Следует обратить внимание, что пользователю мы добавляем отношения с сообщениями, диалогами, интересами, целями поездок, отправлениями и прибытиями – это нужно для того, чтобы упростить нам задачу написания запросов. При работе с любой из этих таблиц, вместо того, чтобы искать в каждом запросе эти данные для каждого конкретного пользователя, мы просто можем обратиться к полю с нужными нами данными в самом пользователе избегая этих шагов.

#### 3.1.2 CRUD-операции

Первым делом были написаны CRUD (create, read, update, delete) запросы для всех таблиц: Пользователи, Сообщения, Интересы, Цели поездок, Отправлений и Прибытий. Часть CRUD запросов представлена на рисунке 3.1.

interests		
PATCH	/user/interests/edit	Edit User Interests
GET	/interests	Get Interests
POST	/interests	Add Interest
DELETE	/interests/{interest_id}	Delete Interest
trip_purposes		
GET	/trip_purposes	Get Trip Purposes
POST	/trip_purposes	Add Trip Purpose
departures		
GET	/departures	Get Departures
POST	/departures	Add Departure
DELETE	/departures/{location_id}	Delete Departure
arrivals		
GET	/arrivals	Get Arrivals
POST	/arrivals	Add Arrival
DELETE	/arrivals/{location_id}	Delete Arrival

Рисунок 3.1 – Запросы в инструменте Swagger

### 3.1.3 Авторизация

Следующим этапом стало добавление авторизации. Для авторизации будем пользоваться стандартом JSON Web Token (далее JWT). Он используется для передачи данных для аутентификации.

При авторизации или регистрации пользователю будет выдаваться два токена: access и refresh. Два токена нужно в целях безопасности.

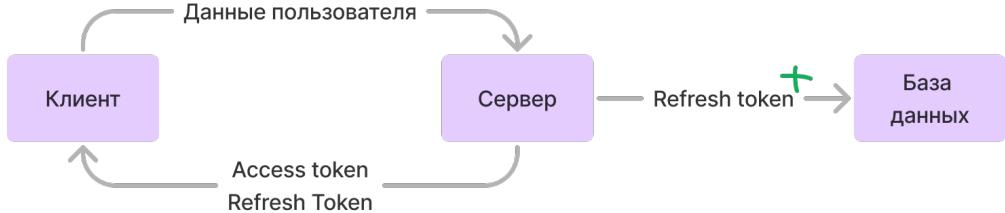
Access токен, нужен для того, чтобы подтвердить является ли человек тем, за кого себя выдает. Этот токен подписывается на сервере и на сервере же проверяется является ли он действительным. Время жизни Access токена ограничено и в нашем случае будет 30 минут.

Refresh токен, он отвечает за выпуск нового access токена, база данных будет хранить пару id пользователя и refresh для каждого авторизованного пользователя. При получении запроса на обновление access токена, сервер сверяет refresh токен из запроса и refresh токен пользователя из базы данных, если они совпадают, сервер отправляет новую пару токенов.

Диаграммы запросов входа, регистрации и обновления токенов представлены на рисунках с 3.2 по 3.4.

На рисунке 3.2 представлена диаграмма входа пользователя. Запрос работает так:

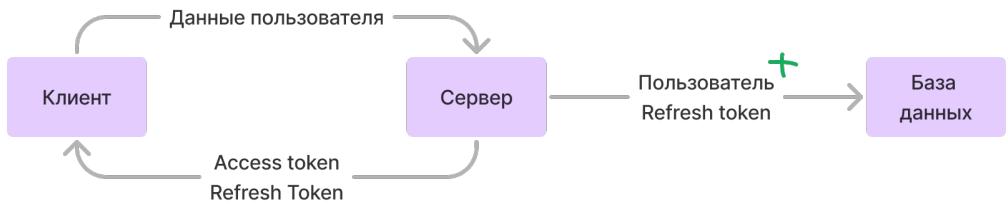
1. Пользователь вводит почту и пароль в форму на клиентской части.
2. Клиент отправляет данные пользователя на сервер.
3. Сервер сверяет пароль с тем, что хранится в базе данных.
4. Если пароль верный, то сервер отправляет пару ключей.



*Рисунок 3.2 – Диаграмма входа*

На рисунке 3.3 представлена диаграмма регистрации пользователя. Запрос работает так:

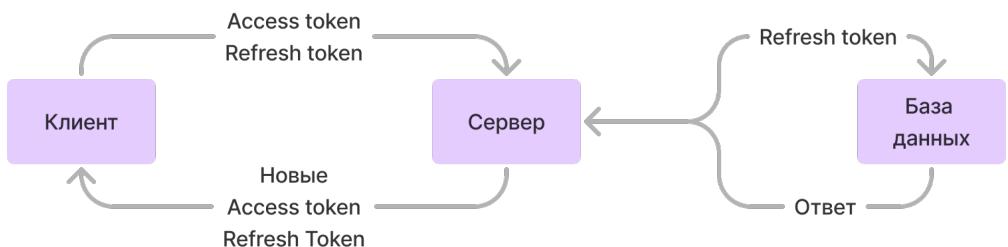
1. Пользователь вводит данные для создания аккаунта.
2. Сервер записывает данные в базу данных.
3. Сервер отправляет пару ключей.



*Рисунок 3.3 – Диаграмма регистрации*

На рисунке 3.4 представлена диаграмма обновления access токена. Запрос работает так:

1. Сервер берет access и refresh токены из http-only cookie.
2. Сервер проверяет правильность refresh токена.
3. Если refresh токен правильный, то сервер отправляет новую пару ключей и обновляет refresh токен пользователя в базе данных.



*Рисунок 3.4 – Диаграмма обновления токена*

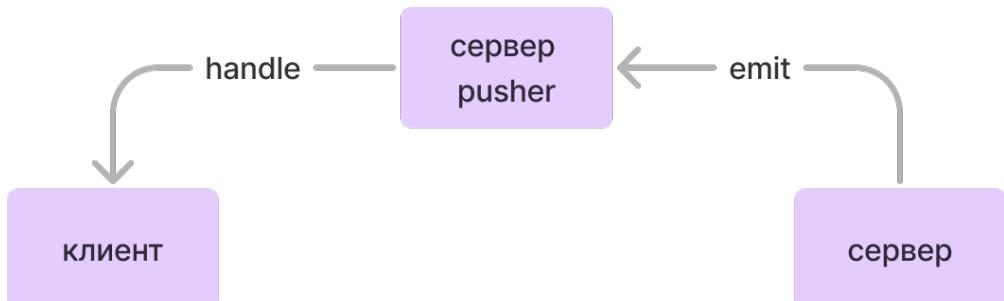
### 3.1.4 Обмен сообщениями

Далее было решено приступить к реализации обмена сообщениями. Для решения этого задачи можно использовать несколько способов.

- SSE (Server Sent Events), формируется соединение от сервера к клиенту, клиент может получать только сигналы от сервера, сам клиент не может отправлять сигналы;
- WebSockets, отдельный сервер, сигналы могут отправлять и клиент и сервер;
- long polling, на сервер отправляется запрос, который может длиться длительное время, а при превышении лимита времени отправится снова, завершившись только когда, сервер ответит, после этого сразу же отправится новый запрос.

Все варианты имеют свои плюсы и минусы, наверное кроме long polling, данный метод можно использовать только за неимением первых двух, так как он повышает нагрузку на сервер, постоянно посылая запросы.

В реализации приложения было решено использовать WebSockets, а точнее сервис «Pusher», который предоставляет максимально простое API для работы. Схема работы с «Pusher» представлена на рисунке 3.5.



*Рисунок 3.5 – Диаграмма взаимодействия с сервером «Pusher»*

При создании чата и при добавлении нового сообщения в базу данных сервер отправляет сигнал на сервер «Pusher», он же в свою очередь отправляет сигнал на подключенный к «Pusher» клиент, который обрабатывает пришедшие на него сигналы.

Пример вызова сигнала при помощи «Pusher» представлен ниже.

```

1 pusher_client.trigger(
2     f"{user.mail}",
3     "conversation:new",
4     jsonable_encoder(conversation),
5 )

```

### 3.1.5 Запросы для форм

В нашем клиентском приложении будет несколько форм, для них требуются отдельные запросы, так как они не попадают в CRUD операции реализованные ранее.

Например таким запросом будет являться форма изменения интересов, с клиента мы будем получать массив id интересов, на сервере мы должны заменить ими текущие.

```

1 @user_router.patch("/interests/edit", tags=["interests"])
2 async def edit_user_interests(
3     user: user_dependency, edit: schema.TagsEdit, db: db_dependency
4 ):
5     if not user:
6         raise HTTPException(status_code=404, detail="Not authorized")
7     user.interests = []
8     db.commit()
9     db.execute(
10         insert(models.user_interest_table).values(
11             [
12                 {"user_id": user.id, "interest_id": interest_id}
13                 for interest_id in edit.tags

```

```
14     ]
15 )
16 )
17 db.commit()
18 return {"ok": True}
```

Так как мы добавили отношение с интересами в модель пользователя ранее, то для того, чтобы удалить текущие интересы пользователя, мы можем обратиться прямиком к текущему пользователю и заменить поле `interests` пустым массивом, для того чтобы удалить их. А затем в цикле мы добавляем в таблицу `user_interest` все новые интересы.

Пользуясь такой логикой мы реализуем и другие формы, логика которых будет рассмотрена в части реализации клиентской части.

## 3.2 Клиентская часть

Реализация клиентской части началась с создания всех основных элементов интерфейса: кнопок, элементов поиска и прочего.

Далее было решено перейти к созданию страниц и логики форм.

### 3.2.1 Главная страница

Первой стала главная, ее итоговый вид представлен на рисунке 3.6.



Рисунок 3.6 – Главная страница

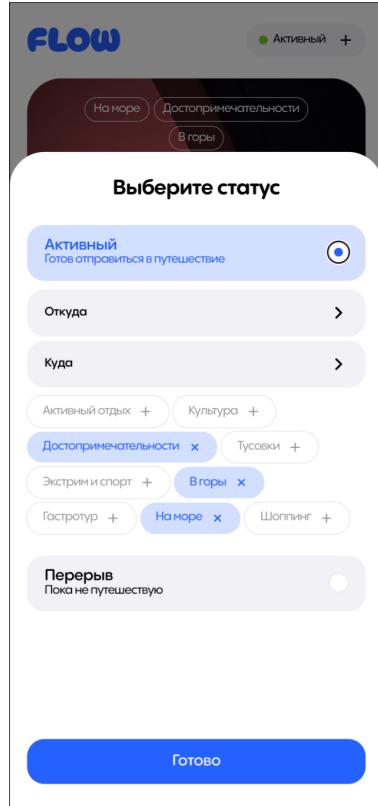
На главной странице имеются карточки пользователей и кнопки для действий. При загрузке главной страницы, мы загружаем данные пользователя, список всех возможных интересов, целей поездки, отправлений, прибытий. Для загрузки данных, мы используем функцию `load` из SvelteKIT. Данная функция выполняет указанный нами код при загрузке страницы. В нашем случае это выполнение запросов. Код функции представлен ниже.

```
1 export const load: LayoutServerLoad = async ({ fetch }) => {
2   const profile: UserWStatus = await (await fetch('http://nginx/api/user/profile'))
3     .json();
4   const trip_purposes: TripPurpose[] = await (await fetch('http://nginx/api/
5     trip_purposes')).json();
6   const interests: Interest[] = await (await fetch('http://nginx/api/interests'))
7     .json();
8   const departures: Departure[] = await (await fetch('http://nginx/api/
9     departures')).json();
```

```
6  const arrivals: Arrival[] = await (await fetch('http://nginx/api/arrivals')).  
7    json();  
8  
9  
10 const data = {  
11   user: profile,  
12   interests: interests,  
13   trip_purposes: trip_purposes,  
14   departures: departures,  
15   arrivals: arrivals,  
16   statusForm,  
17   profileForm,  
18   interestsForm,  
19   likeForm,  
20   avatarForm  
21 };  
22  
23 return data;  
24 };
```

При нажатии кнопки ”нравится” или ”не нравится”, а так же при пролистывании карточки влево или вправо на сервер отправляется запрос на добавление пользователя с карточки в таблицу ”matches”.

Так же с главной страницы мы можем получить доступ к модальному окну изменения статуса. Модальное окно представлено на рисунке 3.7.



*Рисунок 3.7 – Модальное окно изменения статуса*

Данное окно является формой, в которой пользователь выбирает текущий статус (активный, неактивный), если пользователь активен, то он может изменить свои цели поездки, выбрать места из которых он готов отправиться и соответственно куда готов отправиться [7].

Всего пользователь может выбрать не более 3 локаций из откуда и 5 локаций куда, и там, и там, пользователь может выбрать вариант все, тогда будет считаться, что пользователь выбрал все возможные варианты. Целей поездки можно выбрать не более 3.

Для реализации логики форм, используется пакет «SuperForms», он позволяет легко реализовать валидацию данных, вывод ошибок при неправильном заполнении форм и так же упрощает отправку данных из формы на сервер.

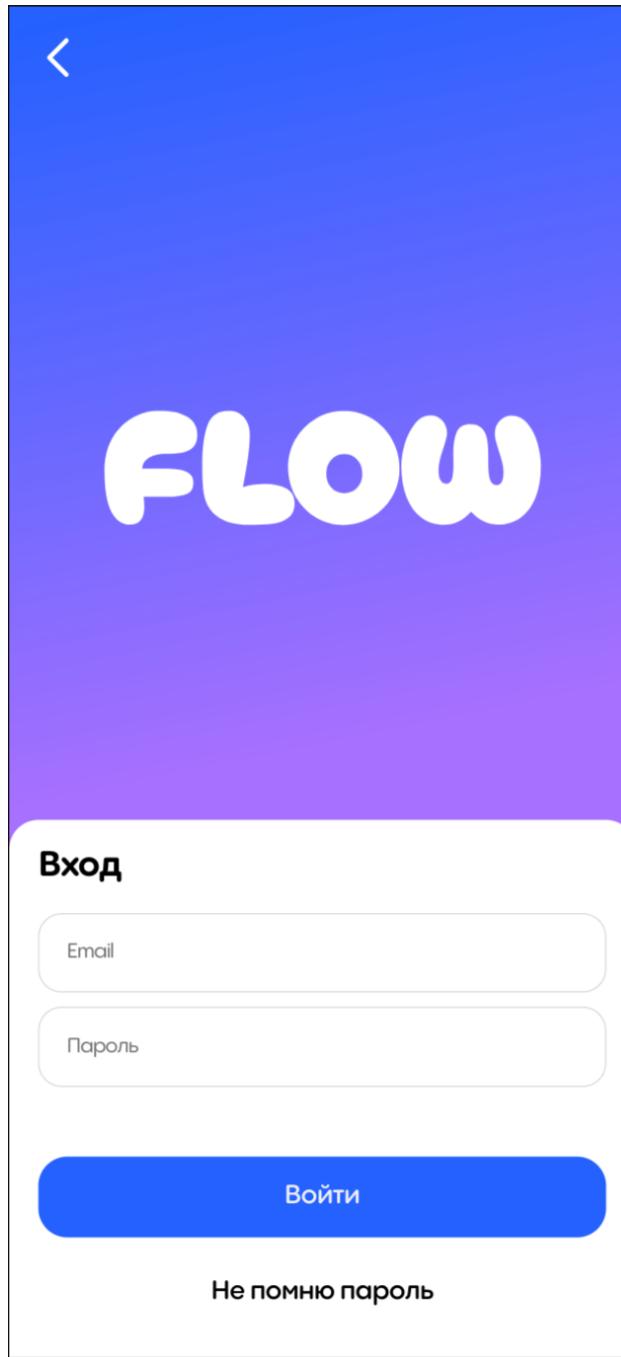
При закрытии формы нажатием на кнопку готово или при нажатии за пределами модального окна, данные из формы отправляются на сервер, при помощи инструмента form actions из SvelteKIT [8]. Плюс использования actions, в том, что даже если JavaScript у пользователя будет выключен, они будут продолжать работать. Код запроса представлен ниже.

```
1 export const actions = {  
2   //...  
3   update_status: async ({ request, fetch }) => {
```

```
4  const statusForm = await superValidate(request, zod(statusSchema));
5  if (!statusForm.valid) return fail(400, { statusForm });
6  await fetch("http://nginx/api/user/status_data/edit", {
7    method: "PATCH",
8    headers: {
9      "Content-Type": "application/json",
10     },
11     body: JSON.stringify(statusForm.data),
12   });
13 },
14 //...
15 } satisfies Actions;
```

### 3.2.2 Авторизация

Страница авторизации представлена на рисунке 3.8.



*Рисунок 3.8 – Страница авторизации*

Мы имеем простую форму для авторизации, всего 2 поля, при подтверждении мы вызываем action входа. Код представлен ниже.

```
1 export const actions = {  
2   default: async ({ request, cookies }) => {  
3     const loginForm = await superValidate(request, zod(loginSchema));  
4     if (!loginForm.valid) return fail(400, { loginForm });  
5  
6     const urlParams = new URLSearchParams();
```

```

7   urlParams.append("username", loginForm.data.mail);
8   urlParams.append("password", loginForm.data.password);
9
10  const requestOptions = {
11    method: "POST",
12    headers: {
13      "Content-Type": "application/x-www-form-urlencoded",
14    },
15    body: urlParams,
16  };
17
18  const response = await fetch('http://nginx/api/auth/login', requestOptions);
19
20  if (response.status == 200) {
21    const data = await response.json();
22    cookies.set(
23      "access_token",
24      decodeURIComponent(`Bearer ${data.access_token}`),
25      { path: "/" },
26    );
27    cookies.set(
28      "refresh_token",
29      decodeURIComponent(`Bearer ${data.refresh_token}`),
30      {
31        path: "/",
32      },
33    );
34    throw redirect(302, "/");
35  } else {
36    return setError(loginForm, "Invalid email or password");
37  }
38},
39} satisfies Actions;

```

### 3.2.3 Чаты

Итоговый вид страницы чатов представлен на рисунке 3.9.

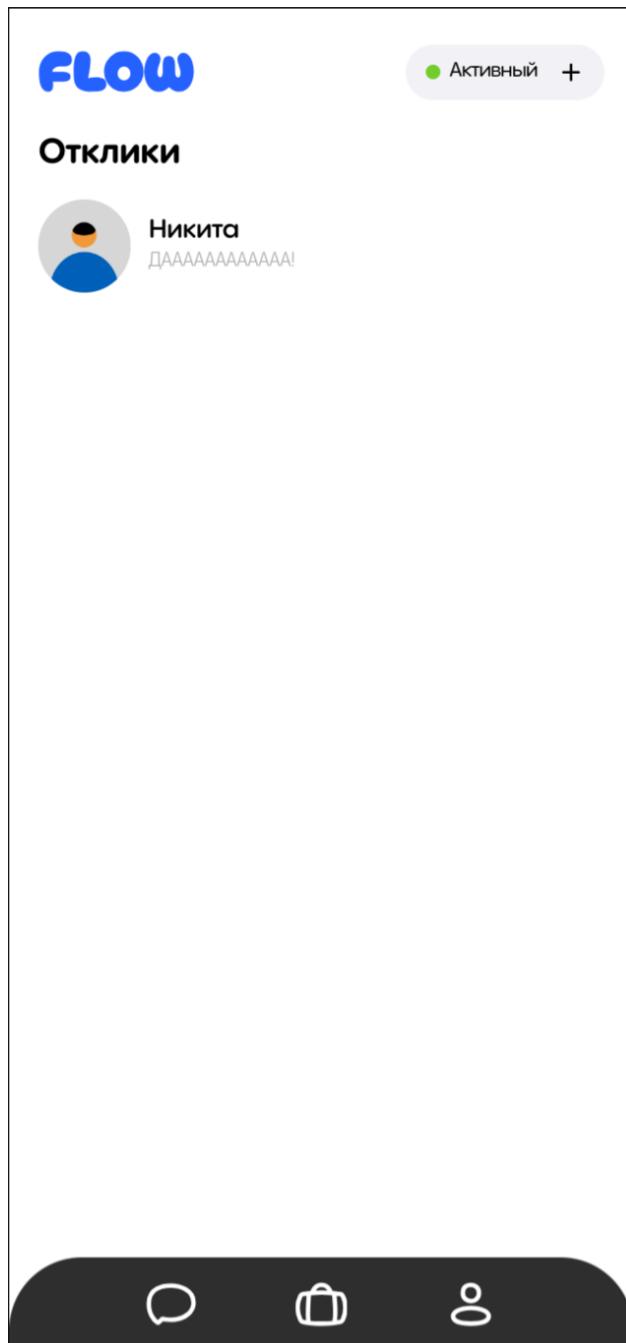
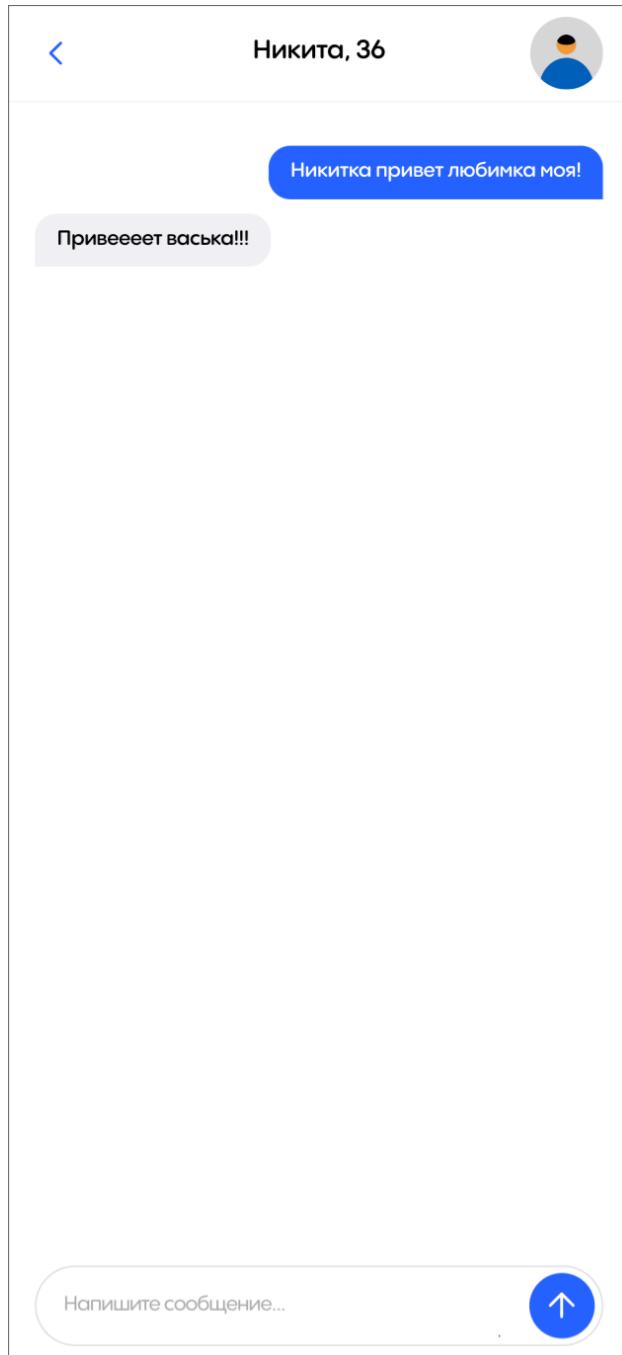


Рисунок 3.9 – Страница чатов

На данной странице отображаются все чаты пользователя, мы получаем их из load функции. На данной странице мы подключаемся к «Pusher» и ожидаем сигнала на появление нового чата или обновления старого. Если такое происходит, мы вызываем функцию invalidateAll, которая перезагружает данные из load функции и мы получаем обновленный данные о чатах [9].

### 3.2.4 Чат с пользователем

Итоговый вид страницы представлен на рисунке 3.10.



*Рисунок 3.10 – Диалог пользователей*

При загрузке данной страницы, мы берем используем функцию `load` и делаем запрос на сервер, чтобы получить данные о данном диалоге. После того как мы получаем данные, создаем элементы сообщений.

Так же мы подключаемся к «Pusher» и ожидаем новых сообщений другого пользователя. Если приходит новое сообщение, мы добавляем его в массив сообщений из которого формируются HTML элементы сообщений. Код подключения к «Pusher» представлен ниже.

```

1 const newHandler = (data: Message) => {
2   messages = [...messages, data];
3   setTimeout(() => {
4     scrollToBottom(messagesContainer);
5   }, 100);
6 };
7
8 onMount(() => {
9   pusherClient.subscribe($page.data.user.mail);
10  pusherClient.bind("message:new", newHandler);
11
12  scrollToBottom(messagesContainer);
13 });

```

Поле ввода является формой, при ее подтверждении мы вызываем form action, для отправки запроса на добавление сообщения на сервер. Код представлен ниже.

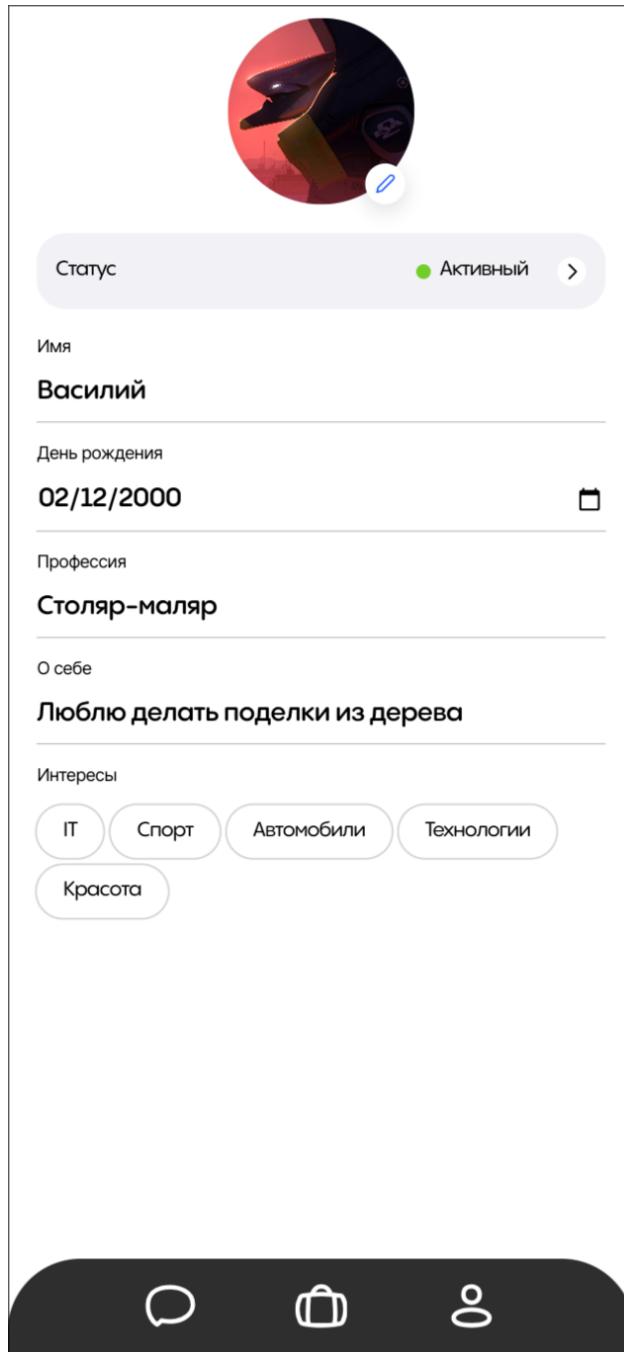
```

1 export const actions = {
2   send_message: async ({ request, fetch, params }) => {
3     const messageForm = await superValidate(request, zod(messageSchema));
4     if (!messageForm.valid) return fail(400, { messageForm });
5     await fetch(`http://nginx/api/chats/${params.conversation_id}`, {
6       method: "POST",
7       headers: {
8         "Content-Type": "application/json",
9       },
10      body: JSON.stringify({
11        content: messageForm.data.message,
12      }),
13    });
14  },
15 } satisfies Actions;

```

### 3.2.5 Профиль

Итоговый вид страницы профиля представлен на рисунке 3.11.



*Рисунок 3.11 – Страница профиля*

Данная страница является самой нагруженной по формам. В сумме здесь 4 формы:

- Форма статуса из шапки;
- форма с данными пользователя;
- форма с интересами пользователя;
- форма с изображением профиля.

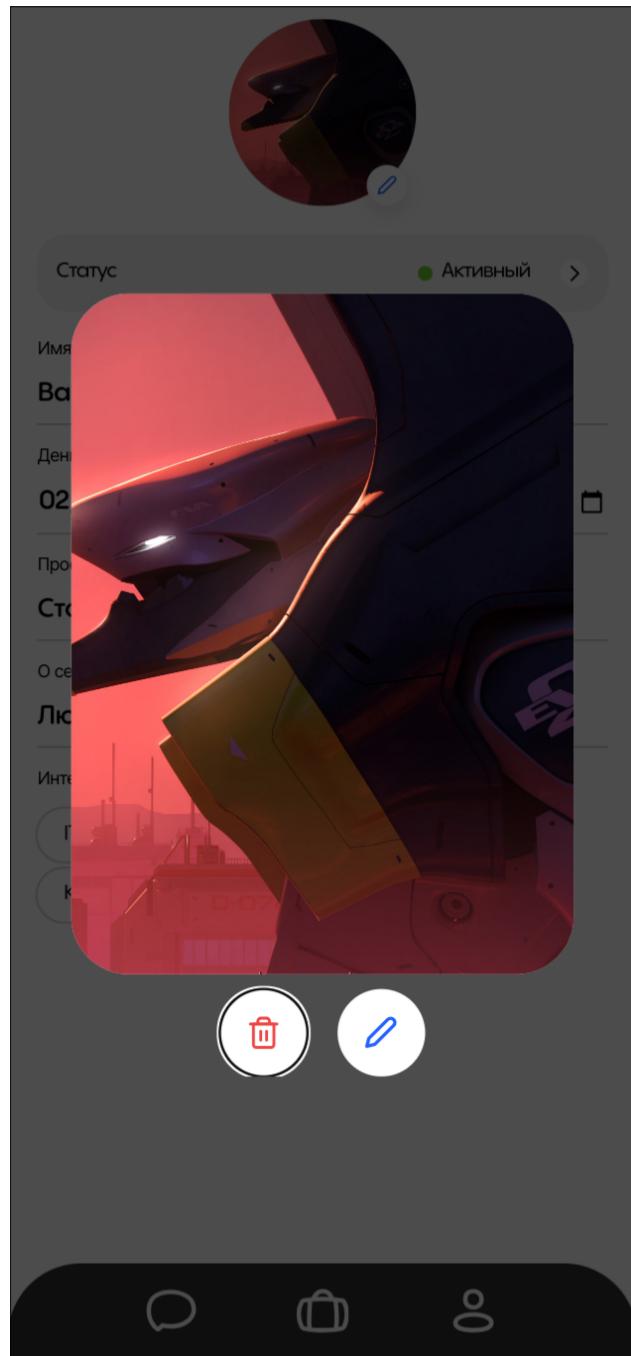
Форма с данными пользователя самая простая, она включает в себя 4 поля ввода и при потере фокуса на любом из полей или при переходе на другую страницу, отправляет запрос на обновление данных пользователя.

Форма с интересами представляет собой список всех возможных интересов, пользователь может выбрать до 5 интересов. Форма представлена на рисунке 3.12.

The screenshot shows a mobile application interface. At the top, there is a circular profile picture of a person in a green environment. Below it, a status bar indicates 'Статус' (Status) and 'Активный' (Active) with a green dot. The main profile area contains fields for 'Имя' (Name) with the value 'Василий', 'День рождения' (Birthday) with the value '02/12/2000', and 'Профессия' (Profession) with the value 'Столяр-маляр'. There is also a section labeled 'О себе' (About me) with the placeholder 'Пиши свое описание...' (Write your description...). A modal window titled 'Интересы' (Interests) is displayed in the foreground. It lists several interests with '+' and 'x' buttons: 'IT x', 'Автомобили x', 'Спорт x', 'Мода +', 'Кулинария +', 'Алкоголь +', 'Искусство +', 'Технологии x', 'Наука +', 'Финансы +', 'Мототехника +', 'Красота x', and 'Бизнес +'. At the bottom of the modal is a large blue 'Готово' (Done) button.

*Рисунок 3.12 – Форма изменения интересов*

Форма с изображением профиля имеет 2 кнопки, для редактирования и удаления фотографии профиля. Она представлена на рисунке 3.13.



*Рисунок 3.13 – Форма изменения изображения профиля*

При загрузке фото, появляются 2 другие кнопки для подтверждения или отмены загрузки фото. Вид данного окна представлен на рисунке 3.14.



Рисунок 3.14 – Форма изменения изображения профиля

Все эти формы так же используют «SuperForms» и form actions. Код представлен ниже.

```
1 export const actions = {
2   update_interests: async ({ request, fetch }) => {
3     const interestsForm = await superValidate(request, zod(interestsSchema));
4     if (!interestsForm.valid) return fail(400, { interestsForm });
5     await fetch("http://nginx/api/user/interests/edit", {
6       method: "PATCH",
7       headers: {
```

```

8         "Content-Type": "application/json",
9     },
10    body: JSON.stringify({
11      tags: interestsForm.data.user_interests,
12    }),
13  );
14},
15 update_status: async ({ request, fetch }) => {
16   const statusForm = await superValidate(request, zod(statusSchema));
17   if (!statusForm.valid) return fail(400, { statusForm });
18   await fetch("http://nginx/api/user/status_data/edit", {
19     method: "PATCH",
20     headers: {
21       "Content-Type": "application/json",
22     },
23     body: JSON.stringify(statusForm.data),
24   });
25},
26 update_profile: async ({ request, fetch }) => {
27   const profileForm = await superValidate(request, zod(profileSchema));
28   if (!profileForm.valid) return fail(400, { profileForm });
29   await fetch("http://nginx/api/user/", {
30     method: "PATCH",
31     headers: {
32       "Content-Type": "application/json",
33     },
34     body: JSON.stringify(profileForm.data),
35   });
36},
37 update_avatar: async ({ request, fetch }) => {
38   const formData = await request.formData();
39   const avatarForm = await superValidate(
40     { file: formData.get("file") as File },
41     zod(avatarSchema),
42   );
43   if (!avatarForm.valid) {
44     return fail(400, { avatarForm });
45   }
46   await fetch("http://nginx/api/user/image", {
47     method: "PATCH",
48     body: formData,
49   });
50},
51 } satisfies Actions;

```

### 3.2.6 Профиль другого пользователя

Итоговый вид страницы профиля другого пользователя представлен на рисунке 3.15.

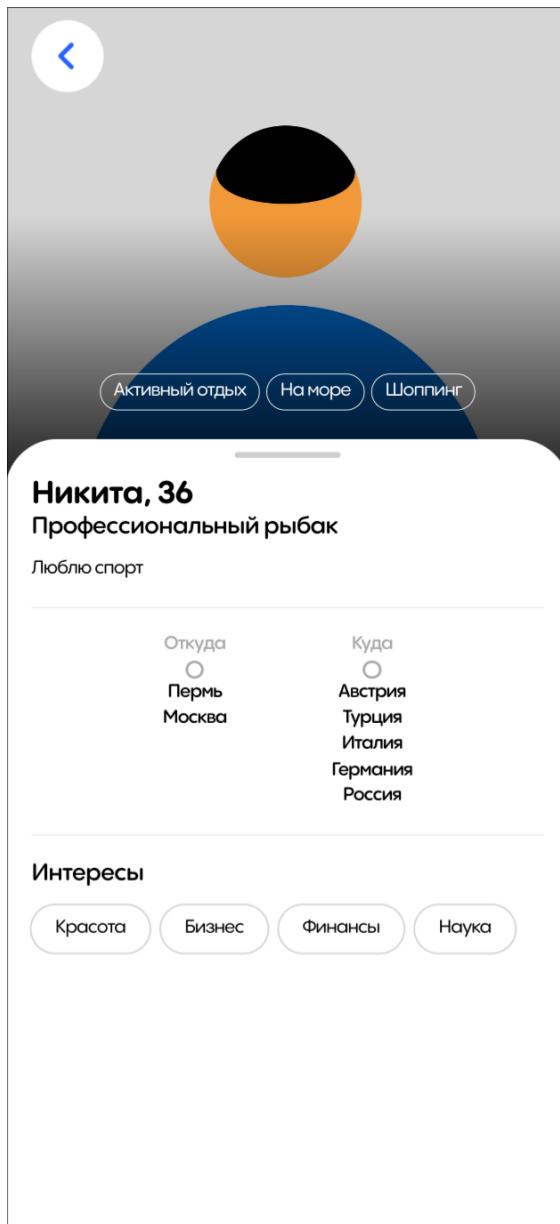


Рисунок 3.15 – Страница пользователя

При загрузке страницы мы запрашиваем данные с сервера при помощи функции `load` и `id` пользователя. Код представлен ниже.

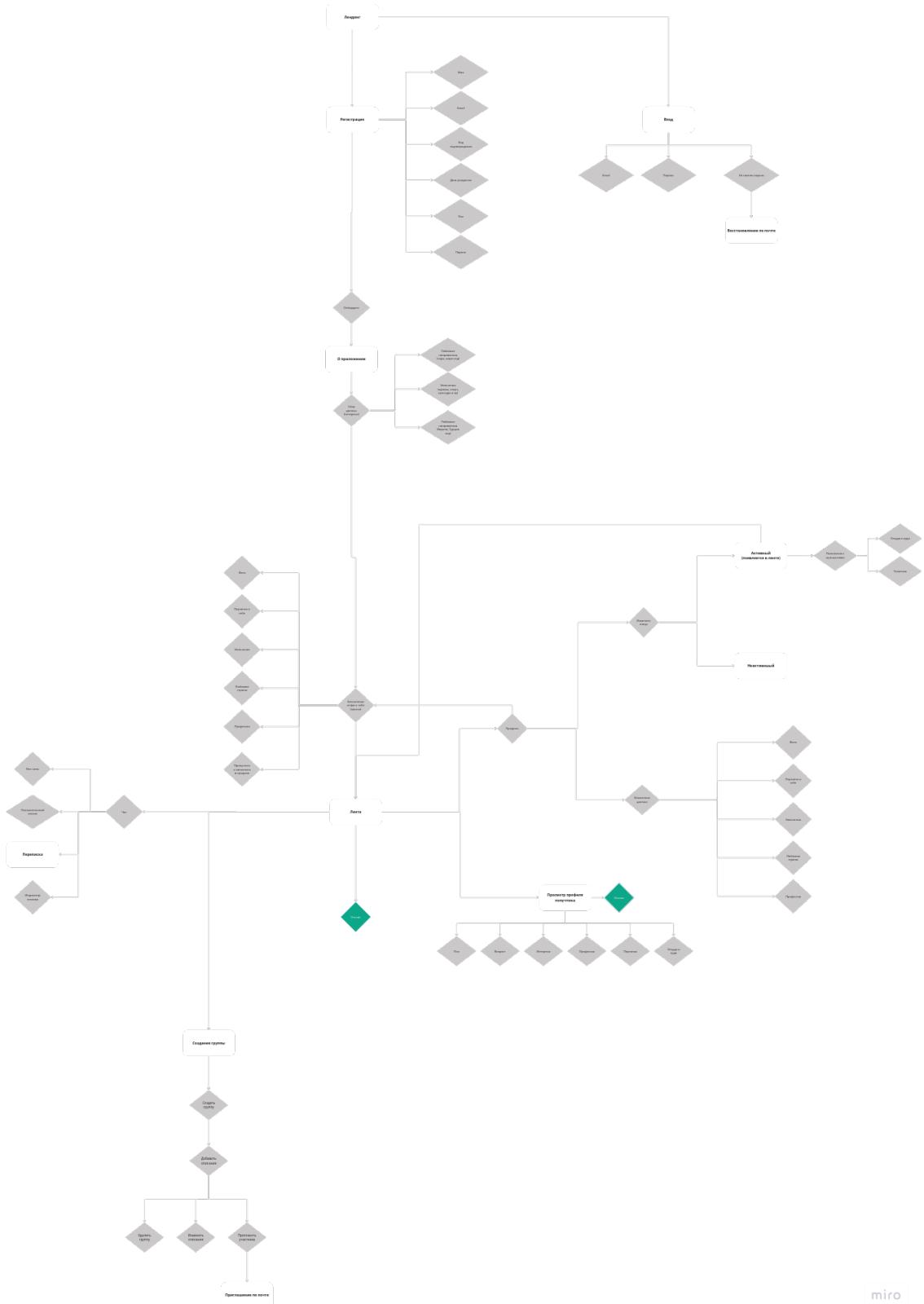
```
1 export const load: PageServerLoad = async ({ fetch, params }) => {
2   const pageUser: UserWStatus = await (
3     await fetch(`http://nginx/api/user/profile/${params.id}`)
4   ).json();
5   const body = { pageUser };
6   return body;
7 };
```

## Список литературы

1. *We Are Social*. DIGITAL 2022: ANOTHER YEAR OF BUMPER GROWTH. — URL: <https://wearesocial.com/uk/blog/2022/01/digital-2022-another-year-of-bumper-growth-2> (дата обр. 30.03.2024).
2. *Ведомости*. Внутренний туризм в России за год увеличился до рекордных 75 млн поездок. — URL: <https://www.vedomosti.ru/society/articles/2024/01/22/1016071-vnutrennii-turizm-v-rf-za-god-uvelichilsya> (дата обр. 30.03.2024).
3. *Statista Advertising and Media Outlook*. The Rapid Rise of TikTok. — URL: <https://www.statista.com/chart/28412/social-media-users-by-network-amo/> (дата обр. 01.04.2024).
4. *Smart Insights and Marketing Land*. 80% of social media browsing is on mobile devices. — URL: <https://techjury.net/blog/mobile-vs-desktop-usage> (дата обр. 30.03.2024).
5. *LePage P., Richard S.* What are Progressive Web Apps? — URL: <https://web.dev/articles/what-are-pwas> (дата обр. 30.03.2024).
6. *roguegpu*. SvelteJS: My ecosystem is bigger than yours. — URL: <https://hackmd.io/@roguegpu/r1RKQMdt3> (дата обр. 01.04.2024).
7. Superforms. — URL: <https://superforms.rocks/> (дата обр. 01.04.2024).
8. *SvelteKIT docs*. Form actions. — URL: <https://kit.svelte.dev/docs/form-actions> (дата обр. 01.04.2024).
9. *Svelte interactive docs*. Advanced loading/invalidateAll. — URL: <https://learn.svelte.dev/tutorial/invalidate-all> (дата обр. 01.04.2024).

# ПРИЛОЖЕНИЕ А

## Диаграмма User Flow



*Рисунок А.1 – Диаграмма User Flow*

miro

# ПРИЛОЖЕНИЕ Б

## Диаграмма базы данных

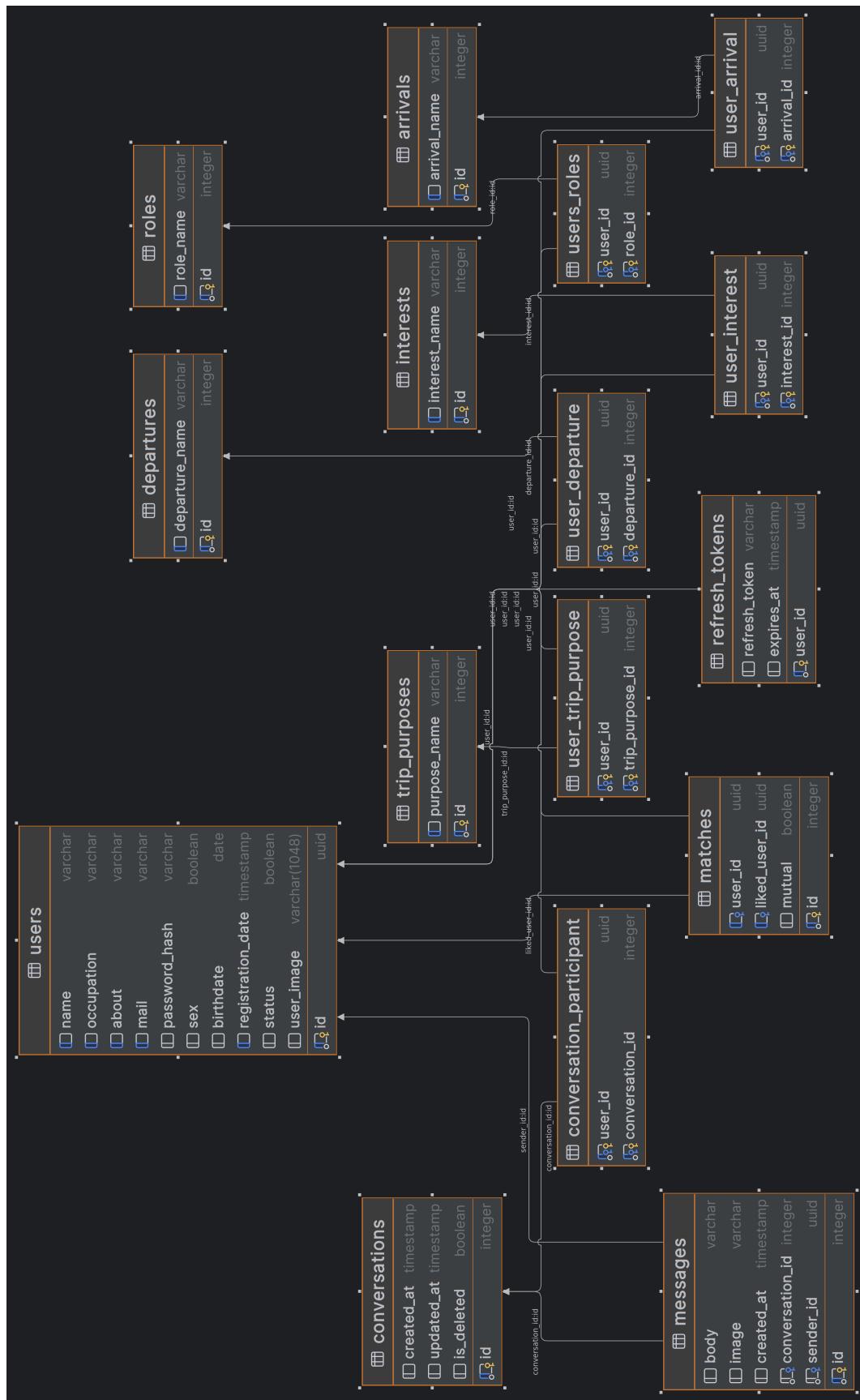


Рисунок Б.1 – Диаграмма базы данных

## ПРИЛОЖЕНИЕ В

### Создание таблиц

```
1 conversation_participant = Table(
2     "conversation_participant",
3     Base.metadata,
4     Column("user_id", ForeignKey("users.id"), primary_key=True),
5     Column("conversation_id", ForeignKey("conversations.id"), primary_key=True),
6 )
7
8 user_interest_table = Table(
9     "user_interest",
10    Base.metadata,
11    Column("user_id", UUID, ForeignKey("users.id"), primary_key=True),
12    Column("interest_id", Integer, ForeignKey("interests.id"), primary_key=True)
13 ,
14
15 user_trip_purpose_table = Table(
16     "user_trip_purpose",
17     Base.metadata,
18     Column("user_id", UUID, ForeignKey("users.id"), primary_key=True),
19     Column(
20         "trip_purpose_id", Integer, ForeignKey("trip_purposes.id"), primary_key=
True
21     ),
22 )
23
24 user_departure_table = Table(
25     "user_departure",
26     Base.metadata,
27     Column("user_id", UUID, ForeignKey("users.id"), primary_key=True),
28     Column("departure_id", Integer, ForeignKey("departures.id"), primary_key=
True),
29 )
30
31 user_arrival_table = Table(
32     "user_arrival",
33     Base.metadata,
34     Column("user_id", UUID, ForeignKey("users.id"), primary_key=True),
35     Column("arrival_id", Integer, ForeignKey("arrivals.id"), primary_key=True),
36 )
37
38
```

```

39 class Users(Base):
40     __tablename__ = "users"
41     id = Column(UUID(as_uuid=True), primary_key=True, index=True)
42     name = Column(String, index=True)
43     occupation = Column(String, index=True)
44     about = Column(String, index=True)
45     mail = Column(String, index=True)
46     password_hash = Column(String)
47     sex = Column(Boolean)
48     birthdate = Column(Date)
49     registration_date = Column(DateTime, default=datetime.datetime.utcnow, index
= True)
50     conversations: Mapped[List["Conversation"]] = relationship(
51         secondary=conversation_participant, back_populates="users"
52     )
53     interests: Mapped[List["Interests"]] = relationship(
54         "Interests", secondary=user_interest_table, back_populates="users"
55     )
56     trip_purposes: Mapped[List["TripPurposes"]] = relationship(
57         "TripPurposes", secondary=user_trip_purpose_table, back_populates="users"
58     )
59     departures: Mapped[List["Departures"]] = relationship(
60         "Departures", secondary=user_departure_table, back_populates="users"
61     )
62     arrivals: Mapped[List["Arrivals"]] = relationship(
63         "Arrivals", secondary=user_arrival_table, back_populates="users"
64     )
65     messages: Mapped[List["Message"]] = relationship(back_populates="sender")
66     status = Column(Boolean, default=True)
67     user_image: Mapped[str] = mapped_column(String(1048), nullable=True)
68
69
70 class RefreshTokens(Base):
71     __tablename__ = "refresh_tokens"
72     user_id = Column(UUID(as_uuid=True), primary_key=True, index=True)
73     refresh_token = Column(String)
74     expires_at = Column(DateTime)
75
76
77 class Matches(Base):
78     __tablename__ = "matches"
79     id = Column(Integer, autoincrement=True, primary_key=True, index=True)
80     user_id = Column(UUID(as_uuid=True), ForeignKey("users.id"), index=True)

```

```

81     liked_user_id = Column(UUID(as_uuid=True), ForeignKey("users.id"), index=True)
82     mutual = Column(Boolean, default=False)
83
84
85 class Roles(Base):
86     __tablename__ = "roles"
87     id = Column(Integer, primary_key=True, autoincrement=True)
88     role_name = Column(String, index=True)
89
90
91 class Interests(Base):
92     __tablename__ = "interests"
93     id = Column(Integer, primary_key=True, autoincrement=True)
94     interest_name = Column(String, index=True)
95     users = relationship(
96         "Users", secondary=user_interest_table, back_populates="interests"
97     )
98
99
100 class TripPurposes(Base):
101     __tablename__ = "trip_purposes"
102     id = Column(Integer, primary_key=True, autoincrement=True)
103     purpose_name = Column(String, index=True)
104     users = relationship(
105         "Users", secondary=user_trip_purpose_table, back_populates="trip_purposes"
106     )
107
108
109 class Departures(Base):
110     __tablename__ = "departures"
111     id = Column(Integer, primary_key=True, autoincrement=True)
112     departure_name = Column(String, index=True)
113     users = relationship(
114         "Users", secondary=user_departure_table, back_populates="departures"
115     )
116
117
118 class Arrivals(Base):
119     __tablename__ = "arrivals"
120     id = Column(Integer, primary_key=True, autoincrement=True)
121     arrival_name = Column(String, index=True)
122     users = relationship(
123         "Users", secondary=user_arrival_table, back_populates="arrivals"

```

```

124     )
125
126
127 class UsersRoles(Base):
128     __tablename__ = "users_roles"
129     user_id = Column(UUID(as_uuid=True), ForeignKey("users.id"), primary_key=True)
130     role_id = Column(Integer, ForeignKey("roles.id"), primary_key=True)
131
132
133 class Conversation(Base):
134     __tablename__ = "conversations"
135
136     id = Column(Integer, primary_key=True, autoincrement=True)
137     created_at = Column(DateTime, default=datetime.datetime.utcnow())
138     updated_at = Column(DateTime, default=datetime.datetime.utcnow())
139     is_deleted = Column(Boolean, default=False)
140
141     users: Mapped[List["Users"]] = relationship(
142         secondary=conversation_participant, back_populates="conversations"
143     )
144     messages: Mapped[List["Message"]] = relationship(back_populates="conversation")
145
146
147 class Message(Base):
148     __tablename__ = "messages"
149
150     id = Column(Integer, primary_key=True, autoincrement=True)
151     body = Column(String)
152     image = Column(String)
153     created_at = Column(DateTime, default=datetime.datetime.utcnow())
154
155     conversation_id: Mapped[int] = mapped_column(ForeignKey("conversations.id"))
156     conversation: Mapped["Conversation"] = relationship(back_populates="messages")
157
158     sender_id: Mapped[UUID] = mapped_column(ForeignKey("users.id"))
159     sender: Mapped["Users"] = relationship(back_populates="messages")

```