МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ

ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет Информационных технологий

Кафедра Программной инженерии

Специальность 1-40 01 01 «Программное обеспечение информационных технологий»

Направление специальности Программирование интернет-приложений

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**КУРСОВОГО ПРОЕКТА:**

по дисциплине «Базы данных»

Тема «Реализация базы данных сервиса предоставления услуг с применением In-Memory технологий»

Исполнитель

Студент(ка) 3 курса группы 4 Корнелюк Валентин Владимирович

(Ф.И.О.)

Руководитель работы ст. преп. Нистюк Ольга Александровна

(учен. степень, звание, должность, подпись, Ф.И.О.)

Курсовой проект защищен с оценкой

Председатель доц. Блинова Е.А.

(подпись)

Минск 2024

**Содержание**

[ВВЕДЕНИЕ 5](#_Toc185437004)

[1 Поставновка задачи 6](#_Toc185437005)

[1.1 Обзор аналогичных решений 7](#_Toc185437006)

[1.2 Требования к курсовому проекту 10](#_Toc185437007)

[2 Проектирование базы данных 11](#_Toc185437008)

[2.1 Диаграмма базы данных 11](#_Toc185437009)

[2.2 Описание таблиц базы данных 12](#_Toc185437010)

[2.3 Описание связей таблиц 15](#_Toc185437011)

[3 Разработка объектов базы данных 16](#_Toc185437012)

[3.1 Пользователи 16](#_Toc185437013)

[3.2 Индексы 16](#_Toc185437014)

[3.3 Процедуры и функции 16](#_Toc185437015)

[3.4 Вывод по разработке объектов базы данных 19](#_Toc185437016)

[4 Описание процедур импорта и экспорта 20](#_Toc185437017)

[4.2 Описание процедуры экспорта данных 20](#_Toc185437018)

[4.3 Описание процедуры импорта данных 20](#_Toc185437019)

[4.4 Вывод по процедурам импорта и экспорта 20](#_Toc185437020)

[5 Тестирование производительности 21](#_Toc185437021)

[6 Описание технологии и ее применения 23](#_Toc185437022)

[7 Краткое описание демонстрационного приложения 24](#_Toc185437023)

[8 Руководство пользователя 25](#_Toc185437024)

[Заключение 33](#_Toc185437025)

[Список используемых источников 34](#_Toc185437026)

[Приложение A 35](#_Toc185437027)

[Приложение Б 36](#_Toc185437028)

[Приложение В 39](#_Toc185437029)

[Приложение Д 56](#_Toc185437030)

[Приложение Е 59](#_Toc185437031)

# ВВЕДЕНИЕ

Тема курсового проекта «Разработка базы данных для службы предоставления услуг» является актуальной и востребованной в условиях стремительного роста использования онлайн-сервисов для записи на услуги в сфере красоты, здоровья и других профессиональных услуг. Эффективное управление записями и расписанием становится важным аспектом, способствующим повышению качества обслуживания и оптимизации ресурсов.

Анализ существующих решений в данной области показывает, что многие приложения обладают удобным интерфейсом для пользователей, однако внутренние структуры данных не всегда обеспечивают гибкость и масштабируемость, необходимые для удовлетворения современных требований. Увеличение числа пользователей, сложность взаимодействия между клиентами и профессионалами, необходимость анализа данных и управления расписанием приводят к возрастанию потребности в специализированной реляционной базе данных.

Целью данного курсового проекта является создание реляционной базы данных, которая обеспечит функциональность для создания, хранения, изменения и удаления записей об услугах, клиентских данных и расписании профессионалов. База данных также будет включать возможности для администрирования и управления безопасностью, что важно для защиты конфиденциальной информации пользователей.

Реляционная база данных, основанная на концепции таблиц и отношений между ними, была выбрана как основа проекта. В качестве системы управления базами данных была выбрана Oracle, которая благодаря высокой производительности и поддержке больших объемов данных позволяет строить надежные и масштабируемые решения. Oracle обладает широким набором инструментов для создания сложных структур данных, что делает её отличным выбором для разработки гибкой и функциональной базы данных, удовлетворяющей требованиям бизнеса.

Основная часть работы будет посвящена техническим аспектам разработки базы данных, в том числе проектированию структуры таблиц, а также обеспечению безопасности и целостности данных. Данный подход к разработке базы данных будет способствовать повышению эффективности работы службы и улучшению взаимодействия между клиентами и профессионалами, что в конечном итоге приведет к повышению удовлетворенности пользователей и увеличению конкурентоспособности сервиса.

# Поставновка задачи

Целью данной работы является проектирование базы данных для программного средства «Сервис предоставления услуг» с применением In-Memory технологий. В качестве модели данных используется реляционная модель, которая позволит систематизировать и структурировать информацию, соответствующую общим требованиям к данным в базе данных. Это обеспечит высокую производительность и оптимизацию скорости обработки запросов, что особенно важно для решения задач в условиях растущего объема данных и числа пользователей.

База данных — это организованная структура, предназначенная для хранения информации, систематизированная таким образом, чтобы эти материалы могли быть найдены и обработаны с помощью электронной вычислительной машины. Реляционная база данных основана на таблицах и отношениях между ними, что делает её удобной для обработки и организации данных, требующих высокой целостности и строгого соблюдения связей. В качестве базы данных выбран Oracle благодаря его высокой производительности, надёжности и широкому функциональному набору, который позволяет обрабатывать большие объёмы данных, не теряя в скорости и точности.

Проектирование базы данных требует учёта множества факторов, таких как структура данных, модели взаимодействия между элементами и оптимизация запросов. In-Memory технологии, применяемые в данном проекте, помогут ускорить доступ к данным, обеспечивая более быструю обработку и высокую отзывчивость системы.

В основной части работы будут подробно рассмотрены все этапы проектирования, включая структуру таблиц, методы обеспечения безопасности и поддержание целостности данных. Такой подход к разработке базы данных способствует повышению эффективности работы сервиса и удобству взаимодействия для пользователей.

Основные требования к приложению:

* Возможность регистрации и авторизации пользователей;
* Настойка личного аккаунта;
* Управление временными слотами мастера;
* Управление постами мастера;
* Обеспечение взаимодействия пользователей;
* Управление записями пользователя.

Весь доступный функционал будет представлен на UML-схеме (Приложение А).

## Обзор аналогичных решений

Одним из первых этапов в создании программного продукта является анализ прототипов и литературных источников.

При анализе было обращено внимание на отсутствие идентичных аналогов разрабатываемому приложению. Вместо этого, были обнаружены приложения, частично содержащие функционал, схожий с предполагаемым функционалом разрабатываемого приложения. В данном разделе будет проведено сравнение и анализ указанных приложений с целью изучения их функционала и возможностей в контексте нашего приложения. Это позволит выявить преимущества и недостатки существующих решений, а также определить уникальные особенности и добавленную ценность разрабатываемого приложения.

Первый аналог – «fokanail.by**»**. На рисунке 1.1 изображён интерфейс приложения:

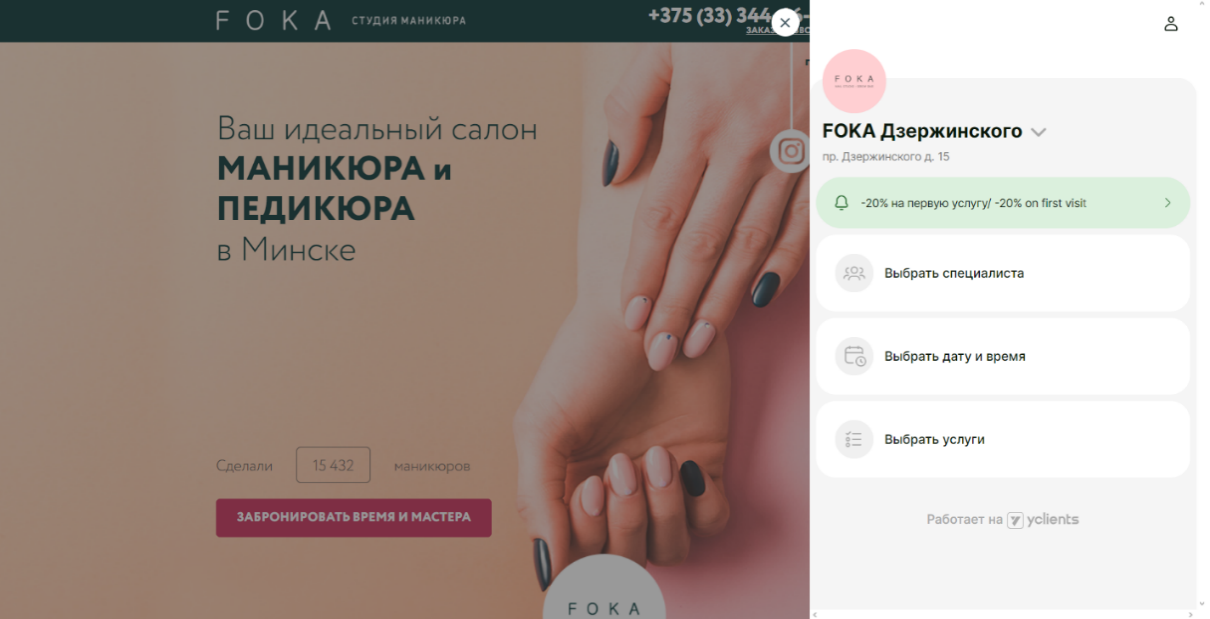
****

Рисунок 1.1 – Интерфейс приложения «foknail»

В приложении я обратил внимание на несколько удобных и функциональных аспектов организации записи на услуги:

* Интерфейс для записи на услуги. В рамках базы данных это подразумевает следующие сущности:

1. услуга – хранит информацию об услугах (название, описание);
2. категория услуг – группировка услуг для облегчения навигации.

* Удобный календарь для выбора даты и времени: Календарь, предоставляемый на сайте, позволяет клиентам легко выбирать удобные даты и время для записи на услуги. Проанализировав этот элемент можно выделить следующие сущности:

1. ячейка записи – хранит информацию о бронировании;
2. расписание – доступные для записи временные интервалы.

* Понятный процесс выбора услуг: Структура сайта делает процесс выбора услуг простым и прозрачным. Ясное представление доступных услуг и их описаний позволяет клиентам быстро найти нужную услугу и ознакомиться с ее описанием перед бронированием. Тут можно выделить следующие сущности:

1. опции записи – хранит детали процедуры;
2. мастер – хранит данные о мастерах, которые представляют услуги.

Таким образом, приложение «foknail.by» строится на четко продуманно структуре данных, где каждая сущность имеет свою роль и связи, что обеспечивает высокое удобство пользования.

В качестве второго аналога рассмотрим приложение «DIKIDI». На рисунке 1.2 представлен интерфейс приложения:

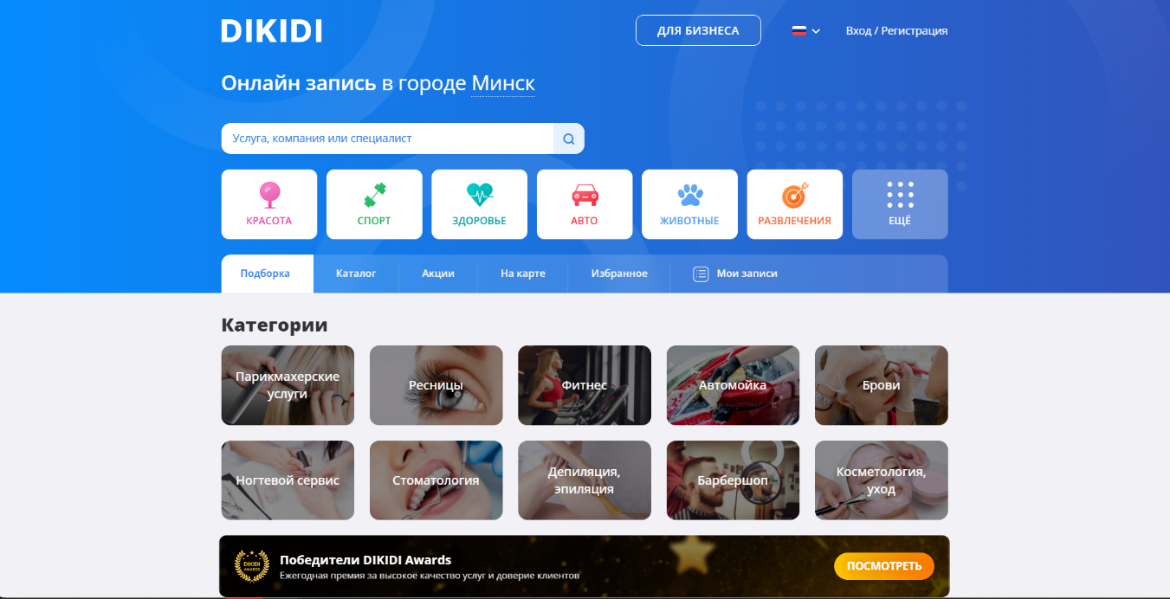


Рисунок 1.2 – Интерфейс приложения «DIKIDI»

При рассмотрении сайта были отмечены следующие плюсы:

* Простой процесс выбора услуг по подборкам: структура сайта обеспечивает легкость и ясность при выборе услуг. Понятное представление доступных услуг и их описаний позволяет клиентам быстро найти нужную услугу и ознакомиться с ее деталями;
* Гибкие возможности персонализации записи: Клиентам предоставляется возможность выбора дополнительных опций или параметров при записи на услуги, таких как выбор мастера или добавление дополнительных услуг. Это позволяет настраивать бронирование в соответствии с индивидуальными предпочтениями и потребностями клиентов.

В приложении «DIKIDI» можно выделить те же основные сущности, что и в предыдущем анализируемом аналоге. Здесь также присутствуют клиенты, мастера и услуги, которые составляют основу системы записи. Клиенты являются пользователями, записывающимися на услуги, мастера представляют специалистов, предоставляющих эти услуги, а сами услуги описываются в виде доступных для бронирования позиций с определёнными характеристиками, такими как название, описание, стоимость и длительность.

Таким образом, структура данных остаётся схожей, поскольку любая система записи на услуги требует похожей логики хранения данных и взаимодействия между ними: клиенты выбирают услуги, предоставляемые конкретными мастерами, с возможностью выбора даты и времени.

В качестве третьего аналога рассмотрим приложение **«**Instagram». Функционал приложения представлен на рисунке 1.3. В данном приложении есть две интересующие нас функции, это механизм подписки и функция создания публикаций, которые могут быть адаптированы в структуру разрабатываемого приложения.

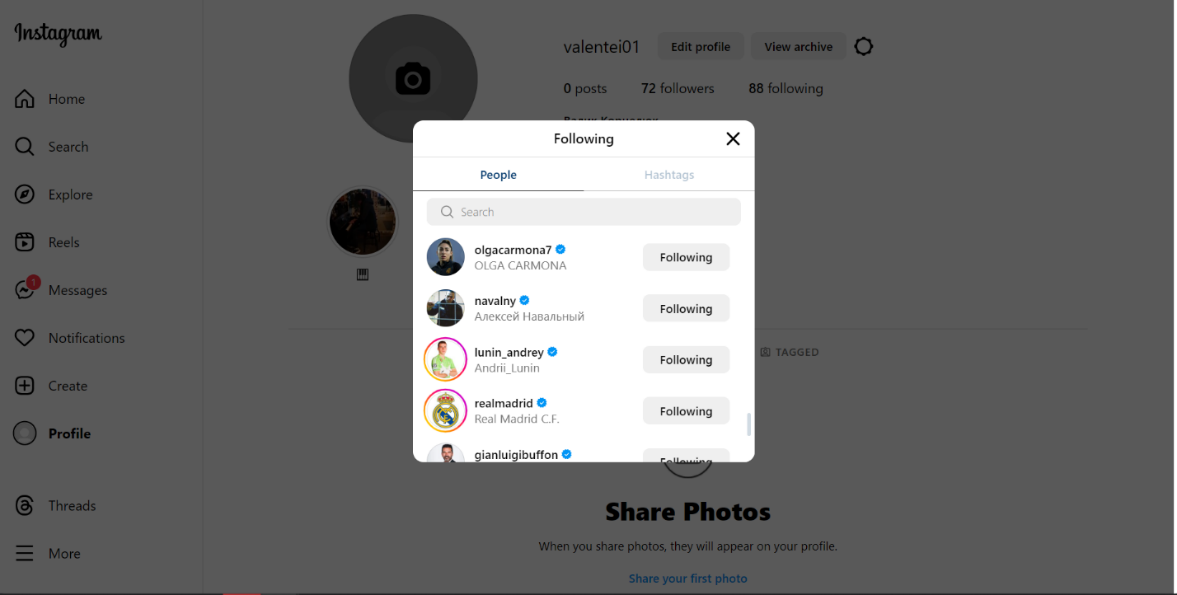


Рисунок 1.3 – Интерфейс приложения «Instagram»

При анализе этого приложения было обращено внимание на следующие аспекты, которые могут быть применены к организации аккаунтов пользователей и механизму подписки в разрабатываемом приложении:

* Механизм подписки на другие аккаунты: в Instagram пользователи могут подписываться на аккаунты других пользователей, чтобы получать обновления и контент. Эта функция может быть реализована в разрабатываемом приложении для возможности пользователям подписываться на профили мастеров, чтобы держать наиболее часто посещаемых мастеров в быстром доступе. Для этого в базе данных нужно создать связь «многие ко многим» через дополнительную таблицу связи»
* Возможность публикации постов: в Instagram пользователи имеют возможность публиковать посты, включающие в себя фотографии, видео или текстовые материалы. Эта функция позволяет пользователям делиться своим творчеством, работами или полезной информацией с аудиторией. Тут можно выделить сущность “Пост”, которая будет связана с сущностью мастера связью «один ко многим»;

Таким образом, приложение **«Instagram»** предлагает эффективные механизмы, которые можно адаптировать в разрабатываемом приложении через продуманную базу данных. Это позволит организовать подписки, публикации и взаимодействие между пользователями, обеспечив при этом удобное хранение и обработку данных.

## Требования к курсовому проекту

Конечным результатом данного курсового проекта является разработанная и созданная реляционная база данный для разрабатываемого приложения. Необходимо, чтобы проектирование базы данных было сделано так, чтобы конечные данные в базе данных соответствовали всем требованиям, которые были описаны в данном разделе.

Функционально должны быть выполнены следующие задачи:

* определение ролей (администратор, пользователь-мастер, пользователь-клиент);
* управление пользователями (регистрация, авторизация);
* настройка личного аккаунта (включая установку фотографии, выбор имени пользователя, указание электронной почты, города, номера телефона);
* управление временными слотами мастера (добавление, изменение, просмотр свободных/зарезервированных слотов);
* управление постами мастера (добавление, удаление);
* обеспечение взаимодействия пользователей (поиск мастера, подписка/отписка на мастера, просмотр свободных временных слотов, резервирование временного слота, просмотр постов мастера);
* просмотр записей (просмотр предстоящих резервирований, просмотр истории резервирований).

Доступ к данным должен осуществляться через соответствующие процедуры, обеспечивая тем самым контролируемый и безопасный доступ. Реализация базы данных предусматривает проведение импорта данных из JSON файлов и экспорта данных в формат JSON. Важным этапом является тестирование производительности базы данных на таблице, содержащей не менее 100 000 строк. Технологический компонент задачи включает в себя применение технологий Oracle, а также демонстрацию применения выбранной in-memory технологии в контексте разработанного приложения.

# Проектирование базы данных

## 2.1 Диаграмма базы данных

В самом начале разработки нужно определить из каких таблиц будет состоять база данных, какие связи будут между ними и какие типы данных будут представлять поля для заполнения таблиц.

Диаграмма базы данных таблиц (Database Table Diagram) – это визуальное представление структуры базы данных и отношений между таблицами, которые хранятся в этой базе данных Диаграмма таблиц и их отношения представлены на рисунке 2.1.

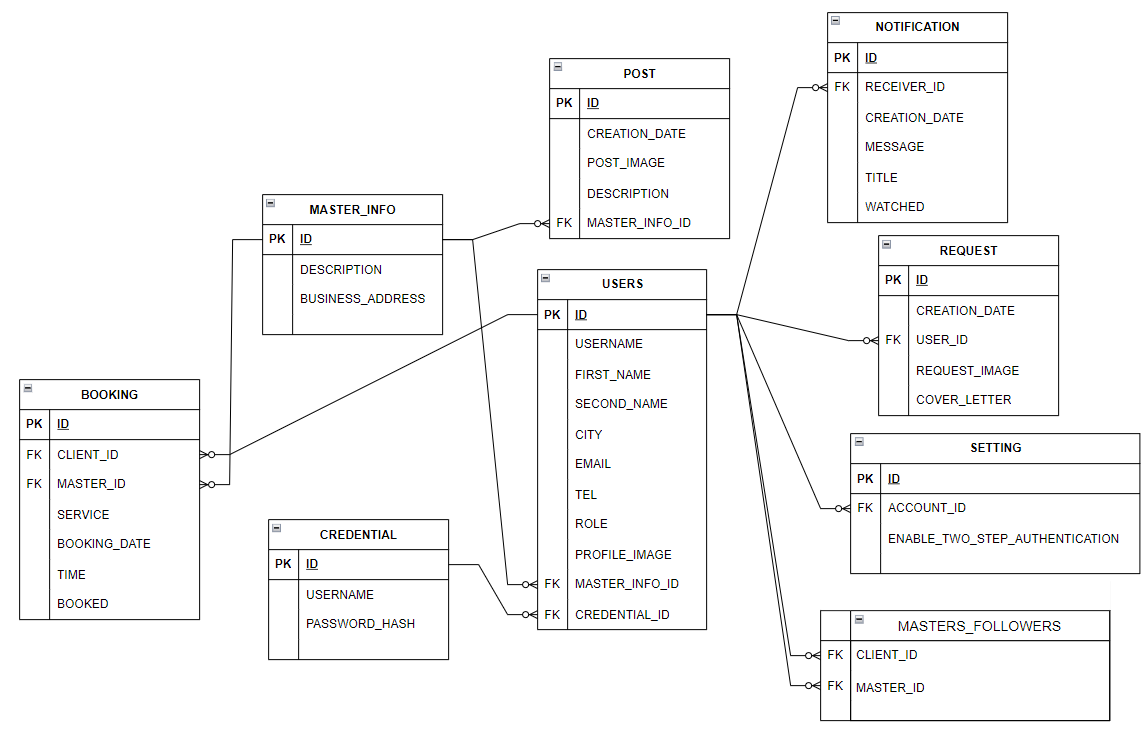


Рисунок 2.1 – Диаграмма таблиц базы данных

Проектирование базы данных играет ключевую роль в обеспечении эффективности и устойчивости работы любой системы. Разработка структуры таблиц, связей между ними и их характеристик должна учитывать, как текущие, так и потенциальные будущие требования к системе. Грамотно спроектированная база данных обеспечивает упрощение работы с данными и улучшает производительность запросов и повышает их читаемость.

## Описание таблиц базы данных

В базе данных представлены 9 таблиц: USERS, CREDENTIAL, MASTER\_INFO, BOOKING, NOTIFICATION, POST, REQUEST, SETTING, MASTERS\_FOLLOWERS.

Таблица USERS хранит информацию о пользователях. Структура представлена в таблице 2.1:

Таблица 2.1 – Столбцы таблицы USERS

|  |  |  |
| --- | --- | --- |
| Наименование | Тип | Ограничение целостности |
| Id | NUMBER | PRIMARY KEY |
| Username | VARCHAR2(255) | NOT NULL, Unique |
| First\_name | VARCHAR2(255) | NOT NULL |
| Second\_name | VARCHAR2(255) | NOT NULL, UNIQUE |
| City | VARCHAR2(100) | NOT NULL |
| email | VARCHAR2(255) | DEFAULT |
| Tel | VARCHAR2(20) | NOT NULL |
| Role | VARCHAR2(50) | NOT NULL |
| Profile\_image | BLOB |  |
| Credential\_id | NUMBER | NOT NULL |
| Master\_info\_id | NUMBER |  |

Таблица MASTER\_INFO содержит дополнительную информацию о пользователе. Структура представлена в таблице 2.2

Таблица 2.2 – Столбцы таблицы MASTER\_INFO

|  |  |  |
| --- | --- | --- |
| Наименование | Тип | Ограничение целостности |
| Id | NUMBER | PRIMARY KEY |
| description | VARCHAR2(255) | NOT NULL |
| business\_address | VARCHAR2(255) | NOT NULL |

Таблица CREDENTIAL содержит информацию об учетных данных пользователя. Структура представлена в таблице 2.3

Таблица 2.3 – Столбцы таблицы CREDENTIAL

|  |  |  |
| --- | --- | --- |
| Наименование | Тип | Ограничение целостности |
| Id | NUMBER | PRIMARY KEY |
| username | VARCHAR2(255) | NOT NULL |
| password\_hash | VARCHAR2(255) | NOT NULL |

Таблица POST хранит информацию о постах мастеров. Структура представлена в таблице 2.4:

Таблица 2.4 – Столбцы таблицы POST

|  |  |  |
| --- | --- | --- |
| Наименование | Тип | Ограничение целостности |
| Id | NUMBER | PRIMARY KEY |
| creation\_date | DATE | NOT NULL |
| post\_image | BLOB | NOT NULL |
| description | VARCHAR2(255) |  |
| master\_info\_id | NUMBER | NOT NULL |

Таблица BOOKING хранит информацию о временных ячейках для записей на услуги. Она позволяет организовать удобное управление расписанием мастеров и клиентов. Данные таблицы используются для бронирования записей и анализа загруженности. Структура представлена в таблице 2.5:

Таблица 2.5 – Столбцы таблицы BOOKING

|  |  |  |
| --- | --- | --- |
| Наименование | Тип | Ограничение целостности |
| Id | NUMBER | PRIMARY KEY |
| client\_id | NUMBER |  |
| master\_id | NUMBER | NOT NULL |
| service | VARCHAR2(255) |  |
| booking\_date | DATE | NOT NULL |
| time | TIMESTAMP | NOT NULL |
| booked | NUMBER(1) | CHECK (booked IN (0, 1) |

Таблица NOTIFICATION хранит информацию об уведомлениях пользователей. Данные из этой таблицы используются для отправки актуальных уведомлений и ведения их истории для каждого пользователя. Структура представлена в таблице 2.6:

Таблица 2.6 – Столбцы таблицы NOTIFICATION

|  |  |  |
| --- | --- | --- |
| Наименование | Тип | Ограничение целостности |
| Id | NUMBER | PRIMARY KEY |
| receiver\_id | NUMBER | NOT NULL |
| creation\_date | DATE | NOT NULL |
| message | VARCHAR2(255) | NOT NULL |
| title | VARCHAR2(255) | NOT NULL |
| time | TIMESTAMP | NOT NULL |
| watched | NUMBER(1) | CHECK (watched IN (0, 1) |

Таблица REQUEST хранит информацию об запросах пользователей на то чтобы стать мастером. Эти данные позволяют администратору эффективно обрабатывать заявки и управлять назначением новых мастеров. Структура представлена в таблице 2.7:

Таблица 2.7 – Столбцы таблицы REQUEST

|  |  |  |
| --- | --- | --- |
| Наименование | Тип | Ограничение целостности |
| Id | NUMBER | PRIMARY KEY |
| user\_id | NUMBER | NOT NULL |
| creation\_date | DATE | NOT NULL |
| request\_image | BLOB | NOT NULL |
| cover\_letter | VARCHAR2(255) | NOT NULL |

Таблица SETTING хранит информацию о настройках приложения для пользователя. Данная таблица позволяет индивидуально настраивать параметры работы приложения для каждого пользователя, обеспечивая персонализированный опыт использования. Благодаря этому повышается удобство взаимодействия с системой и удовлетворение потребностей различных категорий пользователей. Структура представлена в таблице 2.8:

Таблица 2.8 – Столбцы таблицы SETTING

|  |  |  |
| --- | --- | --- |
| Наименование | Тип | Ограничение целостности |
| Id | NUMBER | PRIMARY KEY |
| account\_id | NUMBER | NOT NULL |
| enable\_two\_step\_authentication | NUMBER(1) | CHECK (enable\_two\_step\_authentication IN (0, 1) |

Таблица MASTERS\_FOLLOSERS хранит информацию о пользователеях и мастеров, на которых они подписаны. Эта таблица обеспечивает возможность отслеживания подписок пользователей на мастеров, что позволяет организовать удобное взаимодействие пользователей. Структура представлена в таблице 2.9:

Таблица 2.9 – Столбцы таблицы MASTERS\_FOLLOWERS

|  |  |  |
| --- | --- | --- |
| Наименование | Тип | Ограничение целостности |
| client\_id | NUMBER | NOT NULL |
| master\_id | NUMBER | NOT NULL |

Представленная структура базы данных обеспечивает эффективное и надежное хранение информации, необходимой для функционирования системы. Предусмотренные ограничения целостности и связи между таблицами гарантируют целостность данных и согласованность информации в системе. Листинг создания таблиц находится в приложении Б.

## Описание связей таблиц

Раздел посвящен описанию связей между таблицами базы данных, которые обеспечивают логическую связанность и целостность информации в системе. Описание всех связей приводится в таблице 2.10:

Таблица 2.10 – Описание связей таблиц

|  |  |  |
| --- | --- | --- |
| Наименование столбца первичного ключа | Наименование столбца внешнего ключа | Описание связи |
| BOOKING.CLIENT\_ID | USERS.ID | Связь многие к одному для связи клиента и его записей |
| BOOKING.MASTER\_ID | MASTER\_INFO.ID | Связь многие к одному для связи мастера и его записей |
| POST.MASTER\_INFO\_ID | MASTER\_INFO.ID | Связь многие к одному для связи мастера и его постов |
| USERS.MASTER\_INFO\_ID | MASTER\_INFO.ID | Связь один к одному для связи общей информации пользователя и деталей информации о мастере |
| USERS.CREDENTIAL\_ID | CREDENTIAL.ID | Связь один к одному для связи информации пользователя с его учетными данными |
| SETTING.ACCOUNT\_ID | USERS.ID | Связь один к одному для связи информации пользователя и настроек приложения для него |
| REQUEST.USER\_ID | USERS.ID | Связь один к одному для связи запроса с пользователем от для которого он создан |
| NOTIFICATION.RECEIVER\_ID | USERS.ID | Связь многие к одному для связи пользователя с его уведомлениями |
| MASTERS\_FOLLOWERS.CLIENT\_ID | USERS.ID | Связь многие ко многим чтобы связать пользователя и его подписки |
| MASTERS\_FOLLOWERS.MASTER\_ID | USERS.ID | Связь многие ко многим чтобы связать мастера и его подписчиков |

Определение и реализация связей между таблицами обеспечивает целостность данных и эффективное выполнение запросов в базе данных. Такие связи помогают поддерживать логическую структуру системы, улучшая производительность и гарантируя согласованность информации.

# Разработка объектов базы данных

## Пользователи

В курсовом проекте существует 3 типа пользователей: мастер, который может добавлять новые временные слоты для записи, клиент, который может записывать на услуги к мастеру, администратор, который может сделать обычного пользователя мастером. Создание пользователей представлено в листинге 3.1.

Листинг 3.1 – Создание пользователей

CREATE USER ADMIN IDENTIFIED BY 12345 PROFILE ADMIN\_PROFILE;

CREATE USER MASTER IDENTIFIED BY 12345 PROFILE USERS\_PROFILE;

CREATE USER CLIENT IDENTIFIED BY 12345 PROFILE USERS\_PROFILE;

Данный набор пользователей позволит разграничить доступ к функционалу и объектам базы данных.

## Индексы

Индексы в базе данных используются для ускорения операций поиска и выборки данных, что значительно повышает производительность при работе с большими объемами информации. Они позволяют быстрее находить нужные строки в таблицах, уменьшая время выполнения запросов, особенно при работе с большими данными и сложными запросами с условиями фильтрации и сортировки.

В разрабатываемой базе данных были добавлен индекс для столбца booking\_date таблицы Booking, так как это значение наиболее часто используется для условий выборки, также был добавлен индекс на столбцы username, first\_name и second\_name таблицы Users, так как эти значения часто используются для поиска пользователя. Код создания индексов представлен в листинге 3.2.

Листинг 3.2 – Создание индексов

CREATE INDEX idx\_users\_name ON Users (username, first\_name, second\_name);

CREATE INDEX idx\_booking\_date ON Booking (booking\_date) LOCAL;

Данные индексы позволят ускорить выполнение запросов к указанным таблицам.

## Процедуры и функции

Процедуры и функции являются важным инструментом для обеспечения эффективного взаимодействия с данными. Они позволяют инкапсулировать логику обработки данных, такие как вставка, обновление или удаление записей, в одну или несколько предопределенных операций. Использование хранимых процедур и функций позволяет ускорить выполнение часто повторяющихся операций, улучшить безопасность, ограничив доступ к данным, а также повысить поддержку целостности данных, гарантируя выполнение всей бизнес-логики в рамках одной транзакции.

Процедуры и функции, разработанные в рамках курсового проекта:

* add\_user – позволяет добавить в базу данных нового пользователя;
* find\_user\_by\_id – позволяет получить информацию о пользователе по его идентификатору;
* find\_user\_by\_username – позволяет получить информацию о пользователе по уникальному имени пользователя;
* exists\_user\_by\_username – позволяет проверить существует ли в базе данных пользователь с заданным именем пользователя;
* search\_master – осуществляет поиск мастеров;
* search\_user – осуществляет поиск всех пользователей
* update\_user\_client – обновляет информацию пользователя с ролью клиента;
* update\_user\_master – обновляет информацию пользователя с ролью мастера;
* make\_master – меняет роль пользователя с клиента на мастера.
* follow\_master – добавляет пользователя подписку на мастера;
* unfollow\_master – удаляет подписку на мастера;
* client\_following\_master – проверяет подписан ли пользователь на мастера. Реализация функции представлена в листинге 3.3;

Листинг 3.3 – Реализация функции client\_following\_master.

CREATE OR REPLACE FUNCTION client\_following\_master(

p\_client\_id NUMBER,

p\_master\_id NUMBER

)

RETURN NUMBER IS

v\_res NUMBER;

BEGIN

SELECT COUNT(\*) INTO v\_res

FROM masters\_followers

WHERE client\_id = p\_client\_id AND master\_id = p\_master\_id;

IF v\_res > 0 THEN

RETURN 1;

ELSE

RETURN 0;

END IF;

END;

* find\_notifications\_by\_receiver – позволяет найти список уведомлений для пользователя;
* add\_notification\_to\_user – добавляет новое уведомление пользователю;
* mark\_notifications\_as\_read – помечает уведомления пользователя как прочитанные;
* count\_unread\_notifications – позволяет получить количество непрочитанный пользователем уведомлений. Реализация функции представлена в листинге 3.4;

Листинг 3.4 – Реализация функции count\_unread\_notifications.

CREATE OR REPLACE FUNCTION count\_unread\_notifications(

p\_receiver\_id IN NUMBER

) RETURN NUMBER IS

v\_count NUMBER;

BEGIN

select count(\*) into v\_count from Notification where receiver\_id = p\_receiver\_id AND watched = 0;

return v\_count;

EXCEPTION

WHEN OTHERS THEN

RAISE\_APPLICATION\_ERROR(-20001, 'Error counting unread notifications: ' || SQLERRM);

END;

* find\_post\_by\_id – позволяет получить информацию о конкретном посте.
* delete\_post – удаляет заданный пост;
* add\_post – добавляет новый пост;
* get\_all\_requests – позволяет получить список запросов от пользователей на то, чтобы стать мастером;
* find\_request\_by\_id – позволяет получить информацию о конкретном запросе пользователя;
* add\_request – добавляет новый запрос от пользователя;
* delete\_request – удаляет запрос пользователя из базы данных;
* exists\_request\_by\_user – позволяет проверить существует ли уже запрос от пользователя;
* find\_setting\_by\_id – позволяет найти настройки приложения для конкретного пользователя;
* change\_setting – изменят настройки приложения для конкретного пользователя;
* add\_booking – добавляет новую ячейку для бронирования;
* update\_booking – изменяет ячейку для бронирования;
* find\_booking\_by\_id – позволяет получить данные о конкретной записи;
* find\_bookings\_in\_range – позволяет получить список записей в заданном промежутке;
* find\_free\_bookings\_in\_range – позволяет получить список свободных для бронироавния записей в заданном промежутке;
* find\_bookings\_by\_date – позволяет получить список записей для определенной даты;
* find\_free\_bookings\_by\_date - позволяет получить список незабронированных записей для определенной даты;
* find\_bookings\_by\_client – позволяет найти список предстоящих записей для клиента;
* find\_booked\_bookings\_by\_master – позволяет получить список предстоящих записей для мастера;
* find\_history\_bookings\_by\_client – позволяет получить историю записей клиента;
* find\_history\_bookings\_by\_master – позволяет получить историю записей мастера;
* delete\_booking – удаляет запись;
* delete\_expired\_bookings – удаляет просроченные незабронированные записи.

Процедуры используются для выполнения последовательности действий, таких как вставка, обновление или удаление данных, а функции могут использоваться для возврата результатов вычислений или выполнения проверок. Это помогает оптимизировать производительность, минимизировать количество обращений к базе данных и гарантировать соблюдение бизнес-логики на уровне базы данных. Кроме того, их использование повышает безопасность, так как доступ к данным может быть ограничен через контроль прав доступа к процедурам и функциям. Реализация всех процедур и функций находится в приложении В.

## Вывод по разработке объектов базы данных

В этой главе был рассмотрен процесс разработки объектов базы данных, которые необходимы для эффективного функционирования приложения. В частности, была организована система пользователей, включающая различные роли с разграничением прав доступа, что позволяет обеспечить безопасность и контроль над действиями в приложении. Также были созданы индексы для оптимизации поиска данных и ускорения выполнения запросов, что существенно повышает производительность при работе с большими объемами информации.

Особое внимание было уделено разработке хранимых процедур и функций, которые позволяют централизованно реализовывать бизнес-логику, улучшая поддержку целостности данных и безопасность. Использование таких объектов, как процедуры для добавления, обновления и удаления данных, а также функции для выполнения проверок и вычислений, способствует повышению эффективности работы с базой данных.

Таким образом, проектирование и реализация объектов базы данных обеспечивают надежную и оптимизированную работу приложения, гарантируя соблюдение бизнес-логики и поддержку высоких стандартов безопасности и производительности.

# Описание процедур импорта и экспорта

В данной курсовой работе реализованы процедуры экспорта и импорта данных из JSON файла в базу данных таблицы Booking и наоборот. Код функций экспорта и импорта представлен в приложении Д.

## Описание процедуры экспорта данных

Процедура export\_bookings предназначена для экспорта данных из таблицы Bookings в формат JSON и сохранения их в файл. Процедура принимает один параметр file\_name, который представляет собой имя файла, в который будут экспортированы данные.

При вызове процедуры открывается файл в заранее определенной директории с использованием пакета UTL\_FILE, а затем из таблицы Booking последовательно выбираются записи с полями id, client\_id, master\_id, service, booking\_date, time и booked. Для каждой строки с помощью функции JSON\_OBJECT формируется JSON-объект, который записывается в файл построчно. После обработки всех записей файл закрывается. В случае возникновения ошибки процедура закрывает все открытые файлы и генерирует сообщение об ошибке с использованием RAISE\_APPLICATION\_ERROR.

## Описание процедуры импорта данных

Процедура import\_bookings предназначена для импорта данных из формата JSON в таблицу Booking. Процедура принимает один параметр file\_name, который представляет собой имя файла, из которого будут импортированы данные.

При вызове процедуры файл открывается из директории JSON\_DIR с помощью пакета UTL\_FILE в режиме чтения. Построчно из файла считываются строки, содержащие данные в формате JSON. Для каждой строки выполняется разбор JSON-объекта с использованием функции JSON\_TABLE, которая извлекает значения полей id, client\_id, master\_id, service, booking\_date, time и booked и сохраняет их во временные переменные. Если значение поля id отсутствует, запись вставляется в таблицу Booking без указания id, иначе запись добавляется с заданным значением id. После обработки всех строк файл закрывается, а изменения фиксируются с помощью COMMIT. В случае возникновения ошибок процедура завершает выполнение, закрывает все открытые файлы и вызывает исключение с описанием ошибки.

## Вывод по процедурам импорта и экспорта

В данном разделе был рассмотрен процесс работы с данными в формате JSON, включая их импорт и экспорт. Были разработаны процедуры для загрузки данных из JSON-файла в таблицу базы данных и выгрузки данных из таблицы в JSON-файл.

# Тестирование производительности

Для проверки производительности базы данных необходимо заполнить ее большим количеством различных данных и узнать время выполнения одного запроса.

Для данной задачи был разработаны 2 процедуры, которые позволяют вставить в таблицу 100 000 строк. Код данных процедур представлен в приложении Е.

После заполнения таблицы было выполнен запрос для поиска записей в заданном диапазоне. Результат выполнения представлен на рисунке 5.1

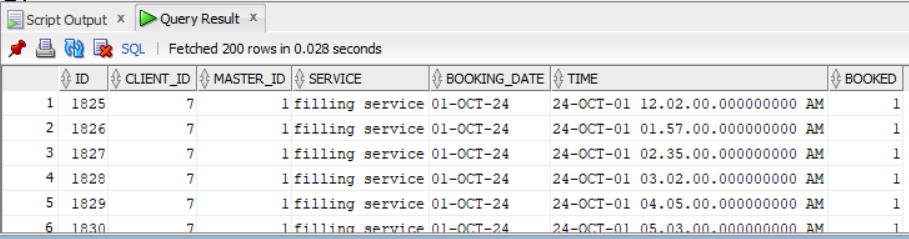


Рисунок 5.1 – Результат выполнения запроса.

Для улучшения производительности выполнения запросов, где основным условием поиска является колонка типа DATE, рекомендуется использовать секционирование таблицы с интервалом в 1 месяц. Это позволит значительно ускорить обработку данных, разделяя таблицу на меньшие, более управляемые части. Результат выполнения запроса после внедрения секционирования представлен на рисунке 5.2.

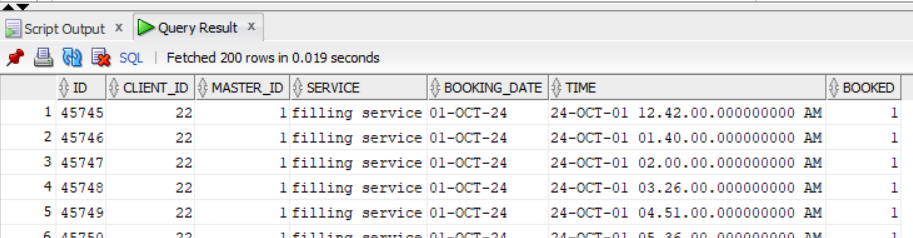


Рисунок 5.2 – Результат выполнения запроса для секционированной таблицы.

Кроме того, для дополнительного повышения производительности можно добавить индекс на колонку, которая используется в условиях поиска, в данном случае на поле типа DATE. Индексация позволит ускорить операции выборки и повысить общую эффективность запросов, особенно в случае больших объемов данных. В сочетании с секционированием таблицы, индексация обеспечит еще большую оптимизацию работы базы данных. Результат выполнения запроса после добавления индекса представлен на рисунке 5.3.

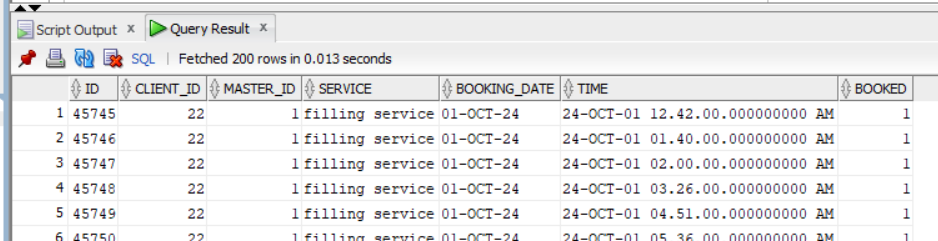


Рисунок 5.3 – Результат выполнения запроса после добавления индекса.

После внедрения секционирования таблицы и добавления индекса на колонку типа DATE, использующуюся в условиях поиска, мы наблюдаем улучшение производительности базы данных. Время выполнения запросов сократилось, что подтверждает эффективность таких оптимизаций. Секционирование позволяет базе данных обрабатывать запросы быстрее, ограничивая выборку данных только нужными диапазонами дат, а индексирование улучшает скорость поиска в пределах этих диапазонов. В результате, запросы выполняются быстрее, что особенно важно при работе с большими объемами данных.

# Описание технологии и ее применения

В данном разделе рассматривается использование in-memory базы данных для хранения одноразовых паролей (One-Time password, OTP) в процессе двухфакторной аутентификации пользователей. In-memory базы данных представляют собой системы, которые сохраняют данные в оперативной памяти, а не на диске. Это позволяет достичь высоких скоростей чтения и записи, так как операции с памятью значительно быстрее, чем с дисковыми хранилищами.

В разрабатываемом приложении используется in-memory база данных H2, которая является высокопроизводительной, легковесной реляционной базой данных, что делает ее идеальной для хранения временных данных.

При запуске приложения в H2 создается таблица для хранения OTP. Код создания таблицы представлен в листинге 6.1

Листинг 3.1 – Создание таблицы для хранения OTP.

CREATE TABLE IF NOT EXISTS user2fa (

id BIGINT AUTO\_INCREMENT PRIMARY KEY,

username varchar(255) NOT NULL,

otp\_password VARCHAR(10) NOT NULL,

token VARCHAR(255) NOT NULL,

expires\_at TIMESTAMP NOT NULL

)

Каждый раз, когда пользователь инициирует запрос на идентификацию, система проверяет включена ли в настройках профиля пользователя двухфакторная идентификация, и, если да, то перенаправляет пользователя на страницу подтверждения OPT. В это же время генерируется новый OTP, который добавляется в таблицу и отправляется пользователю на электронную почту. По истечении времени или успешном подтверждении идентификации OTP удаляется из базы данных, что исключает возможность повторного использования.

В итоге использование in-memory базы данных H2 для хранения одноразовых паролей (OTP) в рамках двухфакторной идентификации обеспечило высокую производительность и эффективность обработки запросов.

# Краткое описание демонстрационного приложения

В рамках проекта также было реализовано приложение, предназначенное для демонстрации работы базы данных, спроектированной в ходе выполнения проекта. Приложение построено с использованием технологий Spring на стороне сервера и React на стороне клиента, что обеспечивает удобство использования и позволяет легко масштабировать приложение.

Приложение предоставляет следующие возможности:

* Регистрация и авторизация пользователей. Пользователи могут зарегистрироваться в приложении, а также включить двухфакторную идентификацию с использованием одноразовых паролей, которые генерируются сервером и отправляются пользователю на почту;
* Работа с профилями пользователей и мастеров. Пользователи могут редактировать свои профили, добавлять информацию о себе, а также подписываться на мастеров. Мастера также могут добавить дополнительную профессиональную информацию в профиль.;
* Управление записями на услуги. Приложение реализует функционал бронирования услуг. Пользователи могут просматривать доступные временные слоты у мастеров и записываться на услуги, а мастера могут управлять своими бронированиями;
* Публикация постов. Мастера могут создавать посты для привлечения клиентов. Посты могут содержать изображение и текстовое описание;
* Уведомления и запросы. Пользователи получают уведомления о важных событиях. Также приложение поддерживает возможность подачи заявок на роль мастера, которые обрабатываются администратором;
* Управление настройками аккаунта. В приложении реализован раздел для настройки пользовательских предпочтений.

Приложение демонстрирует возможности спроектированной базы данных, включая работу с in-memory технологиями, реализацию сложных связей между таблицами, обработку больших объемов данных и обеспечение целостности информации.

# Руководство пользователя

При входе в приложение пользователь попадает на страницу авторизации где он может войти в приложение используя уникальное имя пользователя и пароль. Форма авторизации представлена на рисунке 8.1

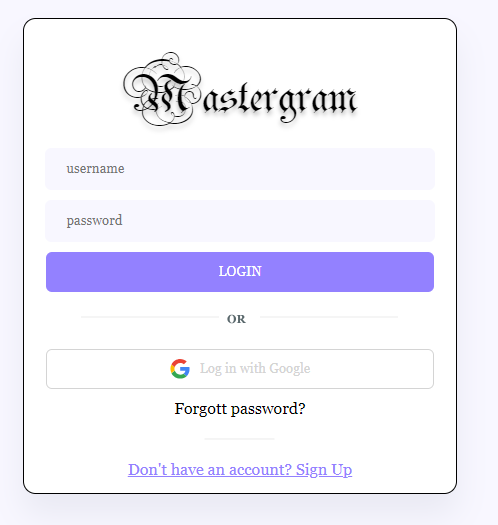


Рисунок 8.1 – Форма авторизации.

Если у пользователя еще нет аккаунта в приложении он может перейти на страницу регистрации. Форма регистрации представлена на рисунке 8.2.



Рисунок 8.2 – Форма регистрации.

Если пользователь включил двух факторную авторизацию, то после ввода данных на странице авторизации он перенаправляется на страницу ввода одноразового пароля. Этот дополнительный шаг предназначен для повышения уровня безопасности и гарантирует, что только авторизованный пользователь сможет получить доступ к приложению. Форма для ввода OTP представления на рисунке 8.3

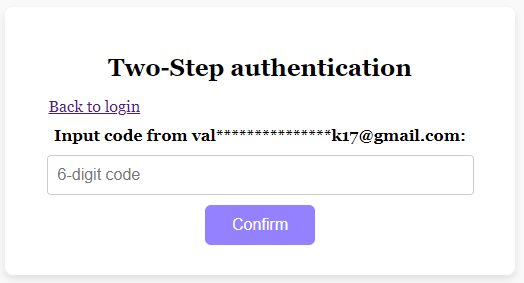


Рисунок 8.3 – Форма для ввода OTP.

После авторизации пользователь попадает на главную страницу в зависимости от его роли в приложении. Для пользователя с ролью «клиент» на главной странице показываются предстоящие услуги, на которые записался пользователь. Главная страница для пользователя с ролью «клиент» представлена на рисунке 8.4

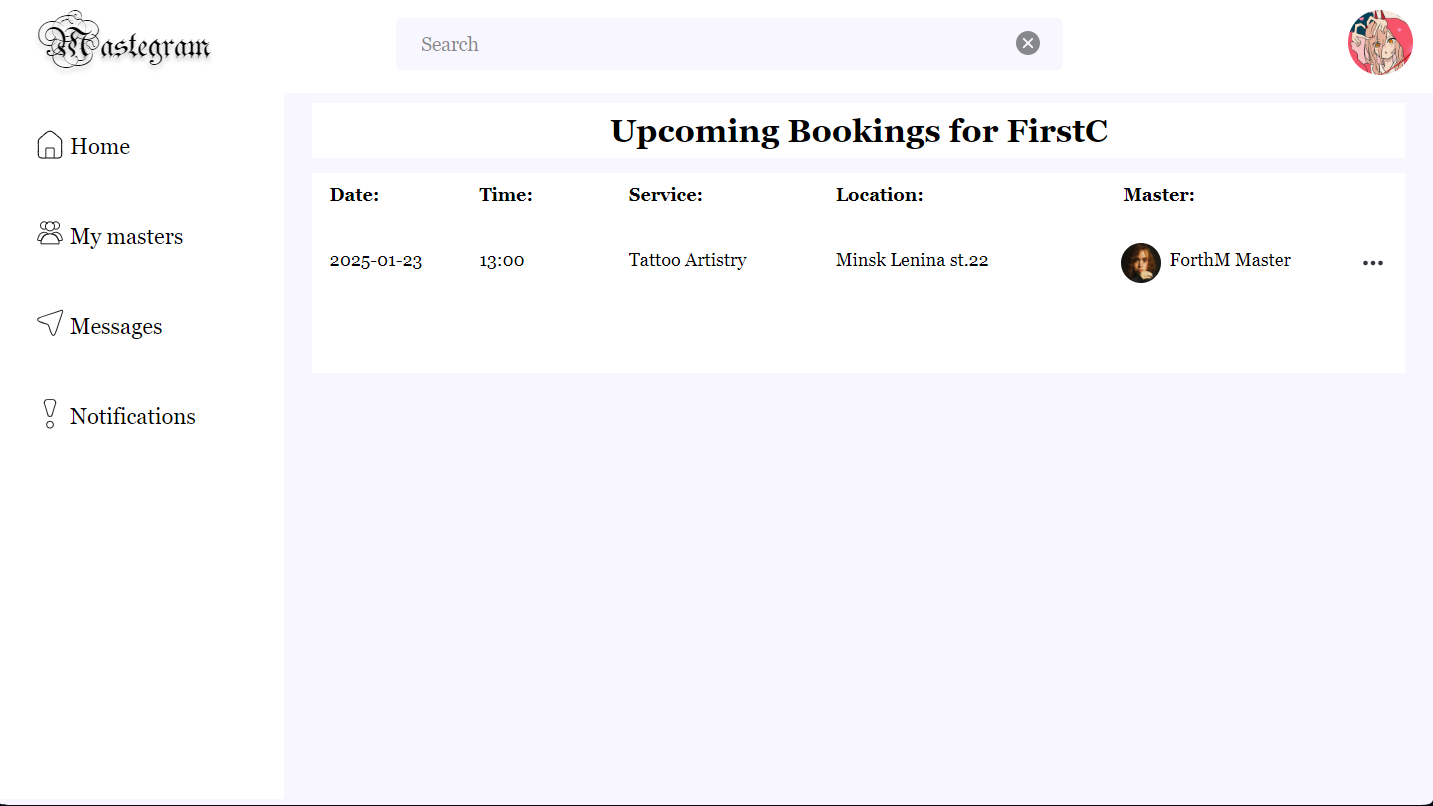


Рисунок 8.3 – Главная страница для пользователя-клиента

Пользователь имеет возможность просматривать детали своей записи, включая информацию о выбранной услуге, времени и мастере. В данном разделе интерфейса предусмотрена возможность отмены бронирования, если пользователь решит изменить свои планы. Пример формы с деталями бронирования, где отображаются все основные сведения о записи, представлен на рисунке 8.4

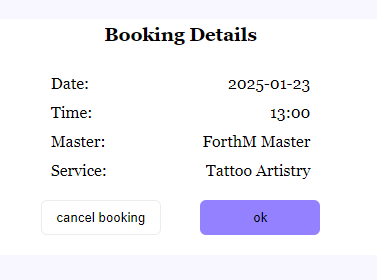


Рисунок 8.3 – Детали записи

В шапке приложения располагается строка поиска, с помощью которой пользователь может найти мастера. Пример результат поиска представлен на рисунке 8.4.

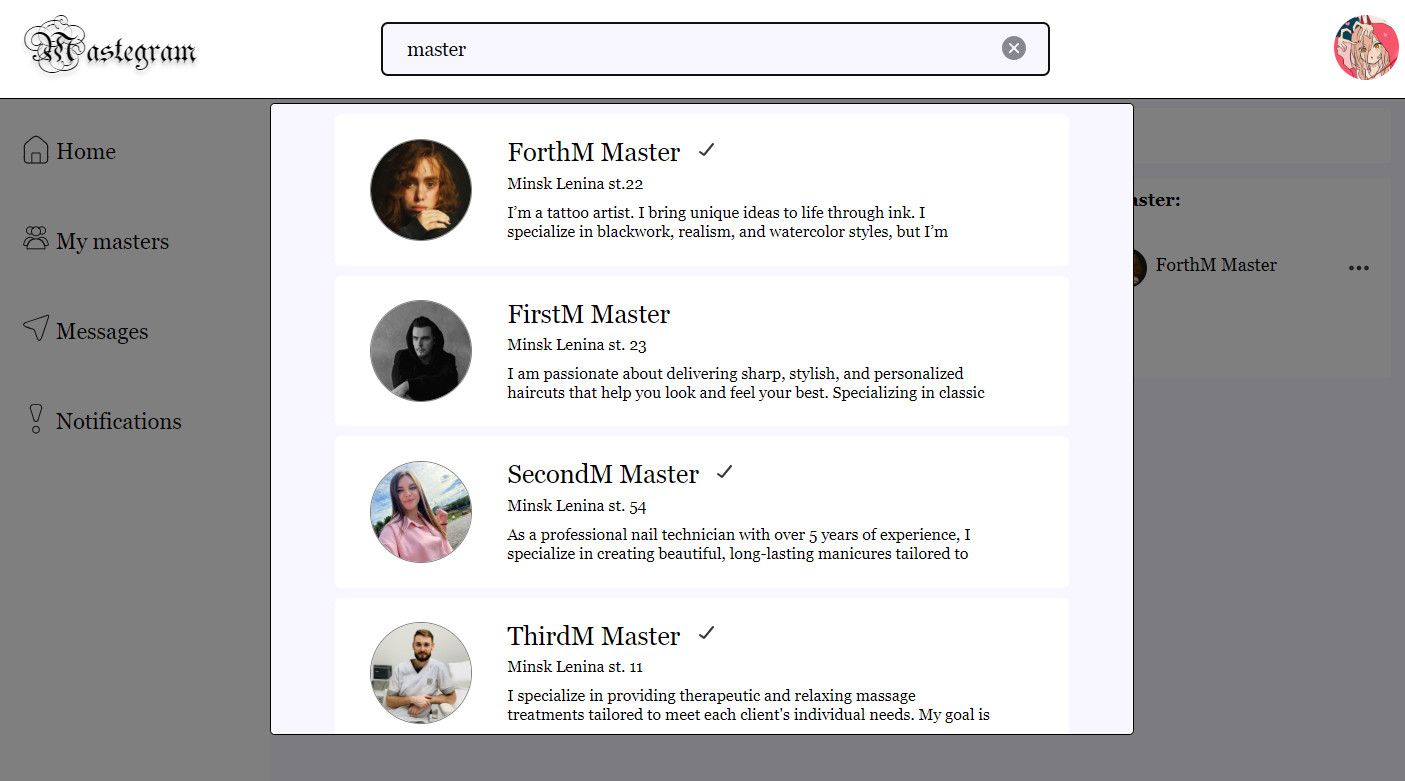


Рисунок 8.4 – Результат выполнения поиска мастера.

После выполнения поиска пользователь может перейти на страницу интересующего его мастера, нажав на соответствующего мастера. На странице мастера пользователь может посмотреть список свободных для бронирования записей и записаться на подходящее ему время. Пример страницы мастера представлен на рисунке 8.5.

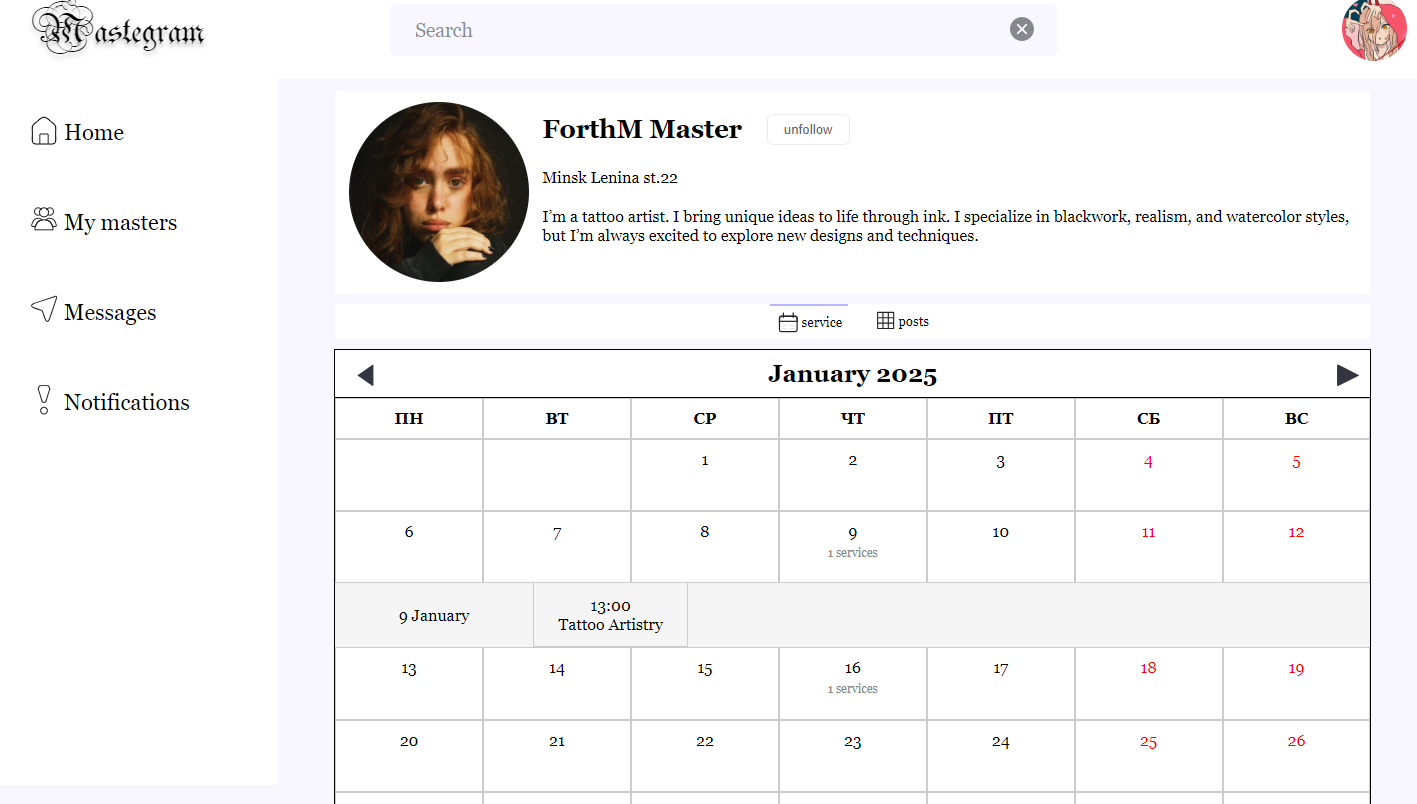


Рисунок 8.5 – Пример страницы мастера.

Пример формы подтверждения бронирования представлен на рисунке 8.6.

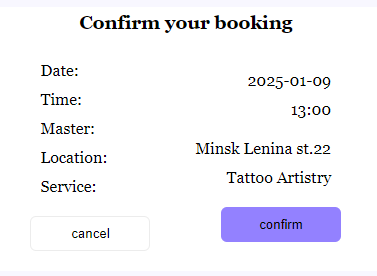


Рисунок 8.5 – Форма подтверждения бронирования.

Кроме того, пользователь имеет возможность ознакомиться с постами, опубликованными мастером. В разделе постов представлен список всех доступных публикаций. Для более детального ознакомления с содержанием, пользователь может открыть конкретный пост и прочитать описание. Пример раздела с постами мастера представлен на рисунке 8.6.

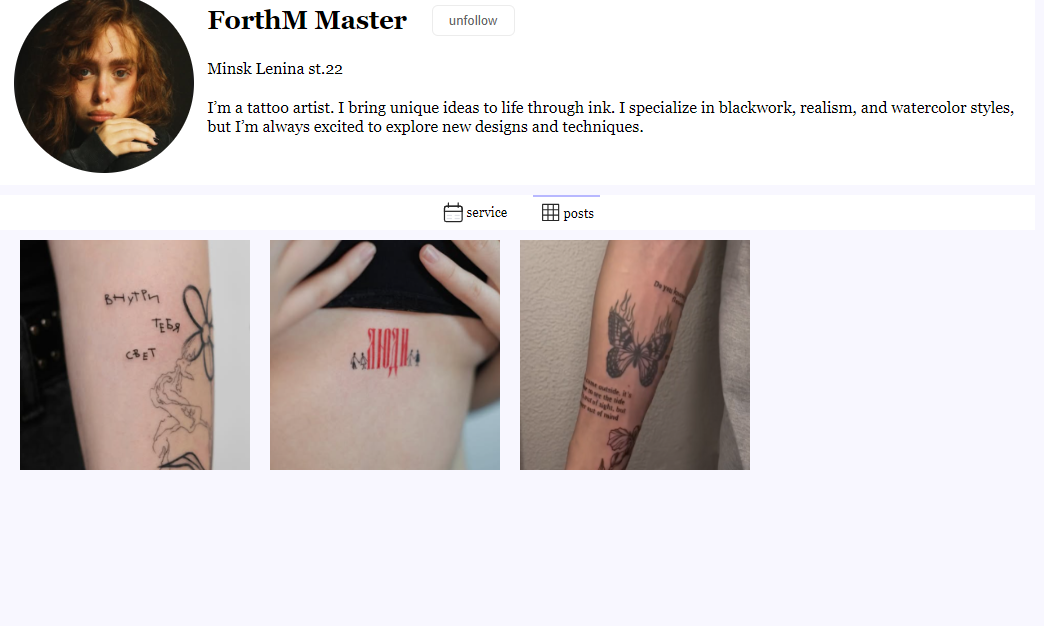


Рисунок 8.6 – Раздел с постами мастера.

Также на странице мастере пользователь может подписаться на мастера или отписаться от него. После подписки на мастера его можно будет найти в списке подписок пользователя, куда можно попасть через боковое меню приложения нажав на элемент «My masters». Пример страницы с подписками пользователя представлен на рисунке 8.7.

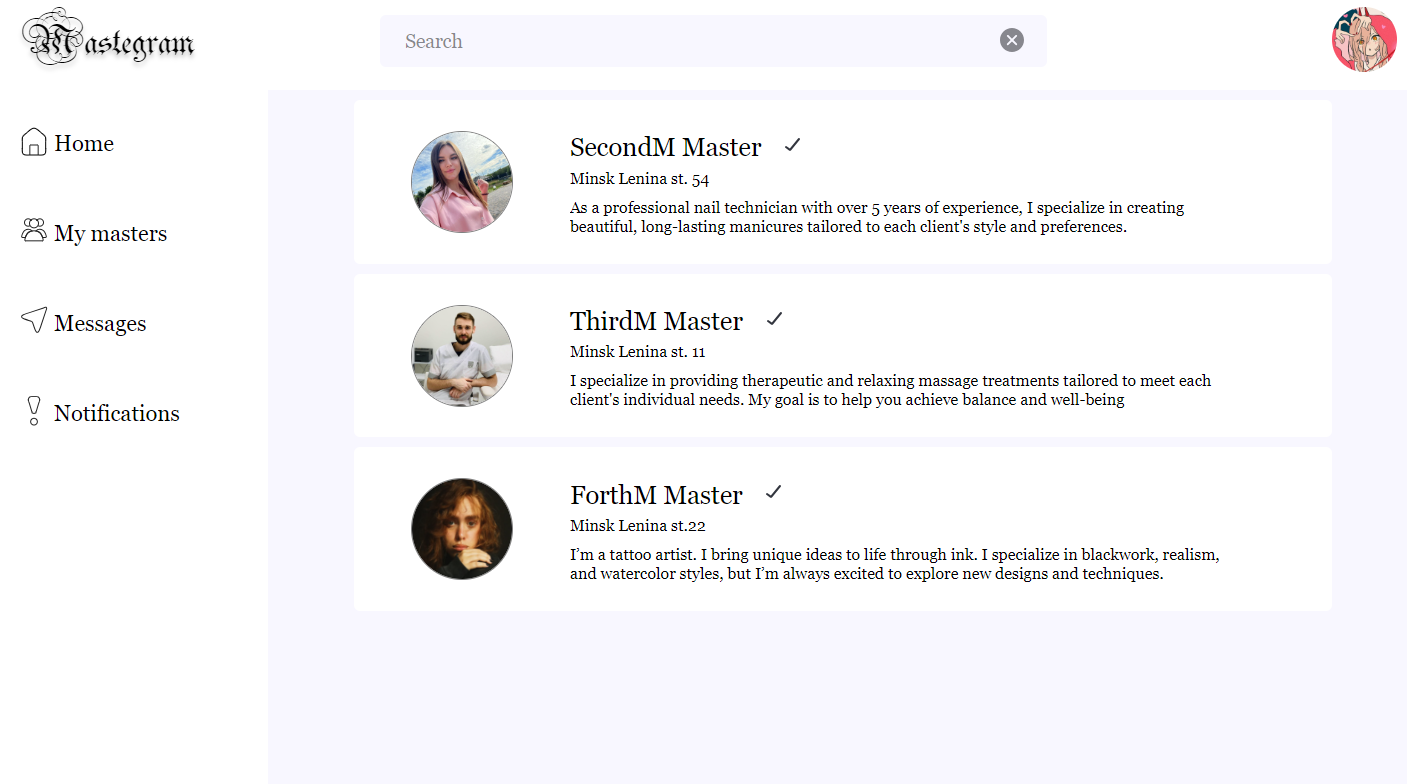


Рисунок 8.7 – Раздел с подписками пользователя.

С помощью бокового меню пользователь имеет возможность быстро перейти в раздел уведомлений, где он может ознакомиться с актуальной информацией о различных событиях, связанных с его аккаунтом и активностью на платформе. Пример раздела с уведомлениями представлен на рисунке 8.8.

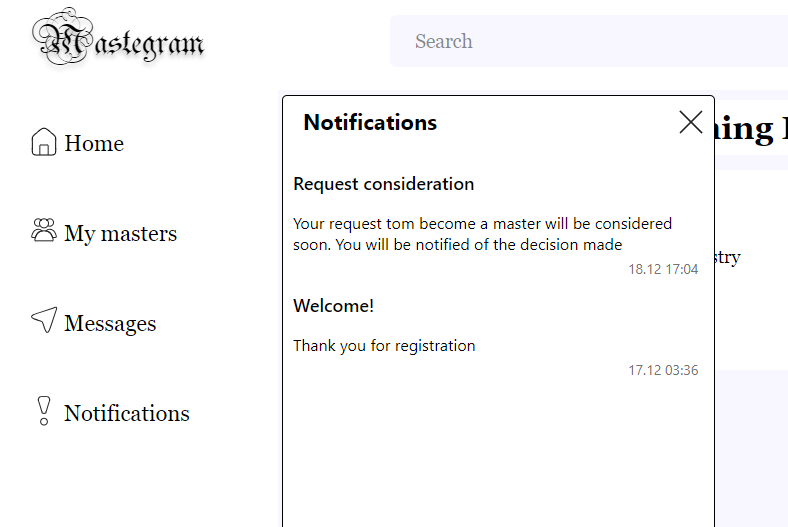


Рисунок 8.8 – Раздел уведомлений.

В шапке приложения пользователь может нажать на иконку профиля, после чего появиться выпадающее меню для настройки профиля, где пользователь может перейти на страницы редактирования профиля, настроек или выйти из приложения. Пример выпадающего меню представлен на рисунке 8.9.

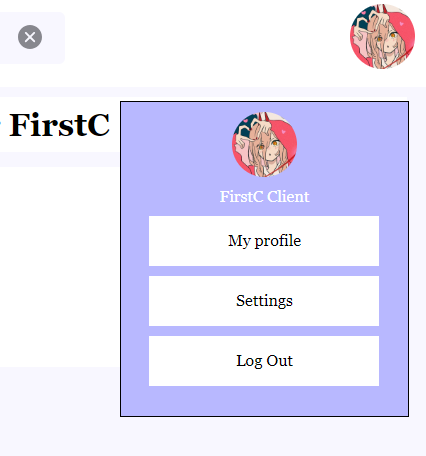


Рисунок 8.8 – Примеры выдающего меня для настройки профиля.

После перехода на страницу «MyProfile» пользователь может посмотреть историю своих записей или перейти к редактированию профиля. Пример страницы «My profile» представлен на рисунке 8.9. При переходе на страницу «Settings» пользователь может задать настройки приложения.

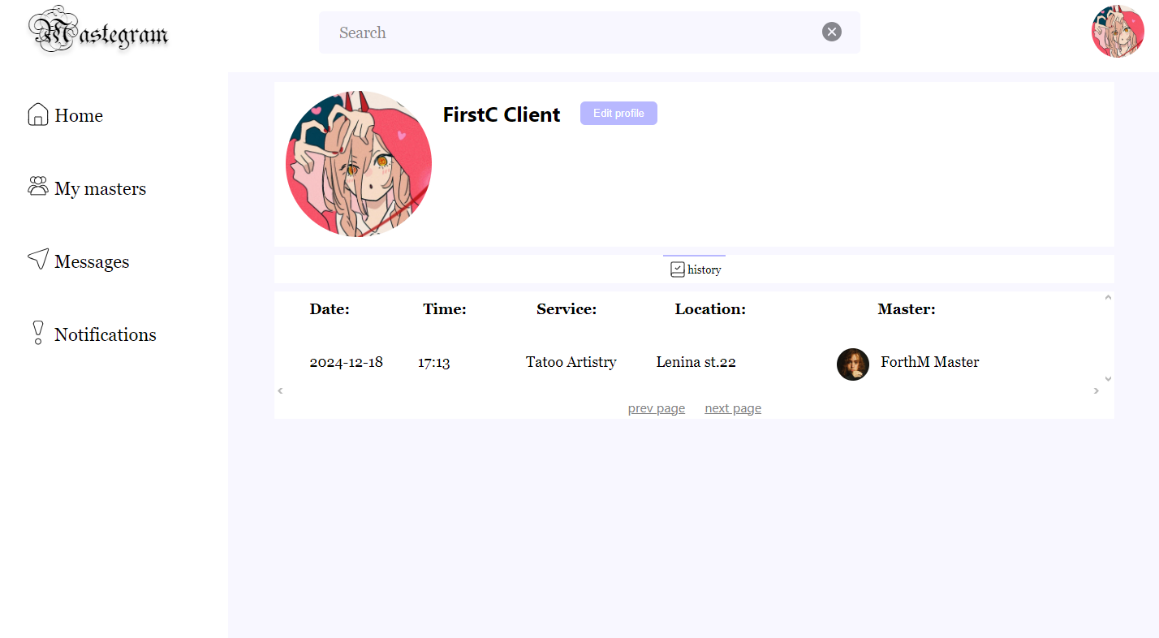


Рисунок 8.9 – Страница профиля пользователя.

Функционал пользователя с ролью «мастер» дополняет расширяет функционал пользователя-клиента, это значит, что пользователь-мастер может выполнять все те же действия что и пользователь-клиент, но также имеет функциональность мастера. Пример главной страницы пользователя-мастера представлен на рисунке 8.10

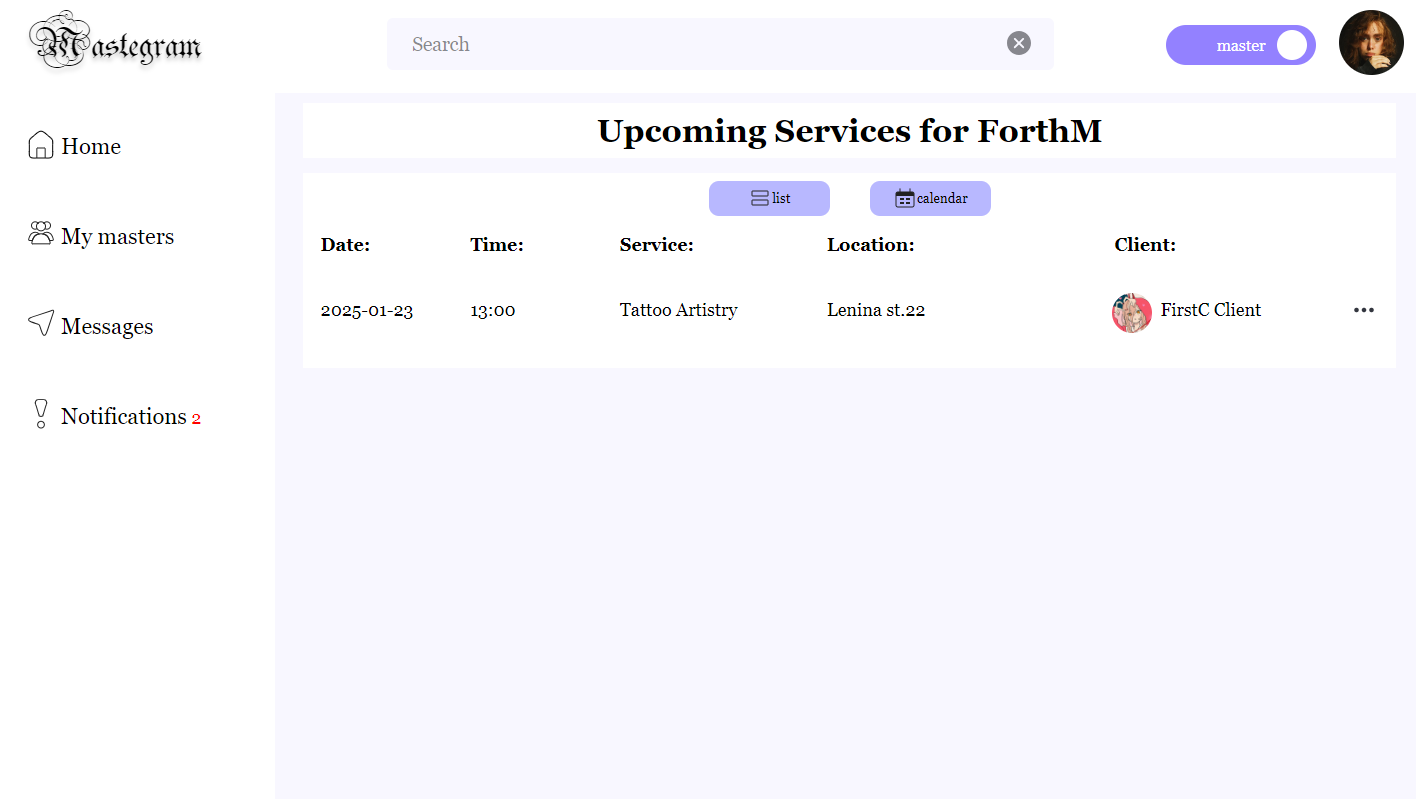


Рисунок 8.10 – Главная страница пользователя-мастера.

На главной странице мастера располагаются предстоящие записи, которые были забронированы клиентами. Мастер может перейти в раздел «Calendar» где он может добавить новые записи на выбранное время. Пример формы для добавления новой записи представлен на рисунке 8.11.

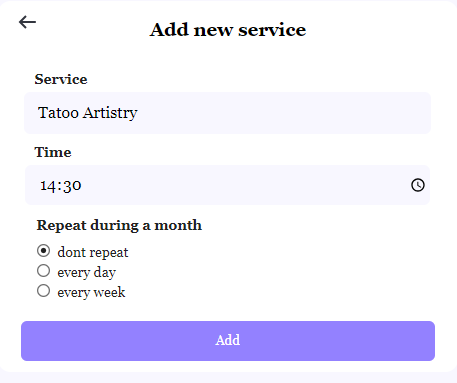


Рисунок 8.10 – Форма для добавления новой записи.

В этом разделе была представлена информация о функционале и возможностях системы, что позволяет пользователям максимально эффективно использовать все ее особенности. В этом разделе можно найти пошаговые инструкции по регистрации, авторизации, созданию и управлению записями, а также получению уведомлений и взаимодействию с мастерами. Руководство призвано помочь пользователям разобраться в приложении, разобраться в процессе бронирования услуг, а также предоставить ответы на часто задаваемые вопросы. Это гарантирует комфортное и беспроблемное использование приложения для всех категорий пользователей.

Заключение

В рамках курсового проекта была разработана система для бронирования услуг с использованием базы Oracle для хранения информации о пользователях, мастерах и записях на услуги. В процессе реализации проекта были тщательно сформулированы требования, спроектирована структура базы данных и написан код.

Система позволяет эффективно управлять данными, поддерживает возможность добавления новых функций без изменения существующих объектов, что соответствует принципам SOLID, в частности, принципу открытости и закрытости (open-closed principle). Например, можно легко добавить новые таблицы и связать их с уже существующими.

Проект был протестирован с точки зрения производительности, и на основе полученных результатов были созданы индексы для повышения эффективности запросов к большому объему данных. Также была реализована возможность импорта и экспорта данных в формате JSON с помощью соответствующих процедур.

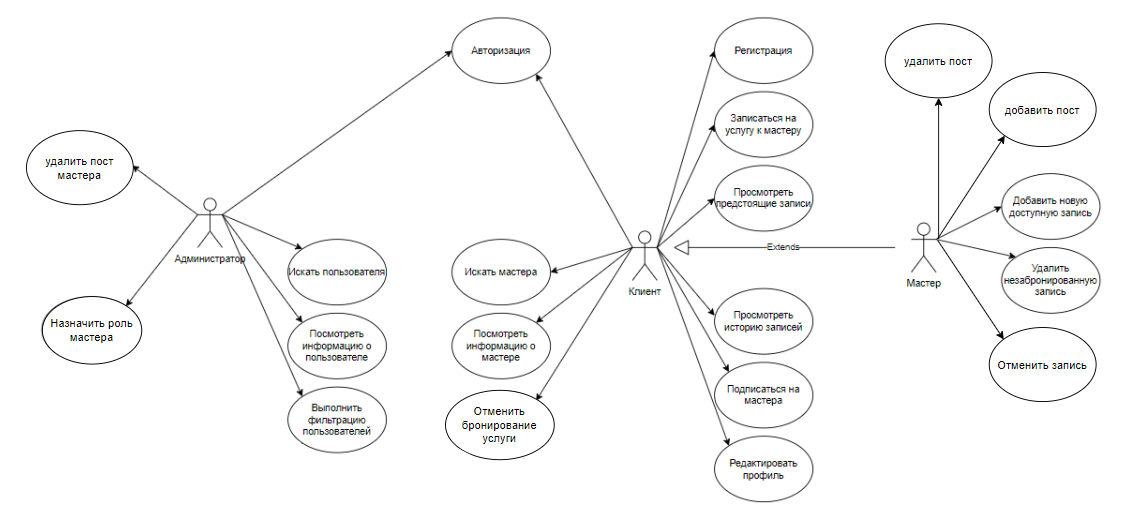
Было также разработано приложение для демонстрации работы базы данных, которое позволяет пользователям взаимодействовать с функционалом системы бронирования услуг. Это приложение реализует весь функционал базы данных, а также обеспечивает интеграцию с системой двухфакторной аутентификации. Приложение продемонстрировало эффективное использование базы данных для хранения и обработки информации, а также показало, как можно интегрировать различные компоненты системы, включая работу с in-memory базой данных (H2) для хранения одноразовых паролей. В приложении также реализована логика шифрования пароля на уровне приложения перед его отправкой в базу данных.

Для удобства пользователей было разработано руководство, которое подробно описывает функциональные возможности приложения и помогает разобраться в процессе работы с системой бронирования.

Список используемых источников

1. Interface форум [Электронный ресурс] / Режим доступа ­– URL: https://www.interface.ru/home.asp?artId=20889– Дата доступа: 14.11.2024
2. Примеры импорта и экспорта JSON-документов (SQL Developer) [Электронный ресурс] / Режим доступа – URL: https://docs.oracle.com/en /database/oracle/oracle-database/21/adjsn/loading-external-json-data.html – Дата доступа: 09.12.2024
3. Руководство пользователя H2 [Электронный ресурс] / Режим доступа – URL: [https://www.h2database.com/html/tutorial.html –](https://www.h2database.com/html/tutorial.html%20–) Дата доступа: 10.12.2024

# Приложение A



# Приложение Б

CREATE TABLE Master\_info (

id NUMBER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,

description VARCHAR2(255) NOT NULL,

business\_address VARCHAR2(255) NOT NULL

) TABLESPACE mastergram\_tablespace;

CREATE TABLE Credential (

id NUMBER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,

username VARCHAR2(255) UNIQUE NOT NULL,

password\_hash VARCHAR2(255) NOT NULL

) TABLESPACE mastergram\_tablespace;

CREATE TABLE Users (

id NUMBER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,

username VARCHAR2(255) NOT NULL,

first\_name VARCHAR2(255) NOT NULL,

second\_name VARCHAR2(255) NOT NULL,

city VARCHAR2(255) NOT NULL,

email VARCHAR2(255) NOT NULL,

tel VARCHAR2(20)NOT NULL,

role VARCHAR2(50) NOT NULL,

profile\_image BLOB,

credential\_id NUMBER NOT NULL,

master\_info\_id NUMBER,

CONSTRAINT fk\_credential FOREIGN KEY (credential\_id) REFERENCES Credential(id),

CONSTRAINT fk\_masterInfo FOREIGN KEY (master\_info\_id) REFERENCES Master\_info(id)

) TABLESPACE mastergram\_tablespace;

CREATE TABLE Post (

id NUMBER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,

creation\_date DATE NOT NULL,

post\_image BLOB NOT NULL,

description VARCHAR2(255),

master\_info\_id NUMBER NOT NULL,

CONSTRAINT fk\_post\_masterinfo FOREIGN KEY (master\_info\_id) REFERENCES Master\_info(id)

) TABLESPACE mastergram\_tablespace;

drop table Booking;

CREATE TABLE Booking (

id NUMBER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,

client\_id NUMBER,

master\_id NUMBER NOT NULL,

service VARCHAR2(255) NOT NULL,

booking\_date DATE NOT NULL,

time TIMESTAMP NOT NULL,

booked NUMBER(1) DEFAULT 0 CHECK (booked IN (0, 1)),

CONSTRAINT fk\_booking\_client FOREIGN KEY (client\_id) REFERENCES Users(id),

CONSTRAINT fk\_booking\_master FOREIGN KEY (master\_id) REFERENCES Master\_info(id)

) TABLESPACE mastergram\_tablespace

partition by range (booking\_date)

interval (numtoyminterval(1, 'month'))

(

partition part0 values less than (to\_date('2024-01-01', 'yyyy-mm-dd')),

partition part1 values less than (to\_date('2024-02-01', 'yyyy-mm-dd'))

);

CREATE TABLE masters\_followers (

client\_id NUMBER NOT NULL,

master\_id NUMBER NOT NULL,

CONSTRAINT pk\_masters\_followers PRIMARY KEY (client\_id, master\_id),

CONSTRAINT fk\_masters\_followers\_client FOREIGN KEY (client\_id) REFERENCES Users(id),

CONSTRAINT fk\_masters\_followers\_master FOREIGN KEY (master\_id) REFERENCES Users(id)

) TABLESPACE mastergram\_tablespace;

CREATE TABLE Notification(

id NUMBER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,

receiver\_id NUMBER NOT NULL,

creation\_date DATE NOT NULL,

message VARCHAR2(255) NOT NULL,

title VARCHAR2(255) NOT NULL,

watched NUMBER(1) DEFAULT 0 CHECK (booked IN (0, 1)),

CONSTRAINT fk\_notification\_user FOREIGN KEY (receiver\_id) REFERENCES Users(id)

) TABLESPACE mastergram\_tablespace;

CREATE TABLE Request(

id NUMBER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,

creation\_date DATE NOT NULL,

user\_id NUMBER NOT NULL,

request\_image BLOB NOT NULL,

cover\_letter VARCHAR2(500) NOT NULL,

CONSTRAINT fk\_request\_user FOREIGN KEY (user\_id) REFERENCES Users(id)

) TABLESPACE mastergram\_tablespace;

CREATE TABLE Setting(

id NUMBER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,

account\_id NUMBER NOT NULL,

enable\_two\_step\_authentication NUMBER(1) DEFAULT 0 CHECK (enable\_two\_step\_authentication IN (0, 1)),

CONSTRAINT fk\_setting\_user FOREIGN KEY (account\_id) REFERENCES Users(id)

) TABLESPACE mastergram\_tablespace;

# Приложение В

-----------------------Notification

CREATE OR REPLACE FUNCTION find\_notifications\_by\_receiver(

p\_receiver\_id IN NUMBER

) RETURN SYS\_REFCURSOR IS

notifications\_cursor SYS\_REFCURSOR;

BEGIN

OPEN notifications\_cursor FOR

SELECT id, receiver\_id, creation\_date, message, title, watched

FROM Notification

WHERE receiver\_id = p\_receiver\_id;

RETURN notifications\_cursor;

EXCEPTION

WHEN OTHERS THEN

RAISE\_APPLICATION\_ERROR(-20001, 'Error trying to find notifications: ' || SQLERRM);

END;

CREATE OR REPLACE PROCEDURE add\_notification\_to\_user (

p\_receiver\_id NUMBER,

p\_creation\_date DATE,

p\_message VARCHAR2,

p\_title VARCHAR2,

p\_watched NUMBER

) AS

BEGIN

INSERT INTO Notification (receiver\_id, creation\_date, message, title, watched)

VALUES (p\_receiver\_id, p\_creation\_date, p\_message, p\_title, p\_watched);

EXCEPTION

WHEN OTHERS THEN

RAISE\_APPLICATION\_ERROR(-20001, 'Error adding notification to user: ' || SQLERRM);

END;

CREATE OR REPLACE PROCEDURE mark\_notifications\_as\_read (

p\_receiver\_id NUMBER

) AS

BEGIN

UPDATE Notification SET watched = 1 WHERE watched = 0 AND receiver\_id = p\_receiver\_id;

EXCEPTION

WHEN OTHERS THEN

RAISE\_APPLICATION\_ERROR(-20001, 'Error marking notifications as read: ' || SQLERRM);

END;

CREATE OR REPLACE FUNCTION count\_unread\_notifications(

p\_receiver\_id IN NUMBER

) RETURN NUMBER IS

v\_count NUMBER;

BEGIN

select count(\*) into v\_count from Notification where receiver\_id = p\_receiver\_id AND watched = 0;

return v\_count;

EXCEPTION

WHEN OTHERS THEN

RAISE\_APPLICATION\_ERROR(-20001, 'Error counting unread notifications: ' || SQLERRM);

END;

----------------------------Post

CREATE OR REPLACE FUNCTION find\_post\_by\_id(

p\_id IN NUMBER

) RETURN SYS\_REFCURSOR IS

post\_cursor SYS\_REFCURSOR;

BEGIN

OPEN post\_cursor FOR

SELECT id, creation\_date, post\_image, description, master\_info\_id

FROM Post

WHERE id = p\_id;

RETURN post\_cursor;

EXCEPTION

WHEN OTHERS THEN

RAISE\_APPLICATION\_ERROR(-20001, 'Error finding post: ' || SQLERRM);

END;

CREATE OR REPLACE PROCEDURE delete\_post (

p\_id NUMBER

) AS

BEGIN

delete Post where id = p\_id;

EXCEPTION

WHEN OTHERS THEN

RAISE\_APPLICATION\_ERROR(-20001, 'Error deleting post: ' || SQLERRM);

END;

CREATE OR REPLACE PROCEDURE add\_post (

p\_creation\_date DATE,

p\_post\_image BLOB,

p\_description VARCHAR2,

p\_master\_info\_id NUMBER

) AS

BEGIN

INSERT INTO Post(creation\_date, post\_image, description, master\_info\_id)

VALUES (p\_creation\_date, p\_post\_image, p\_description, p\_master\_info\_id);

EXCEPTION

WHEN OTHERS THEN

RAISE\_APPLICATION\_ERROR(-20001, 'Error adding post: ' || SQLERRM);

END;

------------------------Request

CREATE OR REPLACE FUNCTION find\_request\_by\_id(

p\_id IN NUMBER

) RETURN SYS\_REFCURSOR IS

request\_cursor SYS\_REFCURSOR;

BEGIN

OPEN request\_cursor FOR

SELECT id, creation\_date, user\_id, request\_image, cover\_letter

FROM Request

WHERE id = p\_id;

RETURN request\_cursor;

EXCEPTION

WHEN OTHERS THEN

RAISE\_APPLICATION\_ERROR(-20001, 'Error finding request: ' || SQLERRM);

END;

CREATE OR REPLACE PROCEDURE add\_request (

p\_creation\_date DATE,

p\_user\_id NUMBER,

p\_request\_image BLOB,

p\_cover\_letter VARCHAR2

) AS

BEGIN

INSERT INTO Request(creation\_date, user\_id, request\_image, cover\_letter)

VALUES (p\_creation\_date, p\_user\_id, p\_request\_image, p\_cover\_letter);

EXCEPTION

WHEN OTHERS THEN

RAISE\_APPLICATION\_ERROR(-20001, 'Error adding request: ' || SQLERRM);

END;

CREATE OR REPLACE PROCEDURE delete\_request (

p\_id NUMBER

) AS

BEGIN

delete Request where id = p\_id;

EXCEPTION

WHEN OTHERS THEN

RAISE\_APPLICATION\_ERROR(-20001, 'Error deleting request: ' || SQLERRM);

END;

CREATE OR REPLACE FUNCTION get\_all\_requests

RETURN SYS\_REFCURSOR IS

request\_cursor SYS\_REFCURSOR;

BEGIN

OPEN request\_cursor FOR

SELECT id, creation\_date, user\_id, request\_image, cover\_letter

FROM Request;

RETURN request\_cursor;

EXCEPTION

WHEN OTHERS THEN

RAISE\_APPLICATION\_ERROR(-20001, 'Error getting all requests: ' || SQLERRM);

END;

CREATE OR REPLACE FUNCTION exists\_request\_by\_user(p\_user\_id IN NUMBER)

RETURN NUMBER IS

v\_count NUMBER;

BEGIN

SELECT COUNT(\*) INTO v\_count

FROM Request

WHERE user\_id = p\_user\_id;

IF v\_count > 0 THEN

RETURN 1;

ELSE

RETURN 0;

END IF;

END;

-------------------Setting

CREATE OR REPLACE FUNCTION find\_setting\_by\_id(

p\_id IN NUMBER

) RETURN SYS\_REFCURSOR IS

setting\_cursor SYS\_REFCURSOR;

BEGIN

OPEN setting\_cursor FOR

SELECT id, account\_id, enable\_two\_step\_authentication

FROM Setting

WHERE id = p\_id;

RETURN setting\_cursor;

EXCEPTION

WHEN OTHERS THEN

RAISE\_APPLICATION\_ERROR(-20001, 'Error finding setting: ' || SQLERRM);

END;

CREATE OR REPLACE PROCEDURE change\_setting(

p\_id IN NUMBER,

p\_enable\_two\_step\_authentication IN NUMBER

) AS

BEGIN

UPDATE Setting set enable\_two\_step\_authentication = p\_enable\_two\_step\_authentication where id = p\_id;

EXCEPTION

WHEN OTHERS THEN

RAISE\_APPLICATION\_ERROR(-20001, 'Error changing setting: ' || SQLERRM);

END;

----------------------------------User

CREATE OR REPLACE FUNCTION find\_user\_by\_id(

p\_id IN NUMBER

) RETURN SYS\_REFCURSOR IS

user\_cursor SYS\_REFCURSOR;

BEGIN

OPEN user\_cursor FOR

SELECT id, username, first\_name, second\_name, city, email, tel, role,

profile\_image, credential\_id, master\_info\_id

FROM Users

WHERE id = p\_id;

RETURN user\_cursor;

EXCEPTION

WHEN OTHERS THEN

RAISE\_APPLICATION\_ERROR(-20001, 'Error finding user: ' || SQLERRM);

END;

CREATE OR REPLACE FUNCTION find\_user\_by\_username(

p\_username IN VARCHAR2

) RETURN SYS\_REFCURSOR IS

user\_cursor SYS\_REFCURSOR;

BEGIN

OPEN user\_cursor FOR

SELECT id, username, first\_name, second\_name, city, email, tel, role,

profile\_image, credential\_id, master\_info\_id

FROM Users

WHERE username = p\_username;

RETURN user\_cursor;

EXCEPTION

WHEN OTHERS THEN

RAISE\_APPLICATION\_ERROR(-20001, 'Error finding user: ' || SQLERRM);

END;

CREATE OR REPLACE FUNCTION exists\_user\_by\_username(p\_username IN VARCHAR2)

RETURN NUMBER IS

v\_count NUMBER;

BEGIN

SELECT COUNT(\*) INTO v\_count

FROM Users

WHERE username = p\_username;

IF v\_count > 0 THEN

RETURN 1;

ELSE

RETURN 0;

END IF;

END;

CREATE OR REPLACE FUNCTION search\_master(

p\_user\_id IN NUMBER,

p\_search IN VARCHAR2

) RETURN SYS\_REFCURSOR IS

master\_cursor SYS\_REFCURSOR;

BEGIN

OPEN master\_cursor FOR

SELECT u.id, u.username, u.first\_name, u.second\_name, u.city, u.email, u.tel, u.role,

u.profile\_image, u.credential\_id, u.master\_info\_id

FROM Users u

JOIN Master\_info mi ON u.master\_info\_id = mi.id

WHERE u.master\_info\_id IS NOT NULL

AND u.id != p\_user\_id

AND (

LOWER(u.first\_name) LIKE '%' || LOWER(p\_search) || '%'

OR LOWER(u.second\_name) LIKE '%' || LOWER(p\_search) || '%'

OR LOWER(u.first\_name) || ' ' || LOWER(u.second\_name) LIKE '%' || LOWER(p\_search) || '%'

OR LOWER(u.second\_name) || ' ' || LOWER(u.first\_name) LIKE '%' || LOWER(p\_search) || '%'

OR LOWER(mi.description) LIKE '%' || LOWER(p\_search) || '%'

);

RETURN master\_cursor;

EXCEPTION

WHEN OTHERS THEN

RAISE\_APPLICATION\_ERROR(-20001, 'Error searching master: ' || SQLERRM);

END;

CREATE OR REPLACE FUNCTION search\_user(

p\_search IN VARCHAR2

) RETURN SYS\_REFCURSOR IS

user\_cursor SYS\_REFCURSOR;

BEGIN

OPEN user\_cursor FOR

SELECT u.id, u.username, u.first\_name, u.second\_name, u.city, u.email, u.tel, u.role,

u.profile\_image, u.credential\_id, u.master\_info\_id

FROM Users u

WHERE (u.role = 'ROLE\_CLIENT' OR u.role = 'ROLE\_MASTER')

AND (

LOWER(u.first\_name) LIKE '%' || LOWER(p\_search) || '%'

OR LOWER(u.second\_name) LIKE '%' || LOWER(p\_search) || '%'

OR LOWER(u.first\_name) || ' ' || LOWER(u.second\_name) LIKE '%' || LOWER(p\_search) || '%'

OR LOWER(u.second\_name) || ' ' || LOWER(u.first\_name) LIKE '%' || LOWER(p\_search) || '%'

);

RETURN user\_cursor;

EXCEPTION

WHEN OTHERS THEN

RAISE\_APPLICATION\_ERROR(-20001, 'Error searching user: ' || SQLERRM);

END;

CREATE OR REPLACE PROCEDURE add\_user (

p\_password\_hash VARCHAR2,

p\_username VARCHAR2,

p\_first\_name VARCHAR2,

p\_second\_name VARCHAR2,

p\_city VARCHAR2,

p\_email VARCHAR2,

p\_tel VARCHAR2,

p\_role VARCHAR2,

p\_profile\_image BLOB,

p\_master\_info\_id NUMBER

) AS

v\_user\_id NUMBER;

v\_credential\_id NUMBER;

BEGIN

INSERT INTO Credential(username, password\_hash) VALUES (p\_username, p\_password\_hash);

SELECT id into v\_credential\_id FROM Credential where username = p\_username;

INSERT INTO Users (username, first\_name, second\_name, city, email, tel, role, profile\_image, credential\_id, master\_info\_id)

VALUES (p\_username, p\_first\_name, p\_second\_name, p\_city, p\_email, p\_tel, p\_role, p\_profile\_image, v\_credential\_id, p\_master\_info\_id);

SELECT id into v\_user\_id FROM Users where username = p\_username;

INSERT INTO Setting(account\_id, enable\_two\_step\_authentication) VALUES (v\_user\_id, 0);

EXCEPTION

WHEN OTHERS THEN

RAISE\_APPLICATION\_ERROR(-20001, 'Error adding new user: ' || SQLERRM);

END;

CREATE OR REPLACE PROCEDURE update\_user\_client (

p\_id NUMBER,

p\_username VARCHAR2,

p\_first\_name VARCHAR2,

p\_second\_name VARCHAR2,

p\_city VARCHAR2,

p\_email VARCHAR2,

p\_tel VARCHAR2,

p\_profile\_image BLOB

) AS

BEGIN

UPDATE USERS set username = p\_username, first\_name = p\_first\_name, second\_name = p\_second\_name,

city = p\_city, email = p\_email, tel = p\_tel, profile\_image = p\_profile\_image

WHERE id = p\_id;

update Credential set username = p\_username where id = (select credential\_id from Users where id = p\_id);

EXCEPTION

WHEN OTHERS THEN

RAISE\_APPLICATION\_ERROR(-20001, 'Error updating client info: ' || SQLERRM);

END;

CREATE OR REPLACE PROCEDURE update\_user\_master (

p\_id NUMBER,

p\_username VARCHAR2,

p\_first\_name VARCHAR2,

p\_second\_name VARCHAR2,

p\_city VARCHAR2,

p\_email VARCHAR2,

p\_tel VARCHAR2,

p\_profile\_image BLOB,

p\_description VARCHAR2,

p\_business\_address VARCHAR2

) AS

v\_master\_info\_id NUMBER;

BEGIN

SELECT master\_info\_id into v\_master\_info\_id from Users where id = p\_id;

IF v\_master\_info\_id IS NULL THEN

insert into Master\_info(description, business\_address) VALUES(p\_description, p\_business\_address)

returning id into v\_master\_info\_id;

UPDATE USERS set username = p\_username, first\_name = p\_first\_name, second\_name = p\_second\_name,

city = p\_city, email = p\_email, tel = p\_tel, profile\_image = p\_profile\_image, master\_info\_id = v\_master\_info\_id

WHERE id = p\_id;

ELSE

update Master\_info set description = p\_description, business\_address = p\_business\_address

where id = v\_master\_info\_id;

UPDATE USERS set username = p\_username, first\_name = p\_first\_name, second\_name = p\_second\_name,

city = p\_city, email = p\_email, tel = p\_tel, profile\_image = p\_profile\_image

WHERE id = p\_id;

END IF;

update Credential set username = p\_username where id = (select credential\_id from Users where id = p\_id);

EXCEPTION

WHEN OTHERS THEN

RAISE\_APPLICATION\_ERROR(-20001, 'Error during updating master info: ' || SQLERRM);

END;

CREATE OR REPLACE PROCEDURE make\_master(p\_user\_id IN NUMBER)

AS

BEGIN

update Users set role = 'ROLE\_MASTER' where id = p\_user\_id;

EXCEPTION

WHEN OTHERS THEN

RAISE\_APPLICATION\_ERROR(-20001, 'Error making master: ' || SQLERRM);

END;

CREATE OR REPLACE PROCEDURE follow\_master (

p\_client\_id NUMBER,

p\_master\_id NUMBER

) AS

BEGIN

insert into masters\_followers(client\_id, master\_id) values(p\_client\_id, p\_master\_id);

EXCEPTION

WHEN OTHERS THEN

RAISE\_APPLICATION\_ERROR(-20001, 'Error following master: ' || SQLERRM);

END;

CREATE OR REPLACE PROCEDURE unfollow\_master (

p\_client\_id NUMBER,

p\_master\_id NUMBER

) AS

BEGIN

delete masters\_followers where client\_id = p\_client\_id AND master\_id = p\_master\_id;

EXCEPTION

WHEN OTHERS THEN

RAISE\_APPLICATION\_ERROR(-20001, 'Error unfollowing master: ' || SQLERRM);

END;

CREATE OR REPLACE FUNCTION is\_client\_following\_master(

p\_client\_id NUMBER,

p\_master\_id NUMBER

)

RETURN NUMBER IS

v\_res NUMBER;

BEGIN

SELECT COUNT(\*) INTO v\_res

FROM masters\_followers

WHERE client\_id = p\_client\_id AND master\_id = p\_master\_id;

IF v\_res > 0 THEN

RETURN 1;

ELSE

RETURN 0;

END IF;

END;

-----------------------Master\_info

CREATE OR REPLACE FUNCTION find\_master\_info\_by\_id(

p\_id IN NUMBER

) RETURN SYS\_REFCURSOR IS

master\_info\_cursor SYS\_REFCURSOR;

BEGIN

OPEN master\_info\_cursor FOR

SELECT id, description, business\_address

FROM Master\_info

WHERE id = p\_id;

RETURN master\_info\_cursor;

EXCEPTION

WHEN OTHERS THEN

RAISE\_APPLICATION\_ERROR(-20001, 'Error finding master info: ' || SQLERRM);

END;

------------------------------------Booking

CREATE OR REPLACE FUNCTION find\_booking\_by\_id(

p\_id IN NUMBER

) RETURN SYS\_REFCURSOR IS

booking\_cursor SYS\_REFCURSOR;

BEGIN

OPEN booking\_cursor FOR

SELECT id, client\_id, master\_id, service, booking\_date, time, booked

FROM Booking

WHERE id = p\_id;

RETURN booking\_cursor;

EXCEPTION

WHEN OTHERS THEN

RAISE\_APPLICATION\_ERROR(-20001, 'Error finding booking: ' || SQLERRM);

END;

CREATE OR REPLACE FUNCTION find\_bookings\_in\_range(

p\_master\_id IN NUMBER,

p\_start\_date DATE,

p\_end\_date DATE,

p\_now\_date DATE,

p\_now\_time TIMESTAMP

) RETURN SYS\_REFCURSOR IS

booking\_cursor SYS\_REFCURSOR;

BEGIN

IF (p\_start\_date = p\_now\_date) THEN

OPEN booking\_cursor FOR

SELECT id, client\_id, master\_id, service, booking\_date, time, booked

FROM Booking

WHERE master\_id = p\_master\_id

AND booking\_date BETWEEN p\_start\_date AND p\_end\_date

AND booking\_date = p\_start\_date

AND time > p\_now\_time;

ELSE

OPEN booking\_cursor FOR

SELECT id, client\_id, master\_id, service, booking\_date, time, booked

FROM Booking

WHERE master\_id = p\_master\_id

AND booking\_date BETWEEN p\_start\_date AND p\_end\_date

AND booking\_date > p\_start\_date;

END IF;

RETURN booking\_cursor;

EXCEPTION

WHEN OTHERS THEN

RAISE\_APPLICATION\_ERROR(-20001, 'Error finding bookings in range: ' || SQLERRM);

END;

CREATE OR REPLACE PROCEDURE add\_booking (

p\_master\_id NUMBER,

p\_service VARCHAR2,

p\_booking\_date DATE,

p\_time TIMESTAMP

) AS

BEGIN

INSERT INTO Booking(master\_id, service, booking\_date, time)

VALUES (p\_master\_id, p\_service, p\_booking\_date, p\_time);

EXCEPTION

WHEN OTHERS THEN

RAISE\_APPLICATION\_ERROR(-20001, 'Error adding new booking: ' || SQLERRM);

END;

CREATE OR REPLACE PROCEDURE update\_booking (

p\_id NUMBER,

p\_client\_id NUMBER,

p\_service VARCHAR2,

p\_booking\_date DATE,

p\_time TIMESTAMP,

p\_booked NUMBER

) AS

BEGIN

UPDATE Booking set client\_id = p\_client\_id, service = p\_service, booking\_date = p\_booking\_date,

time = p\_time, booked = p\_booked

WHERE id = p\_id;

EXCEPTION

WHEN OTHERS THEN

RAISE\_APPLICATION\_ERROR(-20001, 'Error updating booking: ' || SQLERRM);

END;

CREATE OR REPLACE FUNCTION find\_free\_bookings\_in\_range(

p\_master\_id IN NUMBER,

p\_start\_date DATE,

p\_end\_date DATE,

p\_now\_date DATE,

p\_now\_time TIMESTAMP

) RETURN SYS\_REFCURSOR IS

booking\_cursor SYS\_REFCURSOR;

BEGIN

IF (p\_start\_date = p\_now\_date) THEN

OPEN booking\_cursor FOR

SELECT id, client\_id, master\_id, service, booking\_date, time, booked

FROM Booking

WHERE master\_id = p\_master\_id

AND booked = 0

AND booking\_date BETWEEN p\_start\_date AND p\_end\_date

AND booking\_date = p\_start\_date

AND time > p\_now\_time;

ELSE

OPEN booking\_cursor FOR

SELECT id, client\_id, master\_id, service, booking\_date, time, booked

FROM Booking

WHERE master\_id = p\_master\_id

AND booked = 0

AND booking\_date BETWEEN p\_start\_date AND p\_end\_date

AND booking\_date = p\_start\_date;

END IF;

RETURN booking\_cursor;

EXCEPTION

WHEN OTHERS THEN

RAISE\_APPLICATION\_ERROR(-20001, 'Error finding free bookings in range: ' || SQLERRM);

END;

CREATE OR REPLACE FUNCTION find\_bookings\_by\_date(

p\_master\_id IN NUMBER,

p\_date DATE,

p\_now\_date DATE,

p\_now\_time TIMESTAMP

) RETURN SYS\_REFCURSOR IS

booking\_cursor SYS\_REFCURSOR;

BEGIN

OPEN booking\_cursor FOR

SELECT id, client\_id, master\_id, service, booking\_date, time, booked

FROM Booking

WHERE master\_id = p\_master\_id

AND booking\_date = p\_date

AND (p\_date != p\_now\_date OR time > p\_now\_time);

RETURN booking\_cursor;

EXCEPTION

WHEN OTHERS THEN

RAISE\_APPLICATION\_ERROR(-20001, 'Error finding bookings by date: ' || SQLERRM);

END;

CREATE OR REPLACE FUNCTION find\_free\_bookings\_by\_date(

p\_master\_id IN NUMBER,

p\_date DATE,

p\_now\_date DATE,

p\_now\_time TIMESTAMP

) RETURN SYS\_REFCURSOR IS

booking\_cursor SYS\_REFCURSOR;

BEGIN

OPEN booking\_cursor FOR

SELECT id, client\_id, master\_id, service, booking\_date, time, booked

FROM Booking

WHERE master\_id = p\_master\_id

AND booked = 0

AND booking\_date = p\_date

AND (p\_date != p\_now\_date OR time > p\_now\_time);

RETURN booking\_cursor;

EXCEPTION

WHEN OTHERS THEN

RAISE\_APPLICATION\_ERROR(-20001, 'Error finding free bookings by date: ' || SQLERRM);

END;

CREATE OR REPLACE FUNCTION find\_bookings\_by\_client(

p\_client\_id IN NUMBER,

p\_now\_date DATE,

p\_now\_time TIMESTAMP

) RETURN SYS\_REFCURSOR IS

booking\_cursor SYS\_REFCURSOR;

BEGIN

OPEN booking\_cursor FOR

SELECT id, client\_id, master\_id, service, booking\_date, time, booked

FROM Booking

WHERE client\_id = p\_client\_id

AND (booking\_date > p\_now\_date

OR (booking\_date = p\_now\_date AND time > p\_now\_time));

RETURN booking\_cursor;

EXCEPTION

WHEN OTHERS THEN

RAISE\_APPLICATION\_ERROR(-20001, 'Error finding bookings by client: ' || SQLERRM);

END;

CREATE OR REPLACE FUNCTION find\_booked\_bookings\_by\_master(

p\_master\_id IN NUMBER,

p\_now\_date DATE,

p\_now\_time TIMESTAMP

) RETURN SYS\_REFCURSOR IS

booking\_cursor SYS\_REFCURSOR;

BEGIN

OPEN booking\_cursor FOR

SELECT id, client\_id, master\_id, service, booking\_date, time, booked

FROM Booking

WHERE master\_id = p\_master\_id

AND booked = 1

AND (booking\_date > p\_now\_date

OR (booking\_date = p\_now\_date AND time > p\_now\_time));

RETURN booking\_cursor;

EXCEPTION

WHEN OTHERS THEN

RAISE\_APPLICATION\_ERROR(-20001, 'Error finding booked bookings by master: ' || SQLERRM);

END;

CREATE OR REPLACE FUNCTION find\_history\_bookings\_by\_client(

p\_client\_id IN NUMBER,

p\_now\_date DATE,

p\_now\_time TIMESTAMP

) RETURN SYS\_REFCURSOR IS

booking\_cursor SYS\_REFCURSOR;

BEGIN

OPEN booking\_cursor FOR

SELECT id, client\_id, master\_id, service, booking\_date, time, booked

FROM Booking

WHERE client\_id = p\_client\_id

AND (booking\_date < p\_now\_date

OR (booking\_date = p\_now\_date AND time < p\_now\_time));

RETURN booking\_cursor;

EXCEPTION

WHEN OTHERS THEN

RAISE\_APPLICATION\_ERROR(-20001, 'Error finding history bookings by client: ' || SQLERRM);

END;

CREATE OR REPLACE FUNCTION find\_history\_bookings\_by\_master(

p\_master\_id IN NUMBER,

p\_now\_date DATE,

p\_now\_time TIMESTAMP

) RETURN SYS\_REFCURSOR IS

booking\_cursor SYS\_REFCURSOR;

BEGIN

OPEN booking\_cursor FOR

SELECT id, client\_id, master\_id, service, booking\_date, time, booked

FROM Booking

WHERE master\_id = p\_master\_id

AND booked = 1

AND (booking\_date < p\_now\_date

OR (booking\_date = p\_now\_date AND time < p\_now\_time));

RETURN booking\_cursor;

EXCEPTION

WHEN OTHERS THEN

RAISE\_APPLICATION\_ERROR(-20001, 'Error finding history bookings by master: ' || SQLERRM);

END;

CREATE OR REPLACE PROCEDURE delete\_booking (

p\_id NUMBER

) AS

BEGIN

DELETE Booking WHERE id = p\_id;

EXCEPTION

WHEN OTHERS THEN

RAISE\_APPLICATION\_ERROR(-20001, 'Error deleting booking: ' || SQLERRM);

END;

CREATE OR REPLACE PROCEDURE delete\_expired\_bookings

AS

BEGIN

DELETE Booking WHERE booked = 0 and booking\_date < sysdate;

EXCEPTION

WHEN OTHERS THEN

RAISE\_APPLICATION\_ERROR(-20001, 'Error deleting expired bookings: ' || SQLERRM);

END;

# Приложение Д

CREATE OR REPLACE PROCEDURE export\_bookings(file\_name IN VARCHAR2) AS

file\_handle UTL\_FILE.FILE\_TYPE;

BEGIN

file\_handle := UTL\_FILE.FOPEN('JSON\_DIR', file\_name, 'W', 500);

FOR record IN (

SELECT id, client\_id, master\_id, service, booking\_date, time, booked

FROM Booking

) LOOP

DECLARE

json\_record CLOB;

BEGIN

SELECT JSON\_OBJECT(

'id' VALUE record.id,

'client\_id' VALUE record.client\_id,

'master\_id' VALUE record.master\_id,

'service' VALUE record.service,

'booking\_date' VALUE record.booking\_date,

'time' VALUE record.time,

'booked' VALUE record.booked

)

INTO json\_record

FROM DUAL;

UTL\_FILE.PUT\_LINE(file\_handle, json\_record);

END;

END LOOP;

UTL\_FILE.FCLOSE(file\_handle);

EXCEPTION

WHEN OTHERS THEN

UTL\_FILE.FCLOSE\_ALL;

RAISE\_APPLICATION\_ERROR(-20001, 'Error exporting data: ' || SQLERRM);

END;

CREATE OR REPLACE PROCEDURE import\_bookings(file\_name IN VARCHAR2) AS

file\_handle UTL\_FILE.FILE\_TYPE;

v\_id NUMBER;

v\_client\_id NUMBER;

v\_master\_id NUMBER;

v\_service VARCHAR2(255);

v\_booking\_date DATE;

v\_time TIMESTAMP;

v\_booked NUMBER(1);

BEGIN

file\_handle := UTL\_FILE.FOPEN('JSON\_DIR', file\_name, 'R', 500);

WHILE TRUE LOOP

DECLARE

line VARCHAR2(500);

BEGIN

UTL\_FILE.GET\_LINE(file\_handle, line);

SELECT id, client\_id, master\_id, service, booking\_date, time, booked

INTO v\_id, v\_client\_id, v\_master\_id, v\_service, v\_booking\_date, v\_time, v\_booked

FROM JSON\_TABLE(

line,

'$' COLUMNS (

id NUMBER PATH '$.id',

client\_id NUMBER PATH '$.client\_id',

master\_id NUMBER PATH '$.master\_id',

service VARCHAR2(255) PATH '$.service',

booking\_date DATE PATH '$.booking\_date',

time TIMESTAMP PATH '$.time',

booked NUMBER(1) PATH '$.booked'

)

);

IF v\_id IS NULL THEN

INSERT INTO Booking (client\_id, master\_id, service, booking\_date, time, booked)

VALUES (v\_client\_id, v\_master\_id, v\_service, v\_booking\_date, v\_time, v\_booked);

ELSE

INSERT INTO Booking (id, client\_id, master\_id, service, booking\_date, time, booked)

VALUES (v\_id, v\_client\_id, v\_master\_id, v\_service, v\_booking\_date, v\_time, v\_booked);

END IF;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

EXIT; -- Выходим из цикла, если достигнут конец файла

WHEN OTHERS THEN

RAISE\_APPLICATION\_ERROR(-20001, 'Error importing data: ' || SQLERRM);

END;

END LOOP;

UTL\_FILE.FCLOSE(file\_handle);

COMMIT;

EXCEPTION

WHEN OTHERS THEN

UTL\_FILE.FCLOSE\_ALL;

RAISE;

END;

# Приложение Е

CREATE OR REPLACE PROCEDURE fill\_booking\_table\_for\_one\_client(

p\_client\_id IN NUMBER,

p\_master\_id IN NUMBER

) AS

v\_start\_date DATE := TRUNC(SYSDATE - 1);

v\_end\_date DATE := ADD\_MONTHS(v\_start\_date, -12);

v\_current\_date DATE;

v\_current\_time TIMESTAMP;

BEGIN

v\_current\_date := v\_start\_date;

WHILE v\_current\_date > v\_end\_date LOOP

FOR hour IN 0..23 LOOP

v\_current\_time := TO\_TIMESTAMP(v\_current\_date, 'YYYY-MM-DD')

+ (hour / 24)

+ (TRUNC(DBMS\_RANDOM.VALUE(0, 60)) / 1440);

INSERT INTO Booking (

client\_id, master\_id, service, booking\_date, time, booked

) VALUES (

p\_client\_id, p\_master\_id, 'filling service', v\_current\_date, v\_current\_time, 1

);

END LOOP;

v\_current\_date := v\_current\_date - 1;

END LOOP;

COMMIT;

EXCEPTION

WHEN OTHERS THEN

dbms\_output.put\_line('error filling booking table for client ' || sqlerrm);

END;

CREATE OR REPLACE PROCEDURE fill\_booking\_table IS

BEGIN

fill\_booking\_table\_for\_one\_client(7, 1);

fill\_booking\_table\_for\_one\_client(8, 2);

fill\_booking\_table\_for\_one\_client(9, 3);

fill\_booking\_table\_for\_one\_client(10, 4);

fill\_booking\_table\_for\_one\_client(21, 5);

fill\_booking\_table\_for\_one\_client(22, 1);

fill\_booking\_table\_for\_one\_client(23, 2);

fill\_booking\_table\_for\_one\_client(24, 3);

fill\_booking\_table\_for\_one\_client(25, 4);

fill\_booking\_table\_for\_one\_client(26, 5);

fill\_booking\_table\_for\_one\_client(27, 1);

fill\_booking\_table\_for\_one\_client(28, 2);

END;