Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра информатики

Дисциплина: Модели данных и системы управления базами данных

*К защите допустить*:

И. О. Заведующего кафедрой информатики

\_\_\_\_\_\_\_\_\_\_\_\_С. И. Сиротко

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовому проекту

на тему

**ВЕБ-ПРИЛОЖЕНИЕ «DATING AGENCY»**

БГУИР КП 1-40 04 01 006 ПЗ

Студент А. А. Быбко

Руководитель А. В. Давыдчик

Минск 2024

СОДЕРЖАНИЕ

[Введение 5](#_Toc184587053)

[1 Архитектура программного обеспечения 6](#_Toc184587054)

[1.1 Структура и архитектура вычислительной системы 6](#_Toc184587055)

[1.2 История, версии и достоинства 7](#_Toc184587056)

[1.3 Обоснование выбора вычислительной системы 9](#_Toc184587057)

[1.4 Анализ выбранной вычислительной системы для написания программы 10](#_Toc184587058)

[2 Платформа программного обеспечения 11](#_Toc184587059)

[2.1 Структура и архитектура платформы 11](#_Toc184587060)

[2.2 История, версии и достоинства 12](#_Toc184587061)

[2.3 Обоснование выбора платформы 14](#_Toc184587062)

[2.4 Анализ операционной системы для написания программы 15](#_Toc184587063)

[3 Теоретическое обоснование разработки программного продукта 16](#_Toc184587064)

[3.1 Обоснование необходимости разработки 16](#_Toc184587065)

[3.2 Технологии программирования, используемые для решения поставленных задач 17](#_Toc184587066)

[3.3 Связь архитектуры вычислительной системы с разрабатываемым программным обеспечением 17](#_Toc184587067)

[4 Проектирование функциональных возможностей программы 19](#_Toc184587068)

[4.1 Вход в систему 19](#_Toc184587069)

[4.2 Панель пользователя 20](#_Toc184587070)

[4.3 Панель администратора 22](#_Toc184587071)

[5 Проектирование разрабатываемой базы данных программного обеспечения 25](#_Toc184587072)

[5.1 Общие принципы проектирования 25](#_Toc184587073)

[5.2 Описание структуры базы данных 25](#_Toc184587074)

[Заключение 27](#_Toc184587075)

[Список литературных источников 28](#_Toc184587076)

[Приложение А](#_Toc184587077) [(обязательное)](#_Toc184587078) [Листинг программного кода 30](#_Toc184587079)

[Приложение Б](#_Toc184587080) [(обязательное)](#_Toc184587081) [Конечная схема базы данных 43](#_Toc184587082)

# **ВВЕДЕНИЕ**

Использование веб-приложений для организации знакомств в интернете стало одним из ключевых направлений в развитии современных социальных платформ. Такие приложения позволяют людям находить партнеров, исходя из их интересов, предпочтений и других индивидуальных характеристик. Применение баз данных, таких как PostgreSQL, способствует эффективному хранению, обработке и анализу большого объема пользовательской информации, обеспечивая стабильность и безопасность системы.

Цель данного курсового проекта заключается в разработке веб-приложения для службы знакомств с использованием базы данных PostgreSQL. Приложение должно поддерживать функции регистрации пользователей, создания и управления профилями, поиска партнеров по заданным параметрам и взаимодействия между пользователями.

Для достижения поставленной цели определены следующие задачи:

1 Спроектировать структуру базы данных PostgreSQL для хранения данных.

2 Разработать серверную часть приложения, обеспечивающую связь с базой данных и реализацию бизнес-логики.

3 Создать интерфейс веб-приложения для удобного управления профилями и поиска партнеров.

# **1 АРХИТЕКТУРА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ**

**1.1 Структура и архитектура вычислительной системы**

PostgreSQL – это реляционная база данных с открытым кодом, которая поддерживается в течение 30 лет разработки и является одной из наиболее известных среди всех существующих реляционных баз данных. Популярностью у разработчиков и администраторов база данных PostgreSQL обязана своей исключительной гибкости и целостности. [1]

PostgreSQL реализован в архитектуре клиент-сервер. Рабочий сеанс PostgreSQL включает следующие взаимодействующие процессы (программы):

1 Главный серверный процесс, управляющий файлами баз данных, принимающий подключения клиентских приложений и выполняющий различные запросы клиентов к базам данных. Эта программа сервера БД называется postgres.

2 Клиентское приложение пользователя, желающее выполнять операции в базе данных. Клиентские приложения могут быть очень разнообразными: это может быть текстовая утилита, графическое приложение, веб-сервер, использующий базу данных для отображения веб-страниц, или специализированный инструмент для обслуживания БД. Некоторые клиентские приложения поставляются в составе дистрибутива PostgreSQL, однако большинство создают сторонние разработчики.

Как и в других типичных клиент-серверных приложениях, клиент и сервер могут располагаться на разных компьютерах. В этом случае они взаимодействует по сети TCP/IP. Важно не забывать это и понимать, что файлы, доступные на клиентском компьютере, могут быть недоступны (или доступны только под другим именем) на компьютере-сервере.

Сервер PostgreSQL может обслуживать одновременно несколько подключений клиентов. Для этого он запускает («порождает») отдельный процесс для каждого подключения. Можно сказать, что клиент и серверный процесс общаются, не затрагивая главный процесс postgres. Таким образом, главный серверный процесс всегда работает и ожидает подключения клиентов, принимая которые, он организует взаимодействие клиента и отдельного серверного процесса. [2]

## **1.2 История, версии и достоинства**

В 1985 году, когда SQL ещё не был мировым стандартом, группа инженеров Калифорнийского университета Беркли под руководством профессора Майкла Стоунбрейкера начала разработку реляционной СУБД POSTGRES. В основе проекта лежали наработки, которые Стоунбрейкер сделал во время работы над INGRES – одной из первых реляционных СУБД.

К 1988 году команда опубликовала ряд научных статей, описывающих язык запросов POSTQUEL, который лежал в основе POSTGRES. Название недвусмысленно намекало на то, что новый язык более современный и продвинутый, чем SQL. Саму POSTGRES называли «постреляционной СУБД» – её создавали для того, чтобы преодолеть ставшие тогда очевидными ограничения SQL.

Первая версия инновационной СУБД вышла в 1989 году, но уже в 1992-м, после нескольких обновлений, проект закрыли. К счастью, исходный код POSTGRES распространялся по лицензии BSD. Выпускники Беркли Эндрю Ю и Джоли Чену продолжили её развивать, заменив язык POSTQUEL на SQL, который на тот момент уже стал международным стандартом. Новая инкарнация сперва получила имя Postgres95 (по году создания), а затем PostgreSQL. С тех пор вышло 28 версий СУБД.

Сегодня проект поддерживает довольно небольшая по нынешним меркам команда разработчиков. Во главе команды стоит управляющий комитет (Core Team) – члены комитета принимают решения по развитию и выпуску новых версий Postgres. Разработчики делятся на обычных (contributors) и основных (major contributors). Кроме того, небольшая группа разработчиков (commiters) имеет право вносить изменения в исходный код. [3]

Таким образом, История PostgreSQL демонстрирует её эволюцию от исследовательского проекта до одной из самых мощных и популярных систем управления базами данных. Благодаря постоянному развитию и внедрению передовых технологий, PostgreSQL обладает рядом значительных преимуществ, которые делают её идеальной для современных веб-приложений:

1 Расширяемость и богатый набор типов данных. Помимо стандартных, в PostgreSQL есть типы для геометрических расчётов, сетевых адресов и полнотекстового поиска. Мощная система расширений позволяет добавлять новые возможности и типы данных. Кроме того, администратор может писать свои функции и процедуры на Python, PHP, Java, Ruby и многих других языках программирования, а также загружать модули на языке C из центрального репозитория PGXN.

2 Масштабируемость. Для повышения производительности и масштабируемости в PostgreSQL используются разные виды блокировок на уровне таблиц и строк; шесть видов индексов, среди которых B-дерево и обобщённое дерево поиска (GiST) для полнотекстового поиска; наследование таблиц – для быстрого создания таблиц на основе имеющейся структуры. Также можете анализировать скорость выполнения запросов с помощью команды explain, очищать диск от мусора командой vacuum и собирать статистику по таблицам с analyze.

3 Кроссплатформенность. PostgreSQL поддерживается всеми популярными операционными системами, среди которых различные дистрибутивы Linux и BSD, macOS, Windows, Solaris и другие. Интерфейсы для этой СУБД реализованы практически во всех языках программирования.

4 Безопасность. В PostgreSQL есть множество инструментов для защиты данных от злоумышленников: пароль, Kerberos, LDAP, GSSAPI, SSPI, PAM и другие. Она позволяет управлять доступом к объектам БД на нескольких уровнях – от базы данных до отдельных столбцов, а также шифровать данные на аппаратном уровне.

5 Возможности NoSQL. Помимо стандартных форматов, PostgreSQL поддерживает XML, а с девятой версии – JSON и JSONB. Последний позволяет не разбирать JSON-документ перед записью в базу данных, что существенно ускоряет его сохранение в БД.

6 Свободное распространение и открытый код. Проект распространяется под лицензией BSD, что позволяет бесплатно его использовать, модифицировать и распространять. Исходный код можно посмотреть на зеркале официального Git-репозитория.

7 Живое сообщество и исчерпывающее официальное руководство. PostgreSQL поддерживается большим активным сообществом пользователей и разработчиков, которые получают новости о проекте через еженедельные рассылки, обсуждают вопросы администрирования и ведут дискуссии. Официальная документация регулярно обновляется и на момент публикации содержит около 3000 страниц подробных и понятных инструкций. [3]

## **1.3 Обоснование выбора вычислительной системы**

PostgreSQL редко выбирают для сайтов или приложений с небольшим числом пользователей – на ней разворачивают высоконагруженные сложные корпоративные системы, платформы сбора и анализа данных, финансовые сервисы и другие критичные системы.

PostgreSQL решает бизнес-задачи в тех сферах, где простои в работе приводят к финансовым и репутационным потерям: в ритейле, e-commerce, в энергетике и промышленности, в финансах и страховании, в образовательных учреждениях и IT-компаниях. С помощью PostgreSQL можно:

– строить сложные платформы данных для анализа показателей рынка, учёта ресурсов и финансовой оценки;

– хранить данные с датчиков и устройств;

– быстро запускать на рынок продукты и приложения;

– обрабатывать и анализировать большие массивы данных;

– создавать и запускать корпоративные системы управления.

Для решения задач у PostgreSQL есть полный набор возможностей и функций:

– надёжность – ACID, WAL (Write-Ahead Logging), PITR (Point-in-Time Recovery), репликация и резервное копирование;

– производительность – система индексов, гибкая блокировка, интеллектуальный планировщик запросов, кеширование и буфер;

– расширяемость – создание новых функций, типов, агрегатов и индексов, перенос логики приложения на БД;

– расширенная поддержка SQL и многочисленных типов данных.

Но все эти возможности хорошо работают при квалифицированной настройке и обслуживании СУБД. Администратор должен правильно прописать параметры БД в конфигурационном файле, организовать круглосуточный мониторинг, контролировать работу дополнительных сервисов, следить за нагрузкой фоновых процессов, подбирать и модернизировать аппаратную платформу и виртуализацию, корректно спланировать схему данных, настроить репликацию и резервное копирование. [4]

## **1.4 Анализ выбранной вычислительной системы для написания программы**

Веб-приложение для знакомств, как правило, предполагает работу с тысячами, а иногда и миллионами пользователей. В таких условиях критически важно использовать СУБД, которая способна гарантировать надежность и стабильность работы при большом объеме данных. PostgreSQL идеально подходит для хранения таких данных, как: профили пользователей, средства взаимодействия (сообщения, встречи, жалобы), мониторинг.

Эти данные должны храниться эффективно, с возможностью быстрого доступа и обработки, а также с гарантией целостности и надежности, чего PostgreSQL добивается с помощью ACID-транзакций (атомарность, согласованность, изолированность, долговечность). Например, если два пользователя одновременно отправляют запрос на встречу, PostgreSQL обеспечит корректную обработку этих данных с учетом транзакций.

Также веб-приложение для знакомств требует обработки сложных запросов для подбора партнёров. Пользователи будут искать людей по различным критериям: возрасту, интересам, местоположению и т.д. PostgreSQL поддерживает сложные SQL-запросы, включая подзапросы и операторы для работы с географическими данными (например, для поиска пользователей в радиусе определенной локации). Пример: для поиска партнеров на основе расстояния (с использованием данных о местоположении) и интересов можно использовать расширения, такие как PostGIS, которые интегрируют географическую информацию в базу данных.

# **2 ПЛАТФОРМА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ**

## **2.1 Структура и архитектура платформы**

Windows – группа семейств коммерческих проприетарных операционных систем корпорации Microsoft, ориентированных на управление с помощью графического интерфейса. [5]

Архитектура Windows – это совокупность программных и аппаратных компонентов, которые определяют, как операционная система управляет ресурсами компьютера, обеспечивает взаимодействие между приложениями и устройствами, решает проблемы и защищает данные от постороннего доступа (рисунок 2.1). Архитектура Windows может быть разделена на два основных уровня: ядро (kernel) и пользовательский режим (user mode).

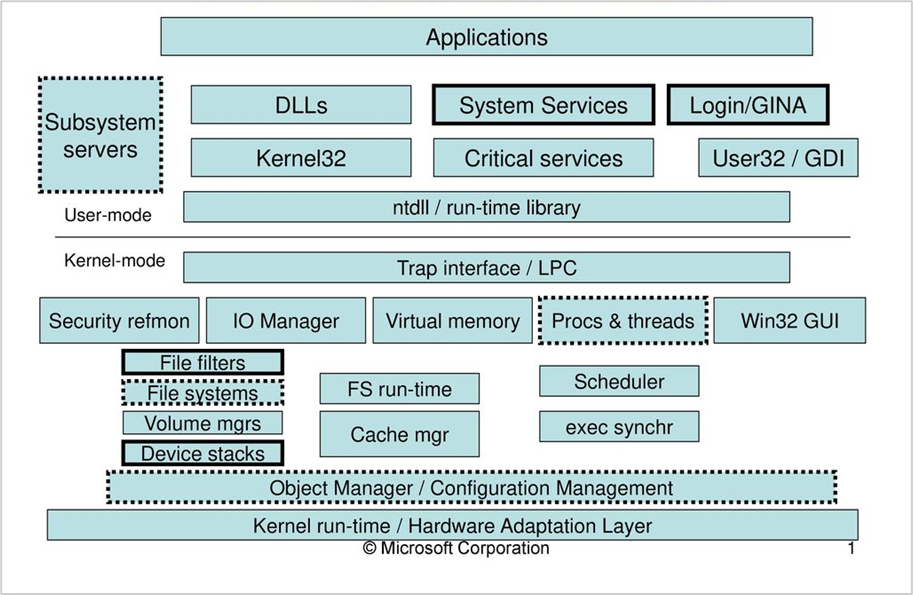


Рисунок 2.1 – Архитектура Windows

Ядро – это низкоуровневая часть операционной системы, которая работает в специальном режиме процессора и имеет непосредственный доступ к аппаратным ресурсам.

Пользовательский режим – это высокоуровневая часть операционной системы, которая работает в обычном режиме процессора и содержит разнообразные приложения и службы.

Ядро и пользовательский режим общаются друг с другом через специальные механизмы, такие как системные вызовы (system calls), сообщения (messages) и объекты ядра (kernel objects).

Windows поддерживает различные архитектуры процессоров, такие как x86, x64, ARM и ARM64. Каждая архитектура имеет свои особенности, такие как размер адресного пространства, набор инструкций, регистры и т.д. Windows адаптируется к разным архитектурам с помощью специального слоя абстракции аппаратного обеспечения (hardware abstraction layer, HAL), который предоставляет единый интерфейс для доступа к аппаратным ресурсам.

Windows отличается от других операционных систем своей гибкостью и совместимостью. Windows может запускать приложения, написанные для разных API (application programming interface), таких как Win32, POSIX, WinRT и т.д., благодаря подсистемам пользовательского режима, которые реализуют эти API. Windows также может работать с разными типами устройств, таких как клавиатуры, мыши, диски, принтеры и т.д., благодаря драйверам устройств, которые обеспечивают связь между ядром и устройствами. [6]

## **2.2 История, версии и достоинства**

Microsoft Windows была анонсирована Биллом Гейтсом 10 ноября 1983 года, за 2 года до ее первого выпуска. Microsoft представила Windows как графический пользовательский интерфейс для MS-DOS, который был представлен двумя годами ранее. Линейка продуктов эволюционировала в 1990-х годах из операционной среды в полностью законченную, современную операционную систему на двух этапах разработки, каждый со своей отдельной кодовой базой.

Первые версии Windows (от 1.0 до 3.11) были графическими оболочками, которые запускались из MS-DOS. Windows 95, хотя по-прежнему базировалась на MS-DOS, была собственной операционной системой. В Windows 95 также было перенесено значительное количество 16-разрядного кода из Windows 3.1. В Windows 95 появилось множество функций, которые с тех пор являются частью продукта, включая меню «Пуск», панель задач и проводник Windows. В 1997 году Microsoft выпустила Internet Explorer 4, который включал обновление для рабочего стола Windows. Он был направлен на интеграцию Internet Explorer и веб в пользовательский интерфейс, а также привнес в Windows множество новых функций, таких как возможность отображения изображений в формате JPEG в качестве обоев рабочего стола и навигация по единому окну в проводнике Windows. В 1998 году Microsoft выпустила Windows 98, которая также включала обновление для рабочего стола Windows и Internet Explorer 4 по умолчанию. Включение Internet Explorer 4 и обновления для рабочего стола привело к антимонопольному делу в Соединенных Штатах. Windows 98 включала поддержку USB из коробки, а также подключи и играй, который позволяет устройствам работать при подключении, не требуя перезагрузки системы или ручной настройки. Windows Me, последняя версия Windows на базе DOS, была ориентирована на потребителей и выпущена в 2000 году. В нем представлены «Восстановление системы», «Центр справки и поддержки», обновленные версии «Дефрагментатор диска» и другие системные инструменты.

В 1993 году Microsoft выпустила Windows NT 3.1, первую версию недавно разработанной операционной системы Windows NT, за которой последовали Windows NT 3.5 в 1994 году и Windows NT 3.51 в 1995 году. «NT» является инициалами для обозначения «Новой технологии». В отличие от серии операционных систем Windows 9x, это была полностью 32-разрядная операционная система. В NT 3.1 была представлена NTFS, файловая система, разработанная для замены старой таблицы распределения файлов (FAT), которая использовалась DOS и операционными системами Windows на базе DOS. В 1996 году была выпущена Windows NT 4.0, которая включала полностью 32-разрядную версию Windows Explorer, написанную специально для нее, благодаря чему операционная система работала как Windows 95. Windows NT изначально разрабатывалась для использования в высокопроизводительных системах и серверах, но с выпуском Windows 2000 были включены многие функции Windows 95 и Windows 98, ориентированные на потребителя, такие как обновление для рабочего стола Windows, Internet Explorer 5, поддержка USB и проигрыватель Windows Media. Эти функции, ориентированные на потребителя, были дополнительно расширены в Windows XP в 2001 году, что включало новый визуальный стиль под названием «Luna», более удобный интерфейс, обновленные версии проигрывателя Windows Media и Internet Explorer 6 по умолчанию и расширенные функции Windows Me, такие как «Центр справки и поддержки и восстановление системы». Windows Vista, выпущенная в 2007 году, была направлена на защиту операционной системы Windows от компьютерных вирусов и другого вредоносного программного обеспечения путем внедрения таких функций, как контроль учетных записей пользователей. Новые функции включают Windows Aero, обновленные версии стандартных игр, Windows Movie Maker и Windows Mail для замены Outlook Express. Несмотря на это, Windows Vista подверглась критике из-за низкой производительности на старом оборудовании и высоких на тот момент системных требований.

Windows 7 появилась в 2009 году, почти через три года после ее запуска, и несмотря на то, что технически у нее были более высокие системные требования, рецензенты отметили, что она работала лучше, чем Windows Vista. Windows 7 удалила многие приложения, такие как Windows Movie Maker, Windows Photo Gallery и Windows Mail, вместо этого потребовав от пользователей загружать отдельные Windows Live Essentials для получения некоторых из этих функций и других онлайн-сервисов. Windows 8, выпущенная в 2012 году, внесла множество противоречивых изменений, таких как замена меню «Пуск» экраном «Пуск», удаление интерфейса Aero в пользу плоского цветного интерфейса, а также введение приложений «Metro» (позже переименованных в приложения для универсальной платформы Windows) и элемента пользовательского интерфейса Charms Bar, все из которых получили значительную критику со стороны рецензентов. Windows 8.1, бесплатное обновление до Windows 8, было выпущено в 2013 году.

Следующая версия Windows, Windows 10, выпущенная в 2015 году, вновь представила меню «Пуск» и добавила возможность запускать универсальные приложения платформы Windows в окне, а не всегда в полноэкранном режиме. Windows 10 в целом была хорошо принята, и многие рецензенты заявили, что Windows 10 – это то, какой должна была быть Windows 8.

Последняя версия Windows, Windows 11, была выпущена 5 октября 2021 года. Windows 11 включает в себя переработанный пользовательский интерфейс, включая новое меню «Пуск», визуальный стиль со скругленными углами и новый макет для Microsoft Store, а также включенный Microsoft Edge по умолчанию. [7]

## **2.3 Обоснование выбора платформы**

Преимущества операционной системы Windows являются:

1 Поддержка всего оборудования. Поскольку ОС Windows используется 72% пользователей, большинство производителей оборудования производят драйверы для Windows.

2 Простота использования. Во всех версиях Microsoft Windows есть что-то общее, что позволяет пользователям легко переходить с одной версии на другую. У пользователей Windows 7 не возникает трудностей с переходом на Windows 10, поскольку большинство функций Windows 10 такие же, как у Windows 7. Пользовательский интерфейс Windows также прост в использовании, чем у UNIX и MAC.

3 Поддержка программного обеспечения. Платформа Windows лучше всего подходит для разработчиков игр и программного обеспечения. Аудитория Windows многочисленна, поэтому разработчики предпочитают создавать утилиты, игры и программное обеспечение для ОС Windows.

4 Функция «Plug and Play». Большинство аппаратных средств может быть обнаружено автоматически с помощью функции «Plug and Play». Благодаря данной функции, не нужно вручную устанавливать оборудование, но оно готово к использованию после подключения, например, веб-камеры, клавиатуры, мыши, мобильного устройства и т.д.

5 Рабочий стол и сенсорный экран. Windows 10 предназначена как для устройств с сенсорным экраном, так и для настольных компьютеров. Пользовательский интерфейс Windows 10 выполнен таким образом, что он лучше работает на любом типе устройств с Windows. [8]

## **2.4 Анализ операционной системы для написания программы**

Операционная система Windows является хорошим выбором для разработки веб-приложений, таких как платформа для знакомств, благодаря своей стабильности и поддержке различных технологий. В условиях высокой нагрузки и большого числа пользователей, Windows гарантирует стабильную работу за счет эффективной поддержки многозадачности и высокопроизводительных серверных решений, таких как Windows Server.

Для разработки платформы знакомств Windows предоставляет доступ к инструментам разработки, таким как Visual Studio и JetBrains Rider, а также поддерживает множество языков программирования, включая C#, Python и JavaScript, что важно для создания различных компонентов веб-приложения. Также операционная система предлагает хорошие средства виртуализации через Hyper-V, что полезно для масштабирования и управления серверной инфраструктурой.

Встроенные средства защиты, такие как BitLocker и Windows Defender, обеспечивают надежную защиту личной информации на уровне операционной системы. Windows также легко интегрируется с облачными решениями, такими как Microsoft Azure, что позволяет эффективно масштабировать приложение и использовать облачные ресурсы для хранения и обработки данных.

# **3 ТЕОРЕТИЧЕСКОЕ ОБОСНОВАНИЕ РАЗРАБОТКИ ПРОГРАММНОГО ПРОДУКТА**

**3.1 Обоснование необходимости разработки**

Современные социальные платформы, такие как приложения для знакомств, становятся неотъемлемой частью цифрового взаимодействия между людьми. Разработка веб-приложения для знакомств имеет высокую актуальность в связи с растущей потребностью в эффективных и безопасных средствах для поиска партнеров. В последние годы количество пользователей таких сервисов значительно возросло, что создает необходимость в создании более удобных и функциональных платформ. Согласно исследованию Statista, в 2023 году на платформы для знакомств приходилось около 11% всех интернет-пользователей, и прогнозируется, что эта цифра будет продолжать расти. [9]

Одной из ключевых проблем таких платформ является управление большими объемами данных, включая профили пользователей, их предпочтения, сообщения и историю взаимодействий. Важно обеспечить безопасность и конфиденциальность этих данных, что требует использования надежных технологий и архитектуры. Веб-приложения для знакомств часто сталкиваются с проблемой масштабируемости, особенно когда количество пользователей достигает миллионов. Для успешного функционирования платформы необходимы технологии, которые обеспечат высокую производительность, возможность быстрого поиска и обработки данных.

Также важно учитывать, что платформы для знакомств должны поддерживать сложные алгоритмы подбора партнеров, используя различные критерии, такие как возраст, интересы и местоположение пользователей. Это требует обработки и хранения данных в структурированном виде, а также обеспечения быстрого доступа к этим данным.

Для решения этих задач необходимо использовать надежные и масштабируемые технологии. Программная платформа, использующая такие решения как Python, Flet, и PostgreSQL, поможет эффективно решать задачи обработки, хранения и анализа данных. Эти технологии обеспечивают необходимую гибкость, безопасность и производительность, что делает разработку веб-приложения для знакомств важной и актуальной задачей.

**3.2 Технологии программирования, используемые для решения поставленных задач**

Для реализации программного обеспечения были выбраны следующие технологии:

1 Python. Это мультипарадигмальный высокоуровневый язык программирования общего назначения с динамической строгой типизацией и автоматическим управлением памятью, ориентированный на повышение производительности разработчика, читаемости кода и его качества, а также на обеспечение переносимости написанных на нём программ. Язык является полностью объектно-ориентированным в том плане, что всё является объектами. Необычной особенностью языка является выделение блоков кода отступами. Синтаксис ядра языка минималистичен, за счёт чего на практике редко возникает необходимость обращаться к документации. Сам же язык известен как интерпретируемый и используется в том числе для написания скриптов. Недостатками языка являются зачастую более низкая скорость работы и более высокое потребление памяти написанных на нём программ по сравнению с аналогичным кодом, написанным на компилируемых языках, таких как C или C++. [10]

2 Библиотека psycopg2. Это один из самых популярных и широко используемых драйверов Python для PostgreSQL. Реализация на языке C обеспечивает высокую производительность, а поддержка параметризованных запросов помогает предотвратить SQL-инъекции. [11]

3 Библиотека flet. Э это библиотека Python, с помощью которой разработчики могут создавать веб-приложения в реальном времени, мобильные приложения и настольные приложения без непосредственного использования Flutter. [12]

**3.3 Связь архитектуры вычислительной системы с разрабатываемым программным обеспечением**

Архитектура вычислительной системы должна быть оптимально интегрирована с архитектурой программного обеспечения, чтобы обеспечить высокую производительность, масштабируемость и безопасность.

Веб-приложение предполагает постоянное взаимодействие с пользователями, что требует высокой пропускной способности серверов и низкой задержки при обработке запросов. Использование PostgreSQL в качестве СУБД позволяет обеспечить надежную и эффективную работу с данными. PostgreSQL идеально подходит для хранения различных типов данных, таких как профили пользователей, сообщения и предпочтения, а также для выполнения сложных запросов на основе различных критериев, таких как возраст, местоположение, интересы и т. д. Серверная инфраструктура должна обеспечивать бесперебойную работу при увеличении нагрузки, а PostgreSQL предоставляет функционал для масштабирования, включая репликацию и шардирование.

Веб-приложение будет работать с большим количеством пользователей, что требует особого внимания к масштабируемости базы данных. PostgreSQL предлагает встроенные механизмы, такие как индексы и оптимизация запросов, которые позволяют поддерживать высокую скорость работы с большими объемами данных. Благодаря возможностям расширения, таким как использование расширений, например, PostGIS для работы с географическими данными (например, для поиска пользователей в определённом радиусе), PostgreSQL обеспечит необходимую производительность при поиске партнеров и взаимодействии между пользователями.

Веб-приложение будет работать с чувствительными данными пользователей, такими как личная информация, фотографии и переписка. Для обеспечения безопасности данных PostgreSQL использует механизмы шифрования, а также встроенную поддержку транзакций с гарантией атомарности, согласованности, изолированности и долговечности (ACID), что снижает риск потери данных при сбоях системы. Кроме того, PostgreSQL поддерживает управление доступом с помощью ролей и привилегий, что обеспечивает дополнительный уровень безопасности при работе с данными пользователей.

Таким образом, интеграция PostgreSQL с архитектурой вычислительной системы позволяет эффективно решать задачи, связанные с обработкой и хранением данных в веб-приложении для знакомств. Это обеспечивает надежность, безопасность и высокую производительность при работе с большими объемами информации, что критически важно для успешного функционирования приложения на платформе с большим количеством пользователей.

# **4 ПРОЕКТИРОВАНИЕ ФУНКЦИОНАЛЬНЫХ ВОЗМОЖНОСТЕЙ ПРОГРАММЫ**

## **4.1 Вход в систему**

На главном экране отображается форма (рисунок 4.1), позволяющая выбрать действие: войти в существующий аккаунт или перейти к регистрации. При входе пользователь вводит имя пользователя и пароль. Если данные введены корректно, система перенаправляет пользователя на его личную страницу. В случае, если пользователь обладает правами администратора, происходит автоматический переход на специальную страницу управления.

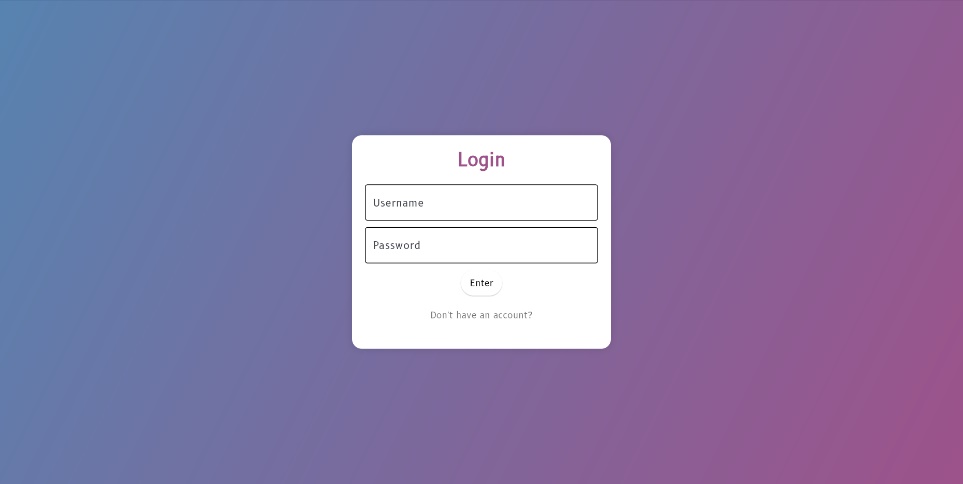


Рисунок 4.1 – Страница входа

Если пользователь не имеет аккаунта, он может перейти на форму регистрации (рисунок 4.2). Здесь необходимо заполнить расширенную анкету, включающую такие данные, как имя пользователя, пароль с подтверждением, личные данные (имя, фамилия, возраст), а также дополнительную информацию: ссылку на фотографию, список хобби, профессию, описание, страну и город. После успешного заполнения всех полей и отправки данных система уведомляет о завершении регистрации и возвращает пользователя на страницу входа.

В случае ошибок, таких как некорректные данные или проблемы с сервером, программа отображает соответствующие уведомления, что позволяет пользователю понять причину и повторить действие.

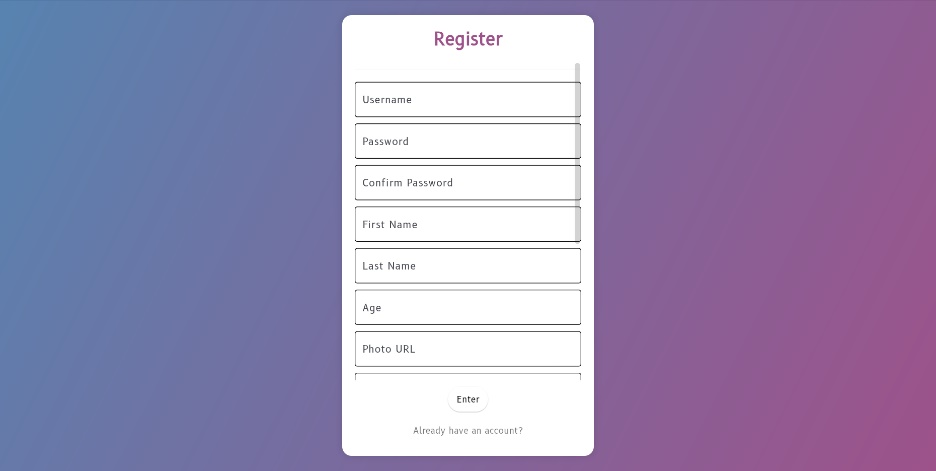


Рисунок 4.2 – Страница регистрации

## **4.2 Панель пользователя**

Панель пользователя предоставляет удобный интерфейс для выполнения различных задач и взаимодействия с системой. Пользователь может работать с несколькими основными разделами, включая управление профилями, чаты, встречи, а также персональные настройки.

После входа в систему пользователь видит список доступных профилей других пользователей, которые можно просматривать (рисунок 4.3). Для каждого профиля отображаются имя, возраст, местоположение, хобби и другая информация. В профиле можно создать чат или назначить встречу с пользователем, указав детали и время (рисунок 4.4). Для удобства поиска предусмотрен фильтр, который помогает быстро находить нужные профили по имени, возрасту или местоположению.

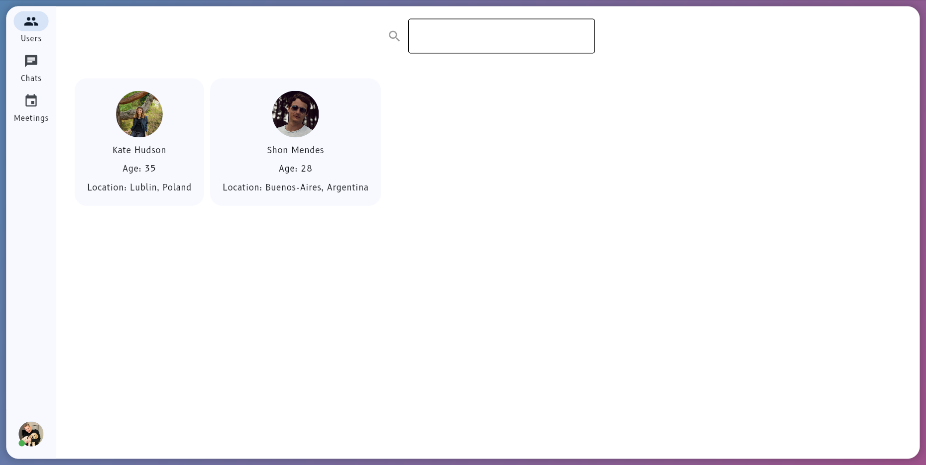


Рисунок 4.3 – Список профилей

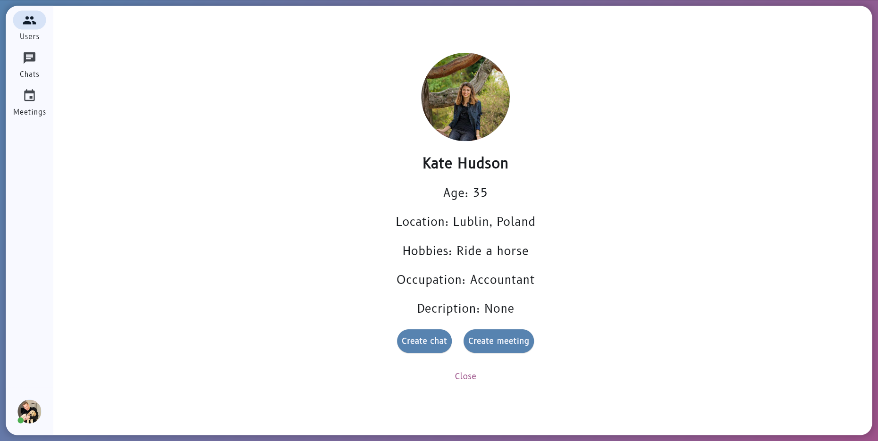


Рисунок 4.4 – Детали профиля

В разделе чатов пользователь может просматривать список своих чатов с другими участниками (рисунок 4.5). Для каждого чата отображается имя, последний отправленный текст и время сообщения. Пользователь может открыть любой чат, чтобы видеть историю переписки и отправлять новые сообщения (рисунок 4.6).

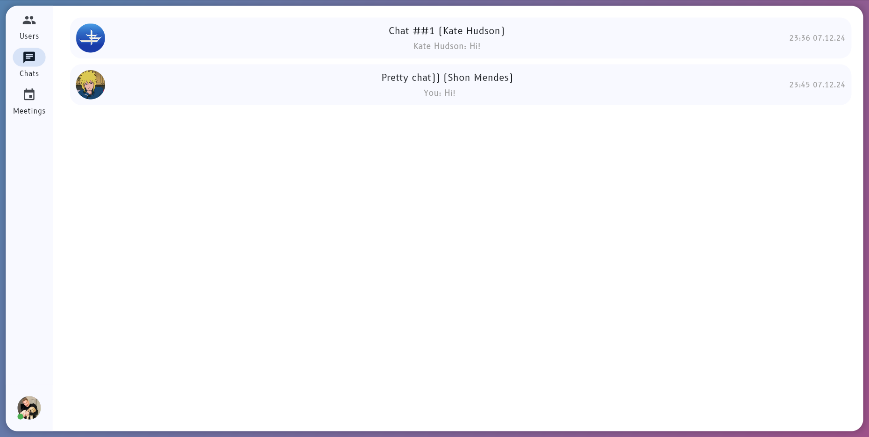


Рисунок 4.5 – Список чатов

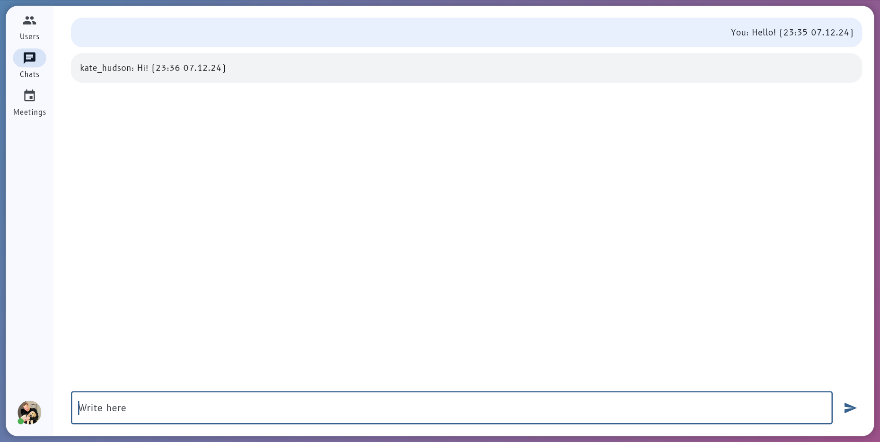


Рисунок 4.6 – Чат с сообщениями

В разделе встреч отображается список запланированных встреч, включая дату, время и местоположение (рисунок 4.7). Пользователь может удалять встречи, если они больше не актуальны.

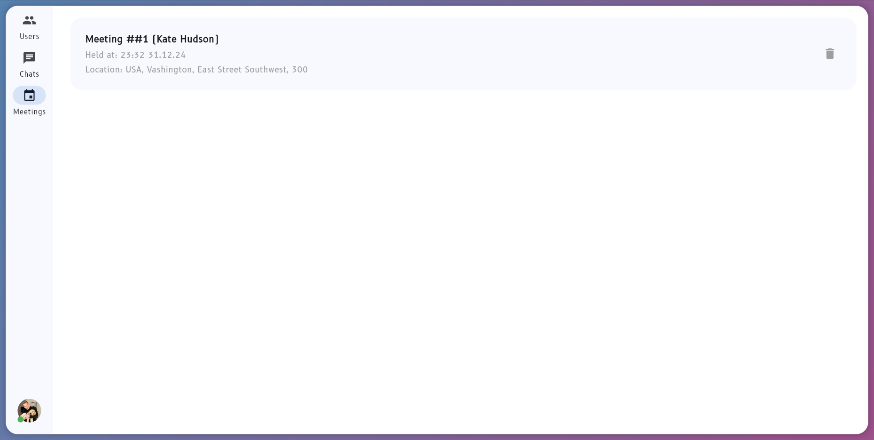


Рисунок 4.7 – Список встреч

В настройках профиля пользователь может изменять свои персональные данные, такие как имя, возраст, фото, хобби и описание (рисунок 4.8). Также доступна возможность создания жалоб, где можно указать описание проблемы.

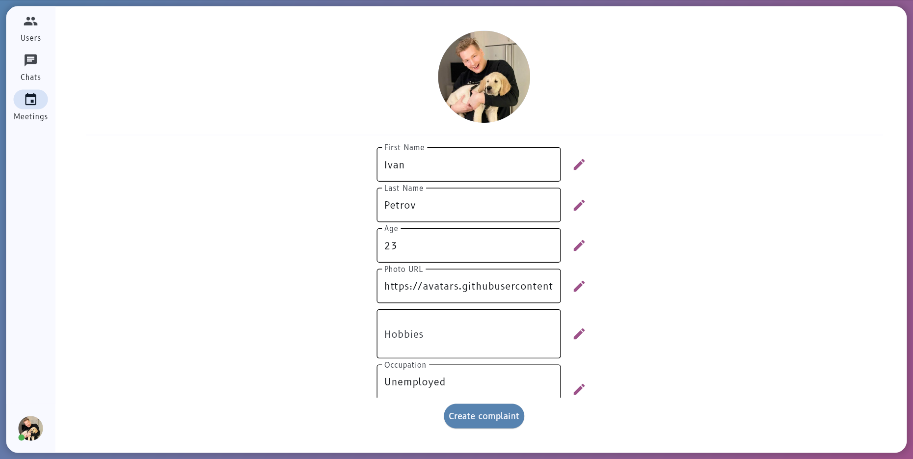


Рисунок 4.8 – Настройки профиля

## **4.3 Панель администратора**

Панель администратора предоставляет доступ к ключевым функциям управления системой и позволяет эффективно модерировать пользователей и их действия. После входа администратор попадает на главную страницу панели, где доступны три основные раздела: управление пользователями, обработка жалоб и просмотр действий.

В разделе пользователей администратор может просматривать список всех зарегистрированных пользователей (рисунок 4.9). Для каждого из них отображается имя, дата регистрации и статус (заблокирован или нет). Администратор может выполнять поиск пользователей по имени и при необходимости блокировать учетные записи.

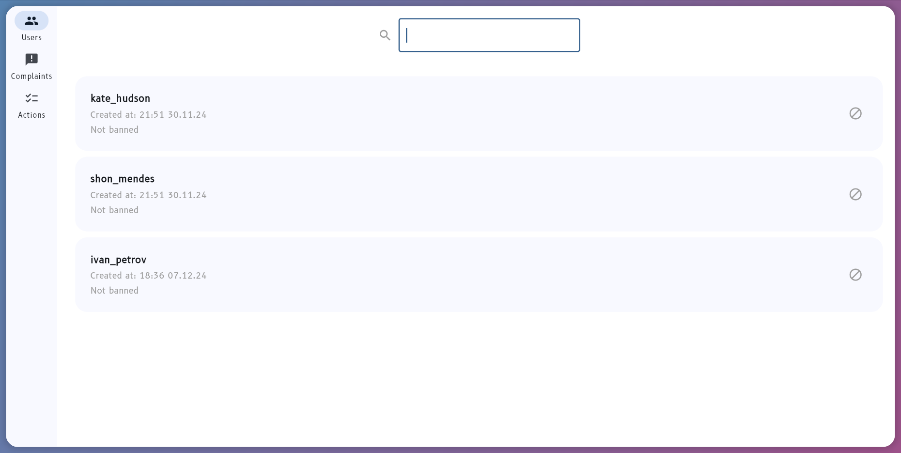


Рисунок 4.9 – Список пользователей

Раздел жалоб предоставляет информацию о жалобах, отправленных пользователями (рисунок 4.10). Для каждой жалобы отображается текст обращения, дата его отправки и информация о пользователе, подавшем жалобу. Администратор может удалять обработанные или неактуальные жалобы.

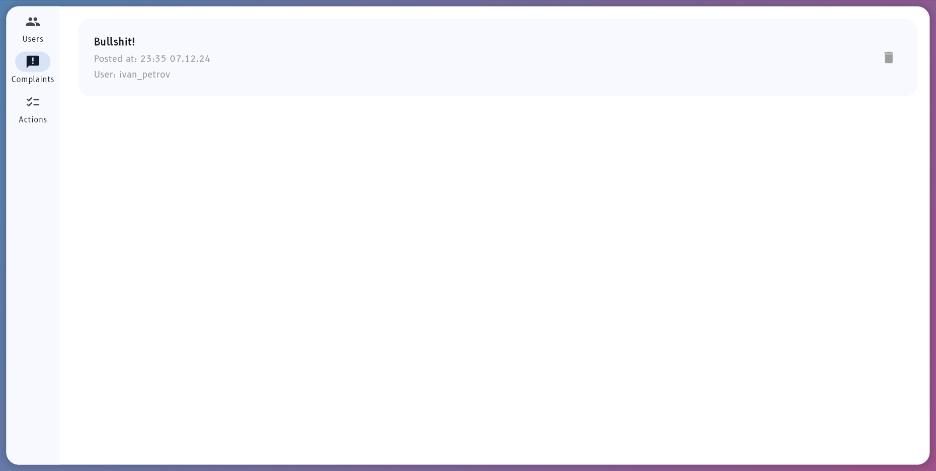


Рисунок 4.10 – Список жалоб

Раздел действий позволяет администратору просматривать список завершенных действий пользователей, включая их описание, дату выполнения и данные об инициаторе (рисунок 4.11). Эта информация помогает отслеживать активность в системе.

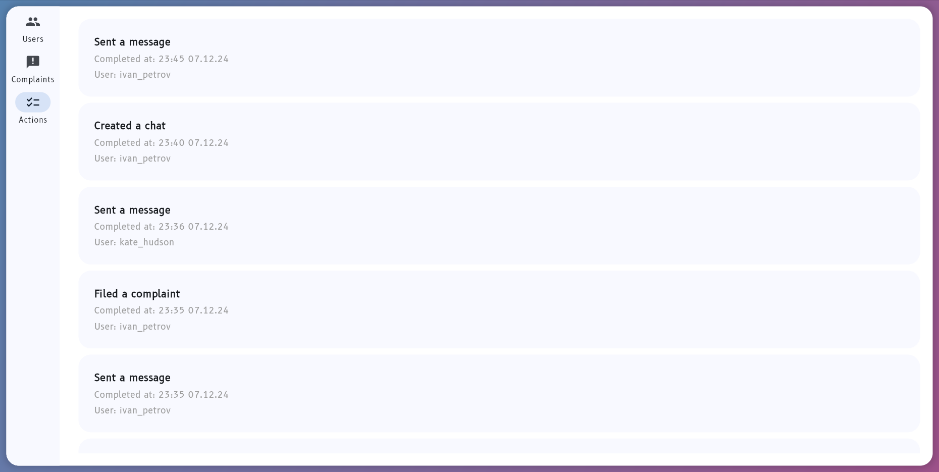


Рисунок 4.11 – Список действий

# **5 ПРОЕКТИРОВАНИЕ РАЗРАБАТЫВАЕМОЙ БАЗЫ ДАННЫХ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ**

## **5.1 Общие принципы проектирования**

Веб-приложение для знакомств требует надежной базы данных, которая будет эффективно обрабатывать и хранить большие объемы информации. Основной задачей проектирования базы данных является создание структуры, которая будет поддерживать работу с пользовательскими данными, профилями, сообщениями, встречами и прочими функциональными элементами приложения.

Исходя из особенностей приложения для знакомств, структура базы данных должна быть гибкой, чтобы поддерживать различные виды информации и взаимодействия пользователей. Будет использоваться реляционная модель, которая обеспечит целостность данных (гарантируется соблюдение правил целостности и согласованности данных при обновлениях, добавлениях и удалениях), масштабируемость (база данных должна быть способна обрабатывать большое количество пользователей и запросов одновременно, что важно для успешной работы в условиях роста аудитории), оптимизацию запросов (для реализации поиска и подбора партнеров важно иметь возможность быстро выполнять сложные SQL-запросы).

## **5.2 Описание структуры базы данных**

В соответствии с проектом базы данных, структура включает несколько основных сущностей:

1 Пользователи (users): основная таблица для хранения данных о пользователях. Включает такие поля, как имя пользователя, пароль и дата создания учетной записи. Взаимодействие с этой таблицей необходимо для авторизации, регистрации и управления пользователями.

2 Профили (profiles): таблица для хранения информации о пользователях, такой как имя, фамилия, возраст, фотография, хобби и описание. Эти данные позволяют персонализировать опыт взаимодействия и служат для создания более детализированных пользовательских профилей.

3 Чаты (chats): таблица, в которой хранятся чаты между пользователями. Благодаря этой таблице, сообщения связывается между собой и самими пользователями.

4 Сообщения (messages): таблица, в которой хранятся отправленные сообщения между пользователями. Это ключевая таблица для реализации чатов, переписок и истории взаимодействий.

5 Местоположения (locations): таблица для хранения информации о локации пользователей и встреч, что важно для поиска по географическим данным, например, для нахождения пользователей поблизости.

6 Встречи (meetings): таблица, которая будет содержать информацию о встречах между пользователями, включая дату и место проведения.

7 Жалобы (complaints): таблица для обработки жалоб пользователей на других участников, что важно для поддержания безопасности на платформе.

8 Роли (roles): таблица, определяющая роли пользователей, такие как администратор, модератор и обычный пользователь.

9 Действия (actions): таблица для хранения различных действий пользователей для отслеживаниях подозрительных действий.

Таблицы связаны между собой с помощью внешних ключей. Например, таблица пользователей связана с таблицей профилей через profile\_id. Для реализации связи «many-to-many», есть таблицы chat\_users и meeting\_users.

В данной базе данных достигнута третья нормальная форма (3NF). Это обеспечивает минимизацию дублирования данных, устранение избыточных зависимостей и улучшение целостности данных. Так как нет таблиц с составными ключами или атрибутами, которые не зависят полностью от всех компонентов ключа, то можно считать, что достигнута BCNF.

Схема базы данных представлена в Приложении Б.

# **ЗАКЛЮЧЕНИЕ**

В ходе выполнения курсового проекта было разработано веб-приложение для знакомств, использующее базу данных PostgreSQL для эффективного хранения и обработки данных пользователей. В процессе работы был спроектирован и реализован ряд ключевых функций, таких как регистрация пользователей, создание и управление профилями, поиск партнеров на основе заданных параметров, реализация взаимодействия пользователей в виде чатов и встреч.

Реализация сервера и интерфейса веб-приложения позволила интегрировать бизнес-логику с базой данных, обеспечивая удобство и быстродействие взаимодействий между пользователями. Использование PostgreSQL для хранения данных подтверждает свою эффективность благодаря гибкости и надежности, что делает его идеальным решением для масштабируемых и высоконагруженных приложений, таких как сервисы знакомств.

В дальнейшем можно улучшить приложение, добавив дополнительные функции, такие как интеграция с внешними сервисами или улучшение алгоритмов.

# **СПИСОК ЛИТЕРАТУРНЫХ ИСТОЧНИКОВ**

[1] Что такое PostgreSQL? [Электронный ресурс]. – Режим доступа: https://azure.microsoft.com/ru-ru/resources/cloud-computing-dictionary/what-is-postgresql – Дата доступа: 01.12.2024.

[2] Документация по PostgreSQL 9.4.1 [Электронный ресурс]. – Режим доступа: https://postgrespro.ru/docs/postgresql/9.4/tutorial-arch – Дата доступа: 01.12.2024.

[3] PostgreSQL: всё, что нужно знать для быстрого старта [Электронный ресурс]. – Режим доступа: https://skillbox.ru/media/code/postgresql-vsye-chto-nuzhno-znat-dlya-bystrogo-starta/ – Дата доступа: 01.12.2024.

[4] PostgreSQL: в чём преимущества управляемой БД перед локальной [Электронный ресурс]. – Режим доступа: https://yandex.cloud/ru/blog/posts/2021/07/managed-postgresql-overview?utm\_referrer=about%3Ablank – Дата доступа: 01.12.2024.

[5] Windows [Электронный ресурс]. – Режим доступа: https://ru.wikipedia.org/wiki/Windows – Дата доступа: 01.12.2024.

[6] Архитектура Windows. Как устроена операционная система Виндовс и как работает? [Электронный ресурс]. – Режим доступа: https://kurets.ru/information-security/317-arhitektyra-windows – Дата доступа: 01.12.2024.

[7] Как устроена Windows: разбираемся в основных компонентах ее архитектуры [Электронный ресурс]. – Режим доступа: https://fileenergy.com/windows/kak-ustroena-windows-razbiraemsya-v-osnovnykh-komponentakh-ee-arkhitektury – Дата доступа: 01.12.2024.

[8] Advantages and disadvantages of windows operating system [Электронный ресурс]. – Режим доступа: https://www.itrelease.com/2019/07/advantages-and-disadvantages-of-windows-operating-system/ – Дата доступа: 01.12.2024.

[9] Карта проникновения онлайн-знакомств в 2023 году [Электронный ресурс]. – Режим доступа: https://infographika.ru/society/internet-and-media-and-social/karta-proniknoveniya-onlayn-znakomstv-v-2023-godu/ – Дата доступа: 01.12.2024.

[10] Python [Электронный ресурс]. – Режим доступа: https://ru.wikipedia.org/wiki/Python – Дата доступа: 01.12.2024.

[11] Как работать с PostgreSQL в Python [Электронный ресурс]. – Режим доступа: https://selectel.ru/blog/tutorials/postgresql-python/#:~:text=Psycopg2%20—%20один%20из%20самых,потому%20уступает%20psycopg2%20в%20производительности – Дата доступа: 01.12.2024.

[12] Building Flutter Apps in Python [Электронный ресурс]. – Режим доступа: https://www.geeksforgeeks.org/building-flutter-apps-in-python/ – Дата доступа: 01.12.2024.

# **ПРИЛОЖЕНИЕ А**

# **(обязательное)**

# **Листинг программного кода**

**init.sql**

CREATE EXTENSION IF NOT EXISTS "uuid-ossp";

CREATE TABLE IF NOT EXISTS roles (

id UUID PRIMARY KEY DEFAULT uuid\_generate\_v4(),

name VARCHAR(50) NOT NULL UNIQUE

);

CREATE TABLE IF NOT EXISTS locations (

id UUID PRIMARY KEY DEFAULT uuid\_generate\_v4(),

country VARCHAR(50) NOT NULL,

city VARCHAR(50) NOT NULL,

address VARCHAR(100)

);

CREATE TABLE IF NOT EXISTS profiles (

id UUID PRIMARY KEY DEFAULT uuid\_generate\_v4(),

first\_name VARCHAR(50) NOT NULL,

last\_name VARCHAR(50) NOT NULL,

age INT NOT NULL,

photo\_url VARCHAR(255) NOT NULL,

hobbies VARCHAR(255),

occupation VARCHAR(100),

description VARCHAR(255),

location\_id UUID NOT NULL REFERENCES locations(id) ON DELETE CASCADE

);

CREATE TABLE IF NOT EXISTS users (

id UUID PRIMARY KEY DEFAULT uuid\_generate\_v4(),

username VARCHAR(50) NOT NULL UNIQUE,

password VARCHAR(100) NOT NULL,

created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

is\_banned BOOLEAN DEFAULT FALSE,

role\_id UUID NOT NULL REFERENCES roles(id) ON DELETE CASCADE,

profile\_id UUID NULL REFERENCES profiles(id) ON DELETE CASCADE

);

CREATE TABLE IF NOT EXISTS actions (

id UUID PRIMARY KEY DEFAULT uuid\_generate\_v4(),

name VARCHAR(100) NOT NULL,

completed\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

user\_id UUID NOT NULL REFERENCES users(id) ON DELETE CASCADE

);

CREATE TABLE IF NOT EXISTS complaints (

id UUID PRIMARY KEY DEFAULT uuid\_generate\_v4(),

content VARCHAR(255) NOT NULL,

posted\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

user\_id UUID NOT NULL REFERENCES users(id) ON DELETE CASCADE

);

CREATE TABLE IF NOT EXISTS chats (

id UUID PRIMARY KEY DEFAULT uuid\_generate\_v4(),

name VARCHAR(100) NOT NULL,

image\_url VARCHAR(255) NOT NULL

);

CREATE TABLE IF NOT EXISTS chat\_users (

id UUID PRIMARY KEY DEFAULT uuid\_generate\_v4(),

chat\_id UUID NOT NULL REFERENCES chats(id) ON DELETE CASCADE,

user\_id UUID NOT NULL REFERENCES users(id) ON DELETE CASCADE,

CONSTRAINT unique\_chat\_user UNIQUE (chat\_id, user\_id)

);

CREATE TABLE IF NOT EXISTS messages (

id UUID PRIMARY KEY DEFAULT uuid\_generate\_v4(),

content VARCHAR(255) NOT NULL,

sent\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

chat\_id UUID NOT NULL REFERENCES chats(id) ON DELETE CASCADE,

user\_id UUID NOT NULL REFERENCES users(id) ON DELETE CASCADE

);

CREATE TABLE IF NOT EXISTS meetings (

id UUID PRIMARY KEY DEFAULT uuid\_generate\_v4(),

name VARCHAR(100) NOT NULL,

held\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

location\_id UUID NOT NULL REFERENCES locations(id) ON DELETE CASCADE

);

CREATE TABLE IF NOT EXISTS meeting\_users (

id UUID PRIMARY KEY DEFAULT uuid\_generate\_v4(),

meeting\_id UUID NOT NULL REFERENCES meetings(id) ON DELETE CASCADE,

user\_id UUID NOT NULL REFERENCES users(id) ON DELETE CASCADE,

CONSTRAINT unique\_meeting\_user UNIQUE (meeting\_id, user\_id)

);

CREATE OR REPLACE FUNCTION delete\_location\_on\_meeting\_delete()

RETURNS TRIGGER AS $$

BEGIN

DELETE FROM locations

WHERE id = OLD.location\_id;

RETURN OLD;

END;

$$ LANGUAGE plpgsql;

CREATE OR REPLACE TRIGGER trigger\_delete\_location

AFTER DELETE ON meetings

FOR EACH ROW

EXECUTE FUNCTION delete\_location\_on\_meeting\_delete();

INSERT INTO roles (id, name)

VALUES

('eb808052-1d0a-42a9-9273-b374ec789adf', 'admin'),

('88a14eaf-5a82-41e7-9224-48e86da7d9f0', 'user')

ON CONFLICT (id) DO NOTHING;

INSERT INTO users (username, password, role\_id)

VALUES

('admin', 'admin', 'eb808052-1d0a-42a9-9273-b374ec789adf')

ON CONFLICT (username) DO NOTHING;

**actions.py**

from psycopg2.extensions import connection as DbConnection

from psycopg2.extras import RealDictCursor

from models import Action

class ActionRepository:

def \_\_init\_\_(self, connection: DbConnection):

self.\_connection = connection

def get\_actions(self) -> list[Action]:

with self.\_connection.cursor(cursor\_factory=RealDictCursor) as cursor:

cursor.execute("SELECT \* FROM actions ORDER BY completed\_at DESC;")

actions\_data = cursor.fetchall()

actions = []

for action\_data in actions\_data:

actions.append(Action(\*\*action\_data))

return actions

def add\_action(self, action: Action) -> None:

with self.\_connection.cursor() as cursor:

cursor.execute(

"INSERT INTO actions (name, user\_id) " "VALUES (%s, %s);",

(action.name, action.user\_id),

)

**chats.py**

from psycopg2.extensions import connection as DbConnection

from psycopg2.extras import RealDictCursor

from models import Chat

class ChatRepository:

def \_\_init\_\_(self, connection: DbConnection):

self.\_connection = connection

def get\_chats\_of\_user(self, user\_id: str) -> list[Chat]:

with self.\_connection.cursor(cursor\_factory=RealDictCursor) as cursor:

cursor.execute(

"SELECT chats.\*, cu2.user\_id AS companion\_id, "

"(SELECT messages.id FROM messages "

"WHERE messages.chat\_id = chats.id "

"ORDER BY messages.sent\_at DESC "

"LIMIT 1) AS last\_message\_id "

"FROM chats JOIN chat\_users cu1 ON "

"chats.id = cu1.chat\_id JOIN chat\_users cu2 "

"ON chats.id = cu2.chat\_id "

"WHERE cu1.user\_id = %s AND cu2.user\_id != cu1.user\_id;",

(user\_id,),

)

chats\_data = cursor.fetchall()

chats = []

for chat\_data in chats\_data:

chats.append(Chat(\*\*chat\_data))

return chats

def get\_chat(self, id: str) -> Chat | None:

if not id or id.isspace():

return None

with self.\_connection.cursor(cursor\_factory=RealDictCursor) as cursor:

cursor.execute(

"SELECT \* FROM chats WHERE id = %s;",

(id,),

)

chat\_data = cursor.fetchone()

if not chat\_data:

return None

else:

return Chat(\*\*chat\_data)

def add\_chat(self, chat: Chat, user\_id: str) -> None:

with self.\_connection.cursor() as cursor:

cursor.execute(

"INSERT INTO chats (name, image\_url) "

"VALUES (%s, %s) RETURNING id;",

(chat.name, chat.image\_url),

)

chat\_id = cursor.fetchone()[0]

cursor.execute(

"INSERT INTO chat\_users (chat\_id, user\_id) "

"VALUES (%s, %s), (%s, %s);",

(

chat\_id,

user\_id,

chat\_id,

chat.companion\_id,

),

)

**complaints.py**

from psycopg2.extensions import connection as DbConnection

from psycopg2.extras import RealDictCursor

from models import Complaint

class ComplaintRepository:

def \_\_init\_\_(self, connection: DbConnection):

self.\_connection = connection

def get\_complaints(self) -> list[Complaint]:

with self.\_connection.cursor(cursor\_factory=RealDictCursor) as cursor:

cursor.execute("SELECT \* FROM complaints;")

complaints\_data = cursor.fetchall()

complaints = []

for complaint\_data in complaints\_data:

complaints.append(Complaint(\*\*complaint\_data))

return complaints

def get\_complaint(self, id: str) -> Complaint | None:

if not id or id.isspace():

return None

with self.\_connection.cursor(cursor\_factory=RealDictCursor) as cursor:

cursor.execute(

"SELECT \* FROM complaints WHERE id = %s;",

(id,),

)

complaint\_data = cursor.fetchone()

if not complaint\_data:

return None

else:

return Complaint(\*\*complaint\_data)

def add\_complaint(self, complaint: Complaint) -> None:

with self.\_connection.cursor() as cursor:

cursor.execute(

"INSERT INTO complaints (content, user\_id) "

"VALUES (%s, %s);",

(complaint.content, complaint.user\_id),

)

def delete\_complaint(self, id: str) -> None:

with self.\_connection.cursor() as cursor:

cursor.execute("DELETE FROM complaints WHERE id = %s;", (id,))

**meetings.py**

from psycopg2.extensions import connection as DbConnection

from psycopg2.extras import RealDictCursor

from models import Meeting

class MeetingRepository:

def \_\_init\_\_(self, connection: DbConnection):

self.\_connection = connection

def get\_meeting(self, id: str) -> Meeting | None:

if not id or id.isspace():

return None

with self.\_connection.cursor(cursor\_factory=RealDictCursor) as cursor:

cursor.execute(

"SELECT \* FROM meetings WHERE id = %s;",

(id,),

)

meeting\_data = cursor.fetchone()

if not meeting\_data:

return None

else:

return Meeting(\*\*meeting\_data)

def get\_meetings\_of\_user(self, user\_id: str) -> list[Meeting]:

with self.\_connection.cursor(cursor\_factory=RealDictCursor) as cursor:

cursor.execute(

"SELECT meetings.\*, mu2.user\_id AS companion\_id, "

"locations.country, locations.city, locations.address "

"FROM meetings JOIN meeting\_users mu1 ON "

"meetings.id = mu1.meeting\_id JOIN meeting\_users mu2 "

"ON meetings.id = mu2.meeting\_id "

"JOIN locations ON location\_id = locations.id "

"WHERE mu1.user\_id = %s AND mu2.user\_id != mu1.user\_id;",

(user\_id,),

)

meetings\_data = cursor.fetchall()

meetings = []

for meeting\_data in meetings\_data:

meetings.append(Meeting(\*\*meeting\_data))

return meetings

def add\_meeting(self, meeting: Meeting, user\_id: str) -> None:

with self.\_connection.cursor() as cursor:

cursor.execute(

"INSERT INTO locations (country, city, address) "

"VALUES (%s, %s, %s) RETURNING id;",

(meeting.country, meeting.city, meeting.address),

)

location\_id = cursor.fetchone()[0]

cursor.execute(

"INSERT INTO meetings (name, held\_at, location\_id) "

"VALUES (%s, %s, %s) RETURNING id;",

(meeting.name, meeting.held\_at, location\_id),

)

meeting\_id = cursor.fetchone()[0]

cursor.execute(

"INSERT INTO meeting\_users (meeting\_id, user\_id) "

"VALUES (%s, %s), (%s, %s);",

(

meeting\_id,

user\_id,

meeting\_id,

meeting.companion\_id,

),

)

def delete\_meeting(self, id: str) -> None:

with self.\_connection.cursor() as cursor:

cursor.execute("DELETE FROM meetings WHERE id = %s;", (id,))

**messages.py**

from psycopg2.extensions import connection as DbConnection

from psycopg2.extras import RealDictCursor

from models import Message

class MessageRepository:

def \_\_init\_\_(self, connection: DbConnection):

self.\_connection = connection

def get\_messages\_by\_chat\_id(self, chat\_id: str) -> list[Message]:

if not chat\_id or chat\_id.isspace():

return []

with self.\_connection.cursor(cursor\_factory=RealDictCursor) as cursor:

cursor.execute(

"SELECT \* FROM messages WHERE chat\_id = %s;",

(chat\_id,),

)

messages\_data = cursor.fetchall()

messages = []

for message\_data in messages\_data:

messages.append(Message(\*\*message\_data))

return messages

def get\_message(self, id: str) -> Message | None:

if not id or id.isspace():

return None

with self.\_connection.cursor(cursor\_factory=RealDictCursor) as cursor:

cursor.execute(

"SELECT \* FROM messages WHERE id = %s;",

(id,),

)

message\_data = cursor.fetchone()

if not message\_data:

return None

else:

return Message(\*\*message\_data)

def add\_message(self, message: Message) -> Message:

with self.\_connection.cursor() as cursor:

cursor.execute(

"INSERT INTO messages (content, chat\_id, user\_id) "

"VALUES (%s, %s, %s) RETURNING id;",

(message.content, message.chat\_id, message.user\_id),

)

message.id = cursor.fetchone()[0]

return message

**profiles.py**

from psycopg2.extensions import connection as DbConnection

from psycopg2.extras import RealDictCursor

from models import Profile

class ProfileRepository:

def \_\_init\_\_(self, connection: DbConnection):

self.\_connection = connection

def get\_profiles(self) -> list[Profile]:

with self.\_connection.cursor(cursor\_factory=RealDictCursor) as cursor:

cursor.execute(

"SELECT profiles.\*, country, city "

"FROM profiles JOIN locations "

"ON location\_id = locations.id;",

(id,),

)

profiles\_data = cursor.fetchall()

profiles = []

for profile\_data in profiles\_data:

profiles.append(Profile(\*\*profile\_data))

return profiles

def get\_profiles\_exclude\_one(self, id: str) -> list[Profile]:

with self.\_connection.cursor(cursor\_factory=RealDictCursor) as cursor:

cursor.execute(

"SELECT profiles.\*, country, city "

"FROM profiles JOIN locations "

"ON location\_id = locations.id WHERE "

"profiles.id != %s;",

(id,),

)

profiles\_data = cursor.fetchall()

profiles = []

for profile\_data in profiles\_data:

profiles.append(Profile(\*\*profile\_data))

return profiles

def get\_profile(self, id: str) -> Profile | None:

if not id or id.isspace():

return None

with self.\_connection.cursor(cursor\_factory=RealDictCursor) as cursor:

cursor.execute(

"SELECT profiles.\*, country, city FROM profiles "

"JOIN locations ON location\_id = locations.id "

"WHERE profiles.id = %s;",

(id,),

)

profile\_data = cursor.fetchone()

if not profile\_data:

return None

else:

return Profile(\*\*profile\_data)

def get\_profile\_by\_user\_id(self, user\_id: str) -> Profile | None:

if not user\_id or user\_id.isspace():

return None

with self.\_connection.cursor(cursor\_factory=RealDictCursor) as cursor:

cursor.execute(

"SELECT profiles.\* FROM users "

"JOIN profiles ON profile\_id = profiles.id "

"WHERE users.id = %s;",

(user\_id,),

)

profile\_data = cursor.fetchone()

if not profile\_data:

return None

else:

return Profile(\*\*profile\_data)

def add\_profile(self, profile: Profile) -> Profile:

with self.\_connection.cursor() as cursor:

cursor.execute(

"INSERT INTO locations (country, city) "

"VALUES (%s, %s) RETURNING id;",

(profile.country, profile.city),

)

location\_id = cursor.fetchone()[0]

cursor.execute(

"INSERT INTO profiles (first\_name, last\_name, age, "

"photo\_url, hobbies, occupation, description, location\_id) "

"VALUES (%s, %s, %s, %s, %s, %s, %s, %s) RETURNING id;",

(

profile.first\_name,

profile.last\_name,

profile.age,

profile.photo\_url,

profile.hobbies,

profile.occupation,

profile.description,

location\_id,

),

)

profile.id = cursor.fetchone()[0]

return profile

def update\_profile(self, profile: Profile) -> None:

with self.\_connection.cursor() as cursor:

cursor.execute(

"UPDATE locations "

"SET country = %s, city = %s "

"WHERE id = (SELECT location\_id "

"FROM profiles WHERE id = %s);",

(profile.country, profile.city, profile.id),

)

cursor.execute(

"UPDATE profiles SET first\_name = %s, "

"last\_name = %s, age = %s, photo\_url = %s, "

"hobbies = %s, occupation = %s, description = %s "

"WHERE id = %s;",

(

profile.first\_name,

profile.last\_name,

profile.age,

profile.photo\_url,

profile.hobbies,

profile.occupation,

profile.description,

profile.id,

),

)

**users.py**

from psycopg2.extensions import connection as DbConnection

from psycopg2.extras import RealDictCursor

from models import User

class UserRepository:

def \_\_init\_\_(self, connection: DbConnection):

self.\_connection = connection

def get\_users(self) -> list[User]:

with self.\_connection.cursor(cursor\_factory=RealDictCursor) as cursor:

cursor.execute(

"SELECT users.\*, roles.name AS role "

"FROM users JOIN roles ON role\_id = roles.id "

"WHERE roles.name = 'user';"

)

users\_data = cursor.fetchall()

users = []

for user\_data in users\_data:

users.append(User(\*\*user\_data))

return users

def get\_user\_by\_profile(self, profile\_id: str) -> User | None:

if not profile\_id or profile\_id.isspace():

return None

with self.\_connection.cursor(cursor\_factory=RealDictCursor) as cursor:

cursor.execute(

"SELECT users.\*, roles.name AS role "

"FROM users JOIN roles ON role\_id = roles.id "

"WHERE profile\_id = %s;",

(profile\_id,),

)

user\_data = cursor.fetchone()

if not user\_data:

return None

else:

return User(\*\*user\_data)

def get\_user\_by\_credentials(

self, username: str, password: str

) -> User | None:

if not username or username.isspace():

return None

if not password or password.isspace():

return None

with self.\_connection.cursor(cursor\_factory=RealDictCursor) as cursor:

cursor.execute(

"SELECT users.\*, roles.name AS role "

"FROM users JOIN roles ON role\_id = roles.id "

"WHERE username = %s AND password = %s;",

(username, password),

)

user\_data = cursor.fetchone()

if not user\_data:

return None

else:

return User(\*\*user\_data)

def get\_user(self, id: str) -> User | None:

if not id or id.isspace():

return None

with self.\_connection.cursor(cursor\_factory=RealDictCursor) as cursor:

cursor.execute(

"SELECT \* FROM users WHERE id = %s;",

(id,),

)

user\_data = cursor.fetchone()

if not user\_data:

return None

else:

return User(\*\*user\_data)

def add\_user(self, user: User) -> None:

with self.\_connection.cursor() as cursor:

cursor.execute(

"SELECT id FROM roles WHERE name = %s;", (user.role,)

)

role\_id = cursor.fetchone()[0]

cursor.execute(

"INSERT INTO users (username, password, is\_banned, "

"role\_id, profile\_id) VALUES (%s, %s, %s, %s, %s);",

(

user.username,

user.password,

user.is\_banned,

role\_id,

user.profile\_id,

),

)

def ban\_user(self, id: str) -> None:

with self.\_connection.cursor() as cursor:

cursor.execute(

"SELECT is\_banned FROM users WHERE id = %s;",

(id,),

)

result = cursor.fetchone()

is\_banned = result[0]

new\_status = not is\_banned

cursor.execute(

"UPDATE users SET is\_banned = %s WHERE id = %s;",

(new\_status, id),

)

# **ПРИЛОЖЕНИЕ Б**

# **(обязательное)**

# **Конечная схема базы данных**