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 для программного средства «Task Wave» с использованием технологии шифрования и маскирования данных»

Исполнитель

Студент(ка) 2 курса группы 5 Кравченко Алексей Дмитриевич

(Ф.И.О.)

Руководитель работы ассист. Нистюк Ольга Александровна

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

Курсовой проект защищен с оценкой

Председатель доц. Блинова Е.А.

(подпись)

Минск 2023

МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования   
«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет информационных технологий   
Кафедра информационных систем и технологий

Утверждаю

Заведующий кафедрой

\_\_\_\_\_\_\_\_\_\_\_\_ Е.А. Блинова

подпись инициалы и фамилия

«\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2023 г.

**ЗАДАНИЕ**

**к курсовой работе**

**по дисциплине** «Базы данных»

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

Курс: 3 Группа: 5

Студент: Кравченко Алексей Дмитриевич

**Тема: «**Реализация базы данных для web-приложения планирования задач и проектов с применением технологии шифрования и маскирования данных»

**1. Срок сдачи студентом законченной работы**: «11» декабря 2023 г.

**2. Исходные данные к курсовой работе:**

**2.1**. Функционально должны быть выполнены следующие задачи:

* определение ролей (администратор, стандартный пользователь, пользователь с расширенными правами);
* управление пользователями (регистрация, авторизация);
* управление проектами и задачами (создание, редактирование, удаление, выполнение);
* просмотр статистики (количество выполненных задач и проектов);
* настройка личного аккаунта (включая установку фотографии, выбор имени пользователя в Telegram, указание электронной почты, названия компании и города проживания, а также возможность изменения имени и логина).

**2.2. Требования.**

* База данных должна быть реализована в СУБД Oracle.
* Доступ к данным должен осуществляться только через соответствующие процедуры.
* Количество объектов БД (таблиц, представлений, индексов, пользователей и пр.) регламентируется задачей.
* Должен быть проведен импорт данных из JSON файлов, экспорт данных в формат JSON.
* Необходимо протестировать производительность базы данных на таблице, содержащей не менее 100 000 строк, и внести изменения в структуру в случае необходимости. Необходимо проанализировать планы запросов к таблице.
* Применить технологию базы данных согласно выбранной теме: подробно описать применяемые системные пакеты, утилиты или технологии; показать применение указанной технологии в базе данных.
* Листинги проекта должны содержать комментарии.

**3. Содержание расчетно-пояснительной записки**

* Введение
* Постановка задачи
* Проектирование базы данных.
* Разработка объектов базы данных
* Описание процедур импорта и экспорта
* Тестирование производительности
* Описание технологии и ее применения в базе данных
* Краткое описание приложения для демонстрации
* Руководство пользователя
* Заключение
* Список используемых источников
* Приложения

**4. Форма представления, выполненного курсовой работы:**

* Пояснительная записка оформляется в MS Word.
* Оформление пояснительной записки должно соответствовать требованиям к оформлению пояснительной записки для курсовых работ.
* Листинги всех скриптов представляются в приложении.
* К записке необходимо приложить DVD-диск, который должен содержать пояснительную записку, листинги и файлы базы данных.

#### Календарный план

|  |  |  |  |
| --- | --- | --- | --- |
| №  п/п | Наименование этапов курсовой работы | Срок выполнения этапов работы | Примечание |
| 1 | Введение | 19.09.2023 |  |
| 2 | Аналитический обзор литературы по теме работы | 26.09.2023 |  |
| 3 | Изучение требований, определение вариантов использования | 03.10.2023 |  |
| 4 | Анализ и проектирование модели базы данных. Описание информационных объектов и ограничений целостности. | 10.10.2023 |  |
| 5 | Создание необходимых объектов | 24.10.2023 |  |
| 6 | Импорт и экспорт данных | 07.11.2023 |  |
| 7 | Описание используемой технологии | 14.11.2023 |  |
| 8 | Тестирование производительности | 21.11.2023 |  |
| 9 | Оформление пояснительной записки | 05.12.2023 |  |
| 10 | Сдача курсовой работы | 11.12.2023 |  |

**5. Дата выдачи задания** «07» сентября 2023 г.

Руководитель \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ О.А. Нистюк

(подпись)

Задание принял к исполнению \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(дата и подпись студента)

**Содержание**

[ВВЕДЕНИЕ 5](#_Toc153134759)

[1 Поставновка задачи 6](#_Toc153134760)

[1.1 Обзор аналогичных решений 6](#_Toc153134761)

[1.2 Требования к проектируемому программному средству 9](#_Toc153134762)

[2 Проектирование базы данных 11](#_Toc153134763)

[3 Разработка объектов базы данных 15](#_Toc153134764)

[3.1 Пользователи 15](#_Toc153134765)

[3.2 Функции 16](#_Toc153134766)

[3.3 Хранимые процедуры 17](#_Toc153134767)

[3.4 Индексы 18](#_Toc153134768)

[4 Описание процедур импорта и экспорта 19](#_Toc153134769)

[5 Тестирования производительности 22](#_Toc153134770)

[6 Описание технологии и её применение в базе данных 24](#_Toc153134771)

[6.1 Шифрование данных 24](#_Toc153134772)

[6.2 Маскирование данных 25](#_Toc153134773)

[6.3 Хранение мультимедийных типов данных 27](#_Toc153134774)

[7 Краткое описание приложения для демонстрации 28](#_Toc153134775)

[8 Руководство пользователя 30](#_Toc153134776)

[ЗАКЛЮЧЕНИЕ 35](#_Toc153134777)

[Список использованной литературы 36](#_Toc153134778)

[Приложение A 37](#_Toc153134779)

[Приложение Б 38](#_Toc153134780)

[Приложение В 41](#_Toc153134781)

[Приложение Г 44](#_Toc153134782)

[Приложение Д 56](#_Toc153134783)

# ВВЕДЕНИЕ

Тема курсового проекта «Программа для организации задач и управления проектами» имеет огромную актуальность и практическую значимость в современном мире, где эффективное управление задачами и проектами является неотъемлемой частью успешного функционирования организаций и достижения их целей.

Анализ достижений в области управления задачами и проектами показывает, что существующие методы и инструменты имеют свои преимущества, однако с ростом сложности и масштаба проектов возникают новые вызовы, требующие более эффективных и интегрированных решений. Быстрая смена требований, распределенные команды, необходимость в управлении ресурсами и сроками, а также взаимодействие между участниками проекта – всё это создает потребность в разработке специализированной программы для организации задач и управления проектами.

Целью данного курсового проектирования является разработка и реализация реляционной базы данных. Эта база данных должна обеспечивать возможность создания, изменения и удаления проектов, пользователей, добавления различных мультимедийных данных, организацию безопасности и администрирование.

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 — база данных, основанная на реляционной модели данных или, как их ещё называют, двумерной таблице. В качестве СУБД для базы данных был выбран Oracle - это одна из самых популярных систем управления базами данных, широко применяемая в различных областях, включая банковское дело, медицину, телекоммуникации и др. Её мощные функциональные возможности и высокая производительность делают её идеальным выбором для обработки и хранения огромного объема данных. Благодаря реляционной модели данных, в основе которой лежит концепция таблиц с отношениями между ними, Oracle позволяет эффективно организовывать и обрабатывать информацию, обеспечивая быструю и точную выдачу результатов поиска и запросов.

В основной части будут затронуты все аспекты разработки проекта и обоснованы некоторые технические приёмы, к которым приходилось прибегнуть, с целью реализации работы веб-сервера с базой данных.

В итоге, разработка данной программы позволит организациям и их сотрудникам эффективно управлять задачами и проектами, повышая производительность и достижение целей организации.

# Поставновка задачи

Целью данной работы является проектирование базы данных для программного средства «TaskWave» с технологией шифрования и маскирования данных СУБД. В качестве модели данных следует использовать реляционную модель. Проектирование необходимо произвести таким образом, чтобы конечные данные соответствовали общим требованиям к информации в базе данных.

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 — база данных, основанная на реляционной модели данных. В качестве базы данных был выбран Oracle, в связи с его высокой производительностью и надежностью.

В основной части будут затронуты все аспекты разработки проекта и обоснованы некоторые технические приёмы, к которым приходилось прибегнуть, с целью реализации работы веб-приложения с базой данных.

Основные требования к приложению:

* Возможность регистрации и авторизации пользователей (с разными ролями);
* Создание одиночных проектов;
* Создание групповых проектов;
* Создание набора задач в рамках каждого проекта;
* Просмотр списка актуальных/отправленных/выполненных задач;
* Просмотр статистики;
* Настройка личного аккаунта (добавление информации, установка фотографии);
* Отправка на проверку выполненных проектов;
* Выполнение проектов;
* Прикрепление к проектам фотографий и файлов;

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. На сегодняшний день можно встретить достаточно большое количество программных решений со схожей тематикой. По этой причине, мы рассмотрим ряд таких программных средств, которые помогут определиться с функционалом, который будет представлен и в приложении «Task Wave».

Первый аналог – **«Trello»**. Trello - это программное средство для организации задач и управления проектами, которое предоставляет удобные и интуитивно понятные инструменты для визуализации и отслеживания рабочих процессов. Оно позволяет пользователям создавать доски, списки и карточки, которые могут представлять собой задачи, проекты, идеи или любую другую единицу работы, которую необходимо выполнить.

На рисунке 1.1 изображён интерфейс приложения:

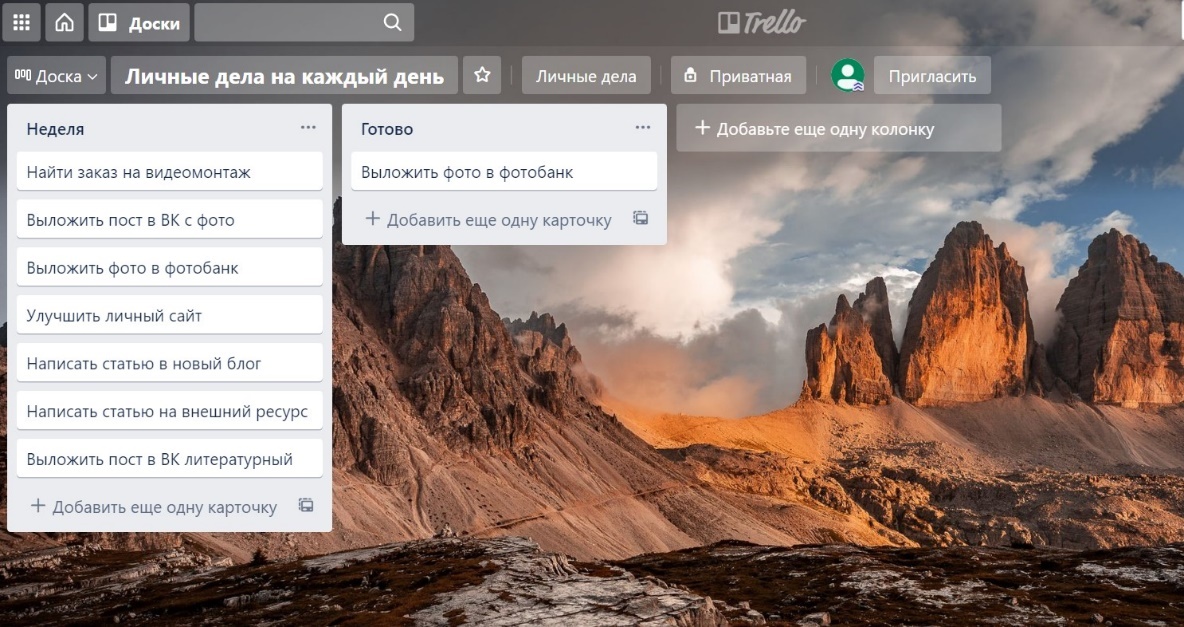


Рисунок 1.1 – Интерфейс приложения «Trello»

Основной принцип Trello — это применение методологии Канбан, которая базируется на принципе визуального управления задачами. Каждая доска в Trello представляет определенный проект или рабочий поток, а списки на доске отображают этапы выполнения задачи. Карточки на списке представляют собой отдельные задачи или элементы работы, которые можно перемещать между списками для отслеживания прогресса.

Программное средство Trello предлагает несколько ключевых функциональных возможностей. Во-первых, пользователи могут создавать доски и настраивать их в соответствии с конкретными потребностями проекта или задачи. Во-вторых, они могут создавать списки на доске и добавлять карточки, которые содержат информацию о задаче, описание, сроки выполнения и другие детали.

Кроме того, Trello предоставляет возможность назначать ответственных лиц на задачи, устанавливать приоритеты, добавлять комментарии и вложения, а также устанавливать оповещения и сроки выполнения. Пользователи могут легко перемещать карточки между списками для обозначения текущего состояния задачи (например, «В ожидании», «В процессе», «Завершено»).

Trello также предоставляет возможность сотрудничества и коммуникации между участниками команды. Пользователи могут добавлять других пользователей на доски и карточки, делиться комментариями, обсуждать задачи и отслеживать активности. Это способствует более эффективному взаимодействию и синхронизации работы внутри команды.

**Преимущества:** интуитивно-понятный интерфейс, бесплатная версия, гибкая система досок и карточек.

**Недостатки:** ограниченная функциональность в бесплатной версии, отсутствие расширенных инструментов для аналитики.

**Вывод: «**Trello» отлично подходит для небольших и средних проектов, где простота использования является ключевым фактором. Он предоставляет базовые инструменты для управления задачами, но может не быть оптимальным решением для крупных и сложных проектов.

В качестве второго аналога рассмотрим **«YouGile»** Это программное средство для организации задач и управления проектами, которое предоставляет всесторонние инструменты для эффективного планирования, отслеживания и совместной работы над проектами. Оно разработано с учетом потребностей современных команд и предлагает интуитивно понятный интерфейс и широкий набор функциональных возможностей. Интерфейс приложения представлен на рисунке 1.2:



Рисунок 1.2 – Интерфейс программного средства «YouGile»

Одной из ключевых особенностей YouGile является его модульность и гибкость. Пользователи могут создавать проекты и определять собственную структуру задач, включая списки, подзадачи, теги и метки. Это позволяет адаптировать программное средство под конкретные потребности проекта или команды.

YouGile предлагает различные инструменты для планирования и отслеживания задач. Пользователи могут создавать задачи, устанавливать сроки выполнения, назначать ответственных лиц, устанавливать приоритеты и отмечать прогресс. Графическое представление досок и карточек позволяет легко визуализировать текущее состояние проекта и прогресс работы.

Кроме того, YouGile предоставляет возможности для коммуникации и совместной работы. Пользователи могут оставлять комментарии, обсуждать задачи и обмениваться файлами прямо в рамках программного средства. Также предусмотрены функции оповещений, которые помогают участникам команды быть в курсе изменений и обновлений проекта.

YouGile поддерживает гибкую систему управления доступом, позволяющую определить права и роли участников проекта. Это обеспечивает конфиденциальность и безопасность данных, а также позволяет гибко настраивать доступ к различным функциям и информации.

Важным преимуществом YouGile является его кросс-платформенность и мобильная поддержка. Пользователи могут получать доступ к своим проектам и задачам через веб-интерфейс или мобильные приложения, что позволяет управлять проектами в любое время и из любого места.

**Преимущества:** интеграция с методологией SCRUM, встроенные инструменты для аналитики и отчетности, гибкая настройка досок и задач.

**Недостатки:** более крупное обучение персонала, платная версия для расширенных возможностей.

**Вывод: «**YouGile» предоставляет мощные инструменты для управления проектами, особенно для команд, следующих методологии SCRUM. Он подходит для проектов среднего и большого масштаба, где требуется более сложное управление и анализ данных.

Также в качестве аналогов я выделил следующие приложения:

* Pyrus;
* Jira;
* ClickUp;
* Wrike.

Все рассмотренные аналоги имеют схожую тематику и функционал, но у каждого есть отличия, что и делает каждое из них уникальным.

## Требования к проектируемому программному средству

Основываясь на анализе аналогичных решений из пункта 1.1, проектируемое программное средство должно включать следующие характеристики и функциональные возможности.

Оно должно предоставлять интуитивно понятный и визуально привлекательный интерфейс, который позволит пользователям создавать задачи и проекты. Графическое представление задач и их статусов должно быть четким и удобным для отслеживания прогресса.

Гибкость и настраиваемость являются важными аспектами. Пользователи должны иметь возможность настраивать структуру задач в соответствии с потребностями проекта, включая добавление задач и дополнительных сведений (прим. мультимедийных файлов). Гибкая система управления доступом должна позволять определить права и роли участников проекта.

Управление сроками и приоритетами также является важной функциональностью. Пользователи должны иметь возможность устанавливать сроки выполнения задач и определять их приоритеты. Программное средство должно предоставлять удобные инструменты для отслеживания сроков и уведомлять пользователей о предстоящих сроках.

Для обеспечения эффективной коммуникации и совместной работы, программное средство должно позволять пользователям оставлять комментарии, обсуждать задачи и обмениваться файлами в рамках программы. Оно также должно обеспечивать удобные инструменты для совместной работы и взаимодействия между участниками проекта.

Аналитика и отчетность представляют значимый аспект. Программное средство должно предоставлять аналитические возможности, такие как отчеты о выполнении задач, чтобы помочь в управлении проектами и принятии информированных решений.

Интеграция с другими инструментами, такими как электронная почта также является важной характеристикой. Программное средство должно обеспечивать возможность интеграции с различными популярными инструментами и сервисами, чтобы обеспечить более эффективный и синхронизированный рабочий процесс.

Уведомления и оповещения являются неотъемлемой частью программного средства. Оно должно предоставлять функциональность для отправки уведомлений и оповещений, чтобы информировать участников проекта о важных событиях, изменениях и обновлениях. Это помогает участникам оставаться в курсе и своевременно реагировать на изменения.

Суммируя, проектируемое программное средство должно обладать привлекательным интерфейсом, гибкой структурой задач, возможностью управления сроками и приоритетами, инструментами коммуникации и совместной работы, аналитическими возможностями, интеграцией с другими инструментами и функциональностью уведомлений. Учитывая эти характеристики, новое программное средство сможет эффективно поддерживать организацию задач и управление проектами.

# Проектирование базы данных

Процесс проектирования базы данных состоит из следующих этапов:

* сбор информации;
* определение сущностей;
* определение атрибутов для каждой сущности;
* определение связей между сущностями;
* нормализация.

На этапе сбора информации вам необходимо точно определить, как будет использоваться база данных, и какая информация будет в ней храниться.

Далее следует этап определения сущностей и на нем определяются сущности, из которых будет состоять база данных.

Сущность — это объект в базе данных, в котором хранятся данные. Сущность может представлять собой нечто вещественное (дом, человек, предмет, место) или абстрактное (банковская операция, отдел компании, маршрут автобуса). В физической модели сущность называется таблицей. Сущности состоят из атрибутов (столбцов таблицы) и записей (строк в таблице).

Обычно базы данных состоят из нескольких основных сущностей, связанных с большим количеством подчиненных сущностей. Основные сущности называются независимыми: они не зависят ни от какой-либо другой сущности. Подчиненные сущности называются зависимыми: для того, чтобы существовала одна из них, должна существовать связанная с ней основная таблица.

Следующий этап — это определение атрибутов созданных сущностей. Атрибут представляет свойство, описывающее сущность. Атрибуты часто бывают числом, датой или текстом. Все данные, хранящиеся в атрибуте, должны иметь одинаковый тип и обладать одинаковыми свойствами.

В физической модели атрибуты называют колонками. После определения сущностей необходимо определить все атрибуты этих сущностей. На диаграммах атрибуты обычно перечисляются внутри прямоугольника сущности.

На этапе определения связей работа происходит с ключами сущностей. Ключом называется набор атрибутов, однозначно определяющий запись. Ключи делятся на два класса: простые и составные.

Простой ключ состоит только из одного атрибута. Например, в базе «Паспорта граждан страны» номер паспорта будет простым ключом: ведь не бывает двух паспортов с одинаковым номером.

Составной ключ состоит из нескольких атрибутов. В той же базе «Паспорта граждан страны» может быть составной ключ со следующими атрибутами: фамилия, имя, отчество, дата рождения. Это — как пример, т. к. этот составной ключ, теоретически, не обеспечивает гарантированной уникальности записи.

Первичным ключом называется совокупность атрибутов, однозначно идентифицирующих запись в таблице (сущности). Один из возможных ключей становится первичным ключом. На диаграммах первичные ключи часто изображаются выше основного списка атрибутов или выделяются специальными символами.

Любой возможный ключ, не являющийся первичным, называется альтернативным ключом. Сущность может иметь несколько альтернативных ключей. Внешним ключом называется совокупность атрибутов, ссылающихся на первичный или альтернативный ключ другой сущности. Если внешний ключ не связан с первичной сущностью, то он может содержать только неопределенные значения. Если при этом ключ является составным, то все атрибуты внешнего ключа должны быть неопределенными. На диаграммах атрибуты, объединяемые во внешние ключи, обозначаются специальными символами.

Реляционные базы данных позволяют объединять информацию, принадлежащую разным сущностям. Отношение — это ситуация, при которой одна сущность ссылается на первичный ключ второй сущности. Они определяются в процессе проектирования базы. Для этого следует проанализировать сущности и выявить логические связи, существующие между ними.

Тип отношения определяет количество записей сущности, связанных с записью другой сущности. Отношения делятся на три основных типа:

* Один-к-одному. Каждой записи первой сущности соответствует только одна запись из второй сущности. А каждой записи второй сущности соответствует только одна запись из первой сущности. Например, есть две сущности: Люди и Свидетельства о рождении. И у одного человека может быть только одно свидетельство о рождении.
* Один-ко-многим. Каждой записи первой сущности могут соответствовать несколько записей из второй сущности. Однако каждой записи второй сущности соответствует только одна запись из первой сущности. Например, есть две сущности: Заказ и Позиция заказа. И в одном заказе может быть много товаров.
* Многие-ко-многим. Каждой записи первой сущности могут соответствовать несколько записей из второй сущности. Однако и каждой записи второй сущности может соответствовать несколько записей из первой сущности. Например, есть две сущности: Автор и Книга. Один автор может написать много книг. Но у книги может быть несколько авторов.

По критерию обязательности отношения делятся на обязательные и необязательные.

* Обязательное отношение означает, что для каждой записи из первой сущности непременно должны присутствовать связанные записи во второй сущности.
* Необязательное отношение означает, что для записи из первой сущности может и не существовать записи во второй сущности.

Далее следует этап нормализации. Нормализацией называется процесс удаления избыточных данных из базы данных. Каждый элемент данных должен храниться в базе в одном и только в одном экземпляре. Существует пять распространенных форм нормализации. Как правило, база данных приводится к третьей нормальной форме.

В процессе нормализации выполняются определенные действия по удалению избыточных данных. Нормализация повышает быстродействие, ускоряет сортировку и построение индекса, уменьшает количество индексов на сущность, ускоряет операции вставки и обновления.

Ограничения — это правила, за соблюдением которых следит система управления базы данных. Ограничения определяют множество значений, которые можно вводить в столбец или столбцы.

Организовав данные в таблицы и определив связи между ними, можно считать, что была создана модель, правильным образом отражающая бизнес-среду. Теперь нужно обеспечить, чтобы данные, вводимые в базу, давали правильное представление о состоянии дела. Иными словами, нужно обеспечить выполнение деловых правил и поддержку целостности базы данных.

Например, ваша компания занимается доставкой книг. Вы вряд ли примете заказ от неизвестного клиента, ведь тогда вы даже не сможете доставить заказ. Отсюда бизнес-правило: заказы принимаются только от клиентов, информация о которых есть в базе данных.

Для реализации необходимого функционала была создана база данных, таблицы которой можно разделить на 2 логические группы: таблицы для хранения информации об пользователях и таблицы для хранения информации об созданных проектах.

Рассмотрим таблицы базы данных на примере диаграммы, представленной на рисунке 2.1.

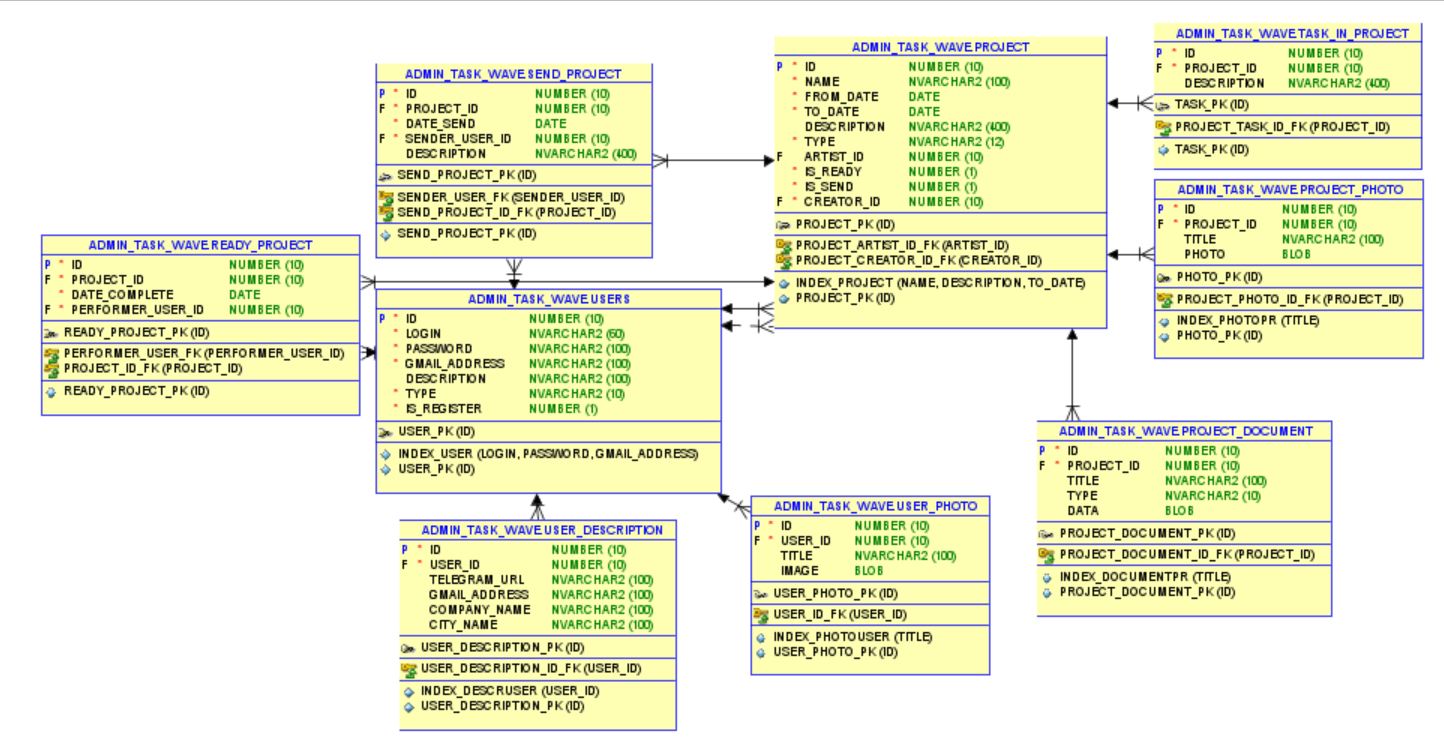


Рисунок 2.1 – Диаграмма таблиц базы данных

База данных состоит из 9 таблиц, связанных между собой внешними ключами.

Таблица «*PROJECT*» хранит основную информацию про проекты, сопутствующие таблицы будут ссылаться на её первичный ключ.

Таблица «*TASK\_IN\_PROJECT*» хранит информацию об задачах, входящих в состав проектов.

Таблица «*PROJECT\_PHOTO*» хранит в себе информацию про фотографии, входящие в проект.

Таблица «PROJECT\_DOCUMENT» хранит в себе информацию про документы, входящие в проект.

Таблица «*SEND\_PROJECT*» хранит информацию об отправленных на проверку проектов.

Таблица «*READY\_PROJECT*» хранит информацию об выполненных проектах.

Таблица «*USERS*» содержит информацию про пользователей.

Таблица «*USER\_DESCRIPTION*» cодержит информацию про описание пользователей.

Таблица «*USER\_PHOTO*» хранит информацию об фотографиях пользователей.

Скрипт создания всех таблиц представлен в Приложении Б.

# Разработка объектов базы данных

## Пользователи

Пользователь базы данных – это физическое или юридическое лицо, которое имеет доступ к БД и пользуется услугами информационной системы для получения информации.

При проектировании базы данных было использовано 3 пользователя. Каждый пользователь имеет свой профиль и табличные пространства.

Первый пользователь – Admin\_Task\_Wave – зарегистрированный пользователь, который обладает правами администратора. Используя этого пользователя, создавались основные объекты базы данных. Он имеет доступ ко всем объектам PDB. С помощью него можно просматривать всех пользователей и при необходимости их удалять. Также пользователь может подтверждать (или не подтверждать) регистрацию пользователей с расширенными правами. Код для создания администратора представлен в листинге 3.1.

CREATE USER ADMIN\_TASK\_WAVE IDENTIFIED BY 1  
PROFILE admin\_TW\_profile  
DEFAULT TABLESPACE TASK\_WAVE\_ADMIN\_TS  
TEMPORARY TABLESPACE TASK\_WAVE\_ADMIN\_TS\_TEMP  
ACCOUNT UNLOCK  
PASSWORD EXPIRE;

Листинг 3.1 – Скрипт создания пользователя Admin\_Task\_Wave

Второй пользователь – User\_Task\_Wave– обладает правами для стандартного пользователя, использующего приложения. Может создавать личные проекты, добавлять к ним описание и различные файлы. Код создания пользователя представлен в листинге 3.2.

CREATE USER USER\_TASK\_WAVE IDENTIFIED BY 1  
PROFILE USER\_TW\_profile  
DEFAULT TABLESPACE TASK\_WAVE\_USER\_TS  
TEMPORARY TABLESPACE TASK\_WAVE\_USER\_TS\_TEMP  
ACCOUNT UNLOCK  
PASSWORD EXPIRE;

Листинг 3.2 – Скрипт создания пользователя User\_Task\_Wave

Третий пользователь – Super\_User\_Task\_Wave – обладает правами пользователя с расширенными правами, что добавляет к возможностям стандартного пользователя, создавания командных проектов, в ходе которых можно определять исполнителя, а затем проверять отправленные им проекты.

Стоит отметить, что для создания такого пользователя требуется одобрение от администратора и изначально такой пользователь отмечен как не зарегистрированный и, соответственно, не имеет возможности войти в аккаунт. Код для создания пользователя представлен в листинге 3.3.

CREATE USER SUPER\_USER\_TASK\_WAVE IDENTIFIED BY 1  
PROFILE SUPER\_USER\_TW\_profile  
DEFAULT TABLESPACE TASK\_WAVE\_SUPER\_USER\_TS  
TEMPORARY TABLESPACE TASK\_WAVE\_SUPER\_USER\_TS\_TEMP  
ACCOUNT UNLOCK  
PASSWORD EXPIRE;

Листинг 3.3 – Скрипт создания пользователя Super\_User\_Task\_Wave

Скрипт выдачи прав пользователям на процедуры и функции представлен в приложении В.

## Функции

Для управления данными через приложение все пользователи использует функции. Функция – объект базы данных, представляющий собой набор SQL-инструкций, который компилируется один раз и хранится на сервере. Стоит отметить, что функции не могут содержать группы операторов DML и DDL. В моём приложении было реализовано 21 функция. Пример создания функции представлен на листинге 3.2. Остальные функции представлены в приложении Г.

------------getUserById--------------  
CREATE OR REPLACE FUNCTION *getUserById*(  
 idUser in USERS.id%type  
)  
RETURN SYS\_REFCURSOR  
IS  
 user\_cursor SYS\_REFCURSOR;  
BEGIN  
 OPEN user\_cursor FOR  
 SELECT ID, LOGIN, PASSWORD, GMAIL\_ADDRESS, DESCRIPTION, TYPE  
 FROM users  
 WHERE USERS.ID = idUser  
 and  
 users.IS\_REGISTER = 1;  
 RETURN user\_cursor;  
END *getUserById*;

Листинг 3.4 – Функция поиска пользователя по id

В данном примере функция принимает в качестве параметра Id пользователя и возвращает переменную типа «RefCursor», в которую записывается результат выполнения select-запроса (информацию про пользователя).

## Хранимые процедуры

По большому счёту, процедуры выполняют аналогичные действия с функциями, за тем исключением, что они не могут возвращать значений (но могут иметь выходные параметры) и имеют возможность в теле использовать DML-операторы. Всего было разработано мною 17 процедур. В приложении Д представлен листинг со всеми созданными процедурами. А на листинге 3.5 представлен пример реализации одной из процедур.

CREATE OR REPLACE PROCEDURE *CreateStandartUser*(  
 user\_login IN USERS.login%TYPE,  
 user\_password IN USERS.password%TYPE,  
 user\_email IN USERS.gmail\_address%TYPE,  
 user\_description IN USERS.description%TYPE := null  
)  
IS  
 cnt NUMBER;  
 encrypted\_password\_result USERS.password%type;  
 user\_type USERS.type%TYPE := 'user';  
 user\_is\_register USERS.is\_register%TYPE := 1;  
  
 BEGIN  
 SELECT *COUNT*(\*) INTO cnt FROM USERS where *UPPER*(LOGIN) = *UPPER*(user\_login);  
 IF(cnt=0) then  
 encrypted\_password\_result := *encrypt\_password*(user\_password);  
 INSERT INTO USERS(login, password, GMAIL\_ADDRESS,DESCRIPTION, type, is\_register)  
 values(  
 user\_login, encrypted\_password\_result,user\_email,user\_description, user\_type, user\_is\_register  
 );  
 COMMIT;  
 ELSE  
 *RAISE\_APPLICATION\_ERROR*(-20001, 'This person already exists');  
 END IF;  
end *CreateStandartUser*;

Листинг 3.5 – Процедура создания стандартного пользователя

Демонстрационная процедура создает стандартного пользователя, в качестве параметров принимает логин, пароль, email и описания пользователя. Следует обратить внимание, что она выполняют проверку на наличия пользователя с переданным в качестве параметров login-ом, а в случае, если такой пользователь существует, то будет выброшено исключение.

## Индексы

Для оптимизации выполнения select-запросов (выборки данных), было разработано ряд индексов. Все они представлены на листинге 3.6.

Индексы в базе данных представляют собой структуры, созданные для ускорения процессов поиска, сортировки и фильтрации данных. Они облегчают выполнение запросов к базе данных, позволяя системе быстрее находить и получать данные, необходимые для выполнения определенного запроса. Индексы оптимизируют производительность базы данных, позволяя ей эффективно обрабатывать большие объемы данных и ускорять выполнение сложных запросов. Их использование способствует повышению эффективности работы с базой данных, а также повышению производительности и доступности системы в целом.

CREATE INDEX index\_user  
 ON USERS(LOGIN, PASSWORD, GMAIL\_ADDRESS);  
  
create INDEX index\_project  
 on PROJECT(NAME, DESCRIPTION, TO\_DATE);  
  
create INDEX index\_photoPr  
 on PROJECT\_PHOTO(TITLE);  
  
create INDEX index\_documentPr  
 on PROJECT\_DOCUMENT(TITLE);  
  
create index index\_descrUser  
 on USER\_DESCRIPTION(USER\_ID);  
  
CREATE index index\_photoUser  
 on USER\_PHOTO(TITLE);

Листинг 3.6 – Индексы

Можно заметить, что для всех основных таблиц были созданы индексы со столбцами, которые чаще всего используются в выборке.

# Описание процедур импорта и экспорта

Для экспорта/импорта данных был выбран формат JSON.

JSON (JavaScript Object Notation) – универсальный формат данных, понятный как человеку, так и компьютеру. Он представляет собой простой и легкий способ описания и обмена информацией между различными программными системами. JSON основывается на объектах и массивах, которые могут содержать значения различных типов данных, включая числа, строки, логические значения, массивы и другие объекты. Этот формат используется во множестве приложений, таких как веб-разработка, облачные сервисы и мобильные приложения, для передачи и сохранения данных.

Для экспорта строк из таблицы Project в формат JSON была разработана функция Export\_Project\_JSON, в качестве параметра она принимает ID проекта. Реализация функции представлена в листинге 4.1.

CREATE OR REPLACE FUNCTION *EXPORT\_PROJECT\_JSON* (p\_project\_id IN NUMBER)  
RETURN CLOB  
AS  
 v\_json CLOB;  
BEGIN  
 SELECT *JSON\_OBJECT*(  
 'id' VALUE id,  
 'name' VALUE name,  
 'from\_date' VALUE from\_date,  
 'to\_date' VALUE to\_date,  
 'description' VALUE description,  
 'type' VALUE type,  
 'artist\_id' VALUE artist\_id,  
 'is\_ready' VALUE is\_ready,  
 'is\_send' VALUE is\_send,  
 'creator\_id' VALUE creator\_id  
 )  
 INTO v\_json  
 FROM PROJECT  
 WHERE id = p\_project\_id;  
  
 RETURN v\_json;  
EXCEPTION  
 WHEN NO\_DATA\_FOUND THEN  
 RETURN NULL;  
END *EXPORT\_PROJECT\_JSON*;

Листинг 4.1 – Функция Export\_Project\_JSON

Данные в JSON-формате будут выведены на консоль. После чего, их можно будет использовать.

Для импорта данных в таблицу Project из файла формата json была разработана процедура Import\_Project\_JSON, которая представлена на листинге 4.2 и 4.3. Поскольку процедура довольно длинная, она была разбита на две части.

CREATE OR REPLACE PROCEDURE *IMPORT\_PROJECT\_JSON* (p\_json\_data IN CLOB)  
AS  
 v\_name NVARCHAR2(100);  
 v\_from\_date DATE;  
 v\_to\_date DATE;  
 v\_description NVARCHAR2(400);  
 v\_type NVARCHAR2(12);  
 v\_artist\_id NUMBER;  
 v\_is\_ready NUMBER(1);  
 v\_is\_send NUMBER(1);  
 v\_creator\_id NUMBER;  
BEGIN  
 SELECT *JSON\_VALUE*(p\_json\_data, '$.name') INTO v\_name FROM DUAL;  
 SELECT *TO\_DATE*(*JSON\_VALUE*(p\_json\_data, '$.from\_date'), 'YYYY-MM-DD"T"HH24:MI:SS') INTO v\_from\_date FROM DUAL;  
 SELECT *TO\_DATE*(*JSON\_VALUE*(p\_json\_data, '$.to\_date'), 'YYYY-MM-DD"T"HH24:MI:SS') INTO v\_to\_date FROM DUAL;  
 SELECT *JSON\_VALUE*(p\_json\_data, '$.description') INTO v\_description FROM DUAL;  
 SELECT *JSON\_VALUE*(p\_json\_data, '$.type') INTO v\_type FROM DUAL;  
 SELECT *JSON\_VALUE*(p\_json\_data, '$.artist\_id') INTO v\_artist\_id FROM DUAL;  
 SELECT *JSON\_VALUE*(p\_json\_data, '$.is\_ready') INTO v\_is\_ready FROM DUAL;  
 SELECT *JSON\_VALUE*(p\_json\_data, '$.is\_send') INTO v\_is\_send FROM DUAL;  
 SELECT *JSON\_VALUE*(p\_json\_data, '$.creator\_id') INTO v\_creator\_id FROM DUAL;

Листинг 4.2 – Процедура Import\_Project\_JSON

В данном фрагменте кода создается процедура IMPORT\_PROJECT\_JSON, которая принимает входной параметр p\_json\_data типа CLOB. Параметр p\_json\_data представляет собой JSON-объект, содержащий информацию о проекте.

Внутри процедуры создаются переменные v\_name, v\_from\_date, v\_to\_date, v\_description, v\_type, v\_artist\_id, v\_is\_ready, v\_is\_send и v\_creator\_id, которые будут использоваться для хранения полученных значений из JSON-объекта.

Для извлечения значений из JSON-объекта и их приведения к соответствующим типам данных, используется функция JSON\_VALUE. Использование данной функции позволяет получить значение определенного поля из JSON-объекта. Например, JSON\_VALUE(p\_json\_data, '$.name') вернет значение поля 'name' из JSON-объекта, переданного в параметре p\_json\_data.

Однако следует обратить внимание, что в данном случае отсутствует проверка на совместимость типов данных. То есть, при выполнении данной процедуры необходимо иметь гарантию, что данные, переданные в параметре p\_json\_data, будут представлены в ожидаемом формате и типе. В противном случае, может произойти ошибка при попытке приведения полученных значений к соответствующим типам данных.

Таким образом, процедура IMPORT\_PROJECT\_JSON позволяет извлечь и сохранить значения полей из JSON-объекта, переданного в параметре p\_json\_data, но требует внимательности при использовании, чтобы гарантировать правильность формата и типов данных.

INSERT INTO PROJECT (  
 name,  
 from\_date,  
 to\_date,  
 description,  
 type,  
 artist\_id,  
 is\_ready,  
 is\_send,  
 creator\_id  
 )  
 VALUES (  
 v\_name,  
 v\_from\_date,  
 v\_to\_date,  
 v\_description,  
 v\_type,  
 v\_artist\_id,  
 v\_is\_ready,  
 v\_is\_send,  
 v\_creator\_id  
 );  
  
 COMMIT;  
EXCEPTION  
 WHEN OTHERS THEN  
 DBMS\_OUTPUT.*PUT\_LINE*('Error: ' || *SQLERRM*);  
 ROLLBACK;  
END *IMPORT\_PROJECT\_JSON*;

Листинг 4.3 – Процедура Import\_Project\_JSON (продолжение)

Далее представлена непосредственно операция добавления новой строки в таблицу Project.

# Тестирования производительности

Тестирование производительности проводилось на таблице Users, которая заполнялась на 100000 строк при помощи цикла. Анонимный блок по заполнению строк представлен в листинге 5.1. Операция вставки такого количества строк заняла порядка 16 минут.

DECLARE  
 v\_user\_login VARCHAR2(50);  
 v\_user\_password VARCHAR2(50);  
 v\_user\_email VARCHAR2(50);  
 v\_user\_description VARCHAR2(100);  
  
BEGIN

FOR i IN 1000..100000 LOOP  
 -- Генерация случайных данных  
 v\_user\_login := 'user' || i;  
 v\_user\_password := 'password' || i;  
 v\_user\_email := 'user' || i || '@example.com';  
 v\_user\_description := 'Description for user ' || i;  
  
 -- Вызов процедуры для вставки пользователя  
 *CreateStandartUser*(  
 user\_login => v\_user\_login,  
 user\_password => v\_user\_password,  
 user\_email => v\_user\_email,  
 user\_description => v\_user\_description  
 );  
 END LOOP;

END;

Листинг 5.1 – Добавление 100000 строк

Далее из этой таблицы получаем пользователя с заданным логином ‘User10000’. Пример демонстрируется на листинге 5.2.

explain plan for select \*  
from USERS  
where LOGIN = 'user10000';  
select \* from table(DBMS\_XPLAN.*DISPLAY*());

Листинг 5.2 – Выборка пользователя с заданным логином

Стоимость данной операции представлена на рисунке 5.1. Следует обратить внимание, что стоимость запроса оценивается до создания индекса. Поэтому данные затраты далеки от того, что может быть на самом деле. В дальнейшем мы оптимизируем наши таблицы при помощи индекса и сравним полученные значения до его создания и после.

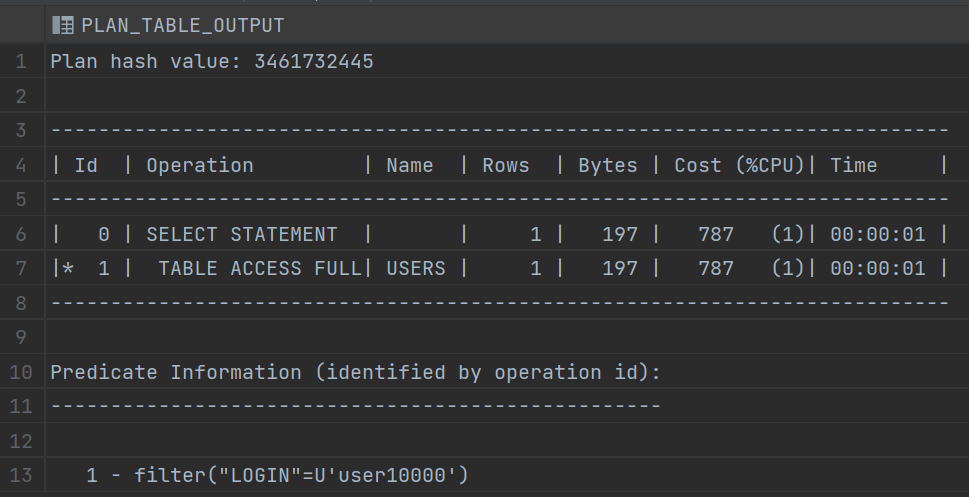


Рисунок 5.1 – Стоимость запроса

Теперь, для повышения производительности запроса, был создан индекс, листинг на который, уже был представлен в одном из предыдущих разделов. После создания индекса была выполнена повторная оценка производительности, результаты представлены на рисунке 5.2.

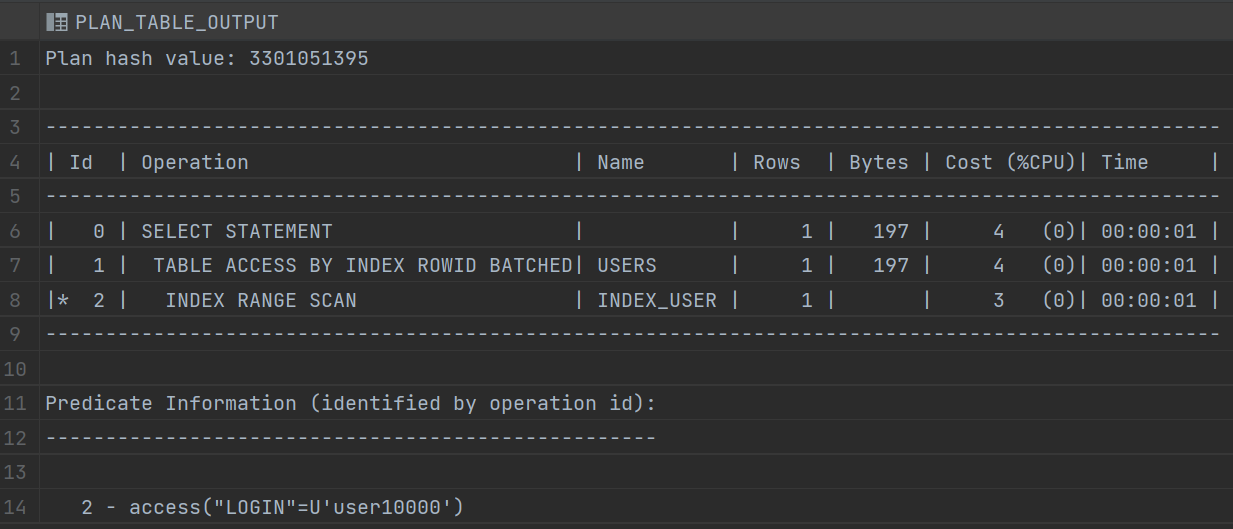


Рисунок 5.2 – Стоимость запроса после создания индекса

Как можно заметить, стоимость запроса после создания индекса уменьшилась. Таким образом, с помощью индексов мы можем увеличить производительность наших процедур и функций, что позволит нам быстрее получать необходимую информацию.

# Описание технологии и её применение в базе данных

## Шифрование данных

В разработке своего проекта я использовала технологию шифрования данных. В Oracle для этого чаще всего используются следующие алгоритмы:

* DES (Data Encryption Standard). Традиционно алгоритм DES занимал ведущие позиции в области шифрования. Алгоритму DES необходим 64-разрядный ключ, но 8 бит ключа не используются. Чтобы подобрать ключ, злоумышленнику придется перебрать до 72 057 594 037 927 936 комбинаций.
* DES3. В этой схеме, базирующейся на исходном алгоритме DES, данные шифруются дважды или трижды (в зависимости от режима вызова). DES3 использует 128- или 192-разрядный ключ; его длина определяется количеством проходов.
* AES. В ноябре 2001 года был одобрен новый стандарт AES (Advanced Encryption Standard), вступивший в силу в мае 2002 года.

Блок данных обычно не шифруется как единое целое. Чаще всего он разбивается на фрагменты по 8 байт, после чего каждый фрагмент шифруется независимо от других. Конечно, длина данных может быть не кратной 8 — в этом случае алгоритм добавляет символы в последний фрагмент до 8 байт. Этот процесс называется заполнением (padding).

Шифрация данных происходит при помощи функции DBMS\_CRYPTO.ENCRYPT. Функция получает четыре аргумента:

* src — исходные данные, подлежащие шифрованию (должны иметь тип данных RAW).
* key — ключ шифрования (также RAW). Длина ключа должна соответствовать выбранному алгоритму. Например, для алгоритма DES она должна быть не менее 64 бит.
* typ — определение трех компонентов (алгоритм, механизм заполнения и метод сцепления) в виде суммы соответствующих констант.
* iv — необязательный вектор инициализации (IV), еще один компонент схемы шифрования, затрудняющий анализ «закономерностей» в зашифрованном тексте.

В своем приложении я использовала:

* алгоритм — AES c 128-разрядным ключом;
* метод сцепления — CBC;
* механизм заполнения — PKCS#5.

Для шифрования необходимо выбрать ключ. В приложении в качестве ключа используется строка «2901200316052000». Это значение относится к типу данных VARCHAR2. Чтобы использовать его в функции ENCRYPT, необходимо сначала преобразовать его к типу RAW. Для этого мы воспользуемся функцией STRING\_TO\_RAW встроенного пакета UTL\_I18N. Также вторым параметром указываем кодировку.

Переменная l\_in\_val типа VARCHAR2 также должна быть преобразована к типу RAW. Переменную, которая будет передаваться в качестве возвращаемого значения функции, требуется преобразовать в шестнадцатеричный формат. Для этого используем функцию RAWTOHEX.

Процедура шифрования пароля представлена ниже в листинге 6.1:

create or replace  
FUNCTION *encrypt\_password*(i\_password in users.password%type)  
 RETURN USERS.password%type  
 IS  
 my\_key VARCHAR2(2000) := '2023110909112023';  
 my\_val VARCHAR2(2000) := i\_password;  
 my\_mod NUMBER := sys.DBMS\_CRYPTO.encrypt\_aes128  
+ sys.DBMS\_CRYPTO.chain\_cbc  
+ sys.DBMS\_CRYPTO.pad\_pkcs5;  
 encrypted\_password RAW(2000);  
BEGIN  
 encrypted\_password := sys.DBMS\_CRYPTO.*encrypt*(utl\_i18n.*string\_to\_raw*(my\_val, 'AL32UTF8'), my\_mod,  
utl\_i18n.*string\_to\_raw*(my\_key, 'AL32UTF8'));  
 RETURN *RAWTOHEX*(encrypted\_password);  
END *encrypt\_password*;

Листинг 6.1 – Шифрование данных с помощью алгоритма AES128

Шифрование данных имеет смысл только в том случае, если зашифрованные данные в какой-то момент будут прочитаны и использованы в приложении. Эта задача решается при помощи функции DECRYPT. По структуре вызова она идентична функции ENCRYPT и получает те же четыре аргумента:

* src — зашифрованные данные.
* key — ключ, использованный для шифрования.
* typ — три компонента (алгоритм, механизм заполнения и метод сцепления), использованные при вызове ENCRYPT.
* iv — вектор инициализации, использованный при вызове ENCRYPT.

## Маскирование данных

Маскирование — это процесс скрытия или изменения идентификационных данных или физического вида человека, объекта или системы, с целью представить или представить себя в другом образе или виде. В общем понятии маскировка используется для обмана или обхода противника, защиты личности или создания впечатления о том, что что-то или кто-то не является тем, кем он на самом деле является.

В контексте компьютерной безопасности маскирование может применяться для сокрытия присутствия злоумышленников или их действий в сети, а также для обмана средств обнаружения и защиты, таких как антивирусные программы или брандмауэры. Это может включать использование фальшивых идентификационных данных, изменение сетевых адресов или использование специальных программ и инструментов для скрытия деятельности.

Маскирование также может применяться в маскировочном искусстве, театре, кино и косметологии, где люди могут использовать макияж, маски, костюмы или протезы, чтобы изменить свой физический облик и создать новый образ.

Важно отметить, что маскирование может иметь и положительные, и отрицательные аспекты, и его использование может зависеть от контекста и целей, для которых оно применяется.

Для реализации маскирования данных я возвращал пароль пользователям в виде нескольких звёздочек. Это происходило преимущественно в функциях, которые возвращали информацию про пользователей. Фрагмент функции с примером возвращения таких данных представлена на листинге 6.2.

…  
 SELECT  
 ud.ID as ID,  
 ud.GMAIL\_ADDRESS AS GMAIL,  
 ud.TELEGRAM\_URL AS TELEGRAM,  
 ud.COMPANY\_NAME AS COMPANY,  
 ud.city\_name AS CITY,  
  
 u.LOGIN AS LOGIN,  
 CASE  
 WHEN u.PASSWORD IS NOT NULL THEN '\*\*\*\*\*\*\*\*'  
 ELSE NULL  
 END AS PASSWORD,  
  
 UP.TITLE AS TITLEPHOTO,  
 UP.IMAGE AS IMAGE  
 FROM  
 USERS U  
 LEFT JOIN USER\_DESCRIPTION ud on U.ID = ud.USER\_ID  
 LEFT JOIN USER\_PHOTO UP on U.ID = UP.USER\_ID

…

Листинг 6.2 – Пример маскирования данных

Как мы видим, с помощью конструкции case-when мы преобразовываем наш пароль к маскированному виду.

## Хранение мультимедийных типов данных

Для работы с данными большого объёма СУБД Oracle предоставляет типы данных BLOB, CLOB, NCLOB и BFILE. Здесь LOB означает large object, или большой объект, и далее по тексту термины LOB и «большой объект» взаимозаменяемы. По сути, большой объект — это абстрактный тип для манипуляции данными большого объема внутри БД, а типы BLOB, CLOB, NCLOB и BFILE - его конкретные реализации.

В курсовом проекте технология мультимедийных типов данных отражена в виде хранимых изображений для пользователей и проектов, документов для проектов.

Для хранения изображений и файловых документов в СУБД применяется тип BLOB. Тип BLOB представляет из себя массив двоичных данных. Пример таблицы, которая содержит изображение, приведен ниже на листинге 6.3:

CREATE TABLE PROJECT\_PHOTO(  
 id NUMBER(10) GENERATED AS IDENTITY (START WITH 1 INCREMENT BY 1),  
 project\_id NUMBER(10) NOT NULL,  
 title NVARCHAR2(100) default null,  
 photo BLOB DEFAULT *EMPTY\_BLOB*(),  
 CONSTRAINT photo\_pk PRIMARY KEY (id),  
 CONSTRAINT project\_photo\_id\_fk FOREIGN KEY (project\_id) REFERENCES PROJECT(id)  
);

Листинг 6.3 – Таблица с фотографией

Как мы видим, для атрибута photo используется тип данных BLOB, который по умолчанию имеет значение null.

# Краткое описание приложения для демонстрации

Приложение «TaskWave» — это web-приложение, которое позволяет создавать и управлять проектами, присутствует возможность создавать как личные, так и командные проекты. Приложение поддерживает три роли, каждая роль имеет свои права и возможности. На рисунке 7.1 представлен логотип приложения.



Рисунок 7.1 – Логотип «TaskWave»

В данном разделе речь пойдёт преимущественно про цели и задачи, для которых разрабатывалось данное приложение. Всё руководство по использованию, включая наглядные демонстрации будут представлены в следующем разделе.

TaskWave — инновационное приложение, созданное специально для повышения эффективности планирования и управления проектами. Оно позволяет организовывать работу, присваивать задачи и следить за их выполнением. TaskWave — ваш незаменимый партнер в достижении целей и успеха вашего проекта.

Особенности приложения:

1. Создание и назначение задач:

TaskWave предоставляет простой и интуитивно понятный интерфейс, который позволяет создавать задачи и назначать их соответствующим членам команды. Вы можете установить сроки выполнения, и добавить комментарии, чтобы каждый участник знал, что от него требуется.

2. Управление проектами:

Необходимость в отслеживании прогресса проекта? TaskWave предоставляет информацию про проекты, где вы можете видеть статус каждой задачи и их зависимости друг от друга. Отслеживайте прогресс, вносите изменения и управляйте проектом с легкостью.

3. График и статистика:

TaskWave имеет аналитический инструмент, который позволяет вам видеть график выполненных задач по месяцам. Это помогает вам анализировать эффективность работы и улучшать процессы для более успешных результатов.

4. Совместная работа:

Члены команды могут выполнять проекты и добавлять комментарии при выполнении, что позволяет всей команде быть в курсе и делать правильные шаги вместе.

Безопасность и надежность:

TaskWave обеспечивает высокий уровень безопасности данных. Такие личные данные, как пароль, шифруются и в дальнейшем работа происходит с зашифрованным паролем.

Простота использования:

TaskWave разработан с учетом удобства использования. Интуитивный интерфейс позволяет быстро освоиться даже новым пользователям. Благодаря простоте и понятности структуры приложения, ваша команда сможет начать работу сразу же без дополнительных обучающих программ.

Мобильное приложение:

Не беспокойтесь о работе вдали от офиса. TaskWave представляет собой кроссплатформенное приложение, которое позволяет вам иметь доступ к вашим проектам и задачам в любом месте. Вы можете обновлять статусы, добавлять новые задачи и контролировать прогресс, где бы вы ни находились.

TaskWave — это идеальное решение для планирования и управления проектами. Оно позволяет организовывать работу и коммуникацию в одной интуитивно понятной среде. Забудьте о неэффективном использовании времени и проблемах с координацией проектов - TaskWave поможет вам достигнуть новых уровней продуктивности и успеха.

# Руководство пользователя

Первый раз, при заходе на сайт, пользователю будет показано окно с авторизацией, при отсутствии учётном записи пользователь может перейти на панель регистрации и зарегистрироваться. На рисунке 8.1 представлено окно авторизации.

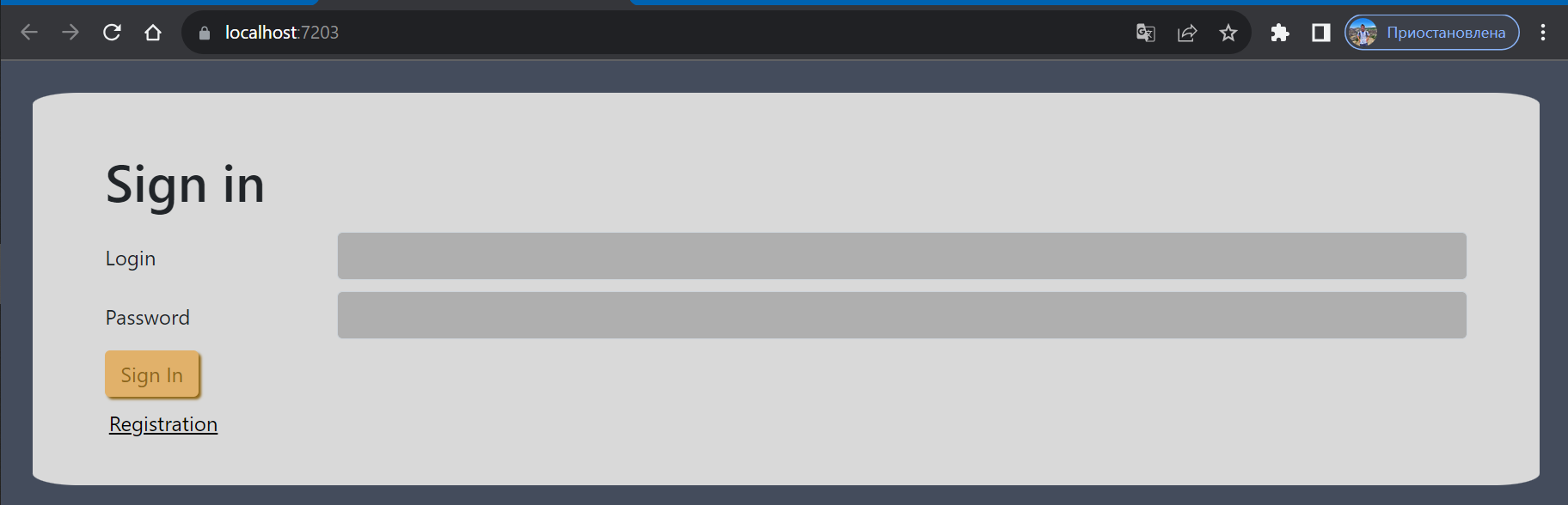


Рисунок 7.1 – Окно авторизации

При аутентификации пользователю необходимо ввести свой логин и пароль, затем нажать на кнопку «Sign In». При правильном вводе данных, произойдёт переход на главную страницу соответствующего пользователя (далее мы рассмотрим все варианты).

При переходе в окно регистрации нам будет представлен следующий интерфейс, который демонстрируется на рисунке 7.2.

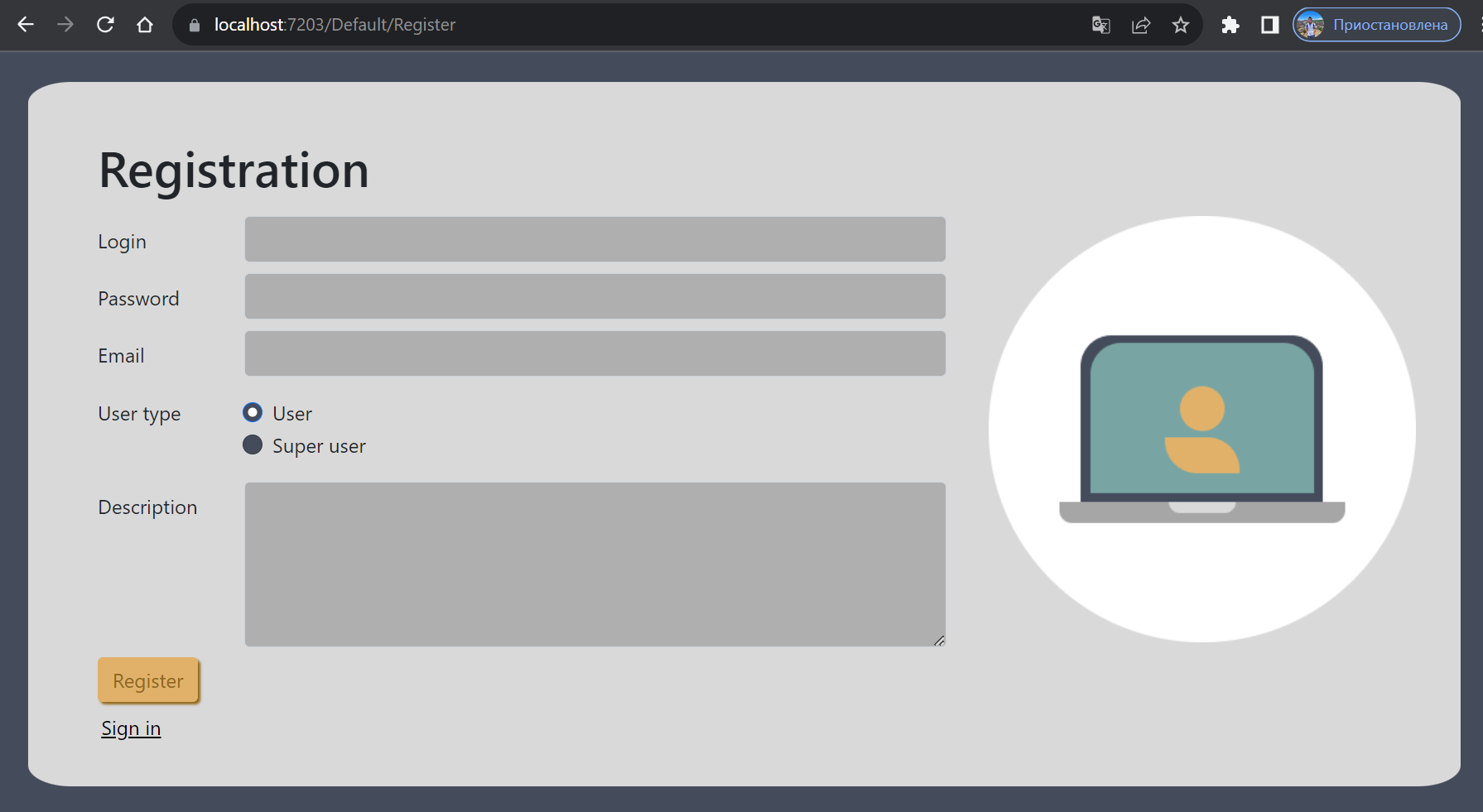


Рисунок 7.2 – Окно регистрации

Стоит обратить внимание на валидность вводимых данных. В поле ввода логина и пароля можно вводить любые строковые данные, длинной не более 100 символов. Однако, стоит учитывать, что логин является уникальным, следовательно, исключена ситуация, когда несколько пользователей имеют одинаковый логин. В поле email допускается вводить только строки, соответствующие шаблону для ввода почты. Тип пользователя может быть User и SuperUser. Обычный пользователь может создавать только локальные проекты для самого себя, а пользователь с расширенными правами может так же создавать проекты для других пользователей. Описание является обязательным полем только для пользователей с ролью SuperUser, так же эти пользователи должны дождаться подтверждения своей регистрации от администратора.

После успешной аутентификации пользователю открывается основной функционал, который специфичен его роли. Вначале рассмотрим интерфейс для стандартного пользователя, который представлен на рисунке 7.3.

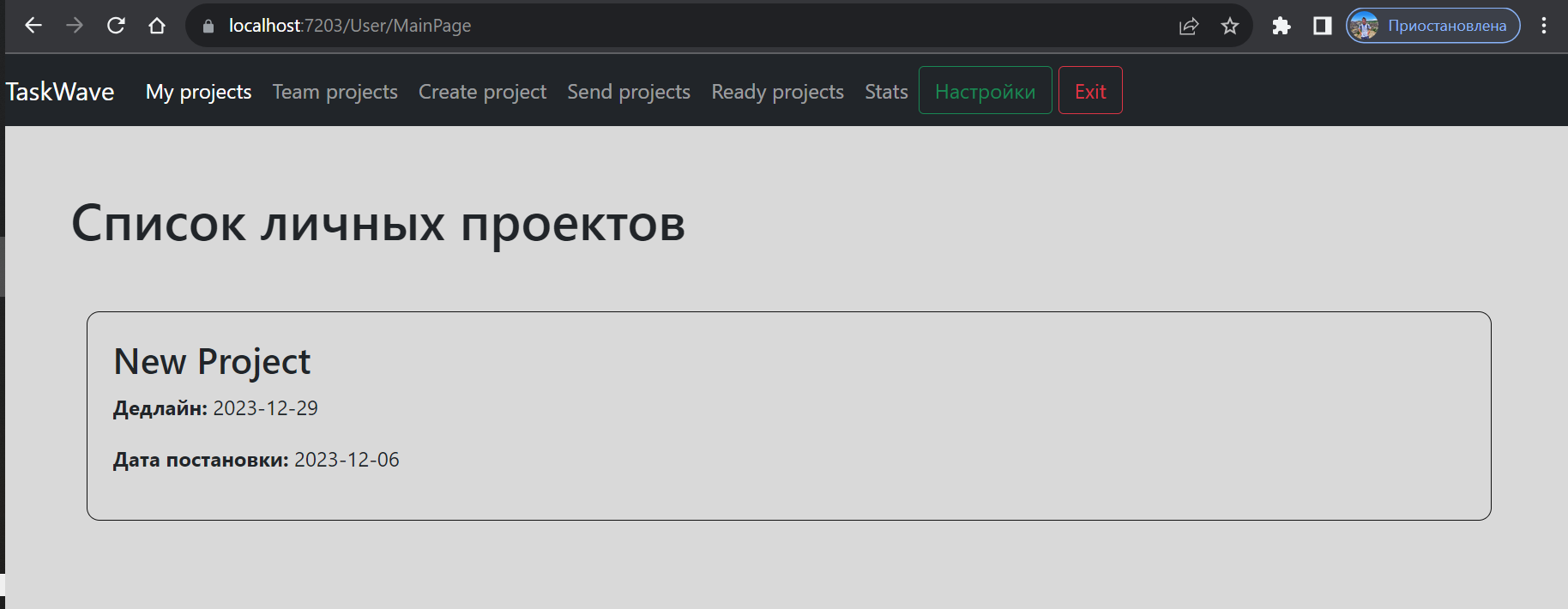


Рисунок 7.3 – Главная страница стандартного пользователя

Остановимся на пояснении всех пунктов меню:

* My project это список личных проектов;
* Team project это список командных проектов;
* Create project это форма для создания проекта (рисунок 7.4);
* Send project это список отправленных проектов на проверку;
* Ready project это список готовых проектов;
* Stats это статистика;
* Настройки позволяют внести изменения про информацию пользователя;
* Exit позволяет выйти из аккаунта.

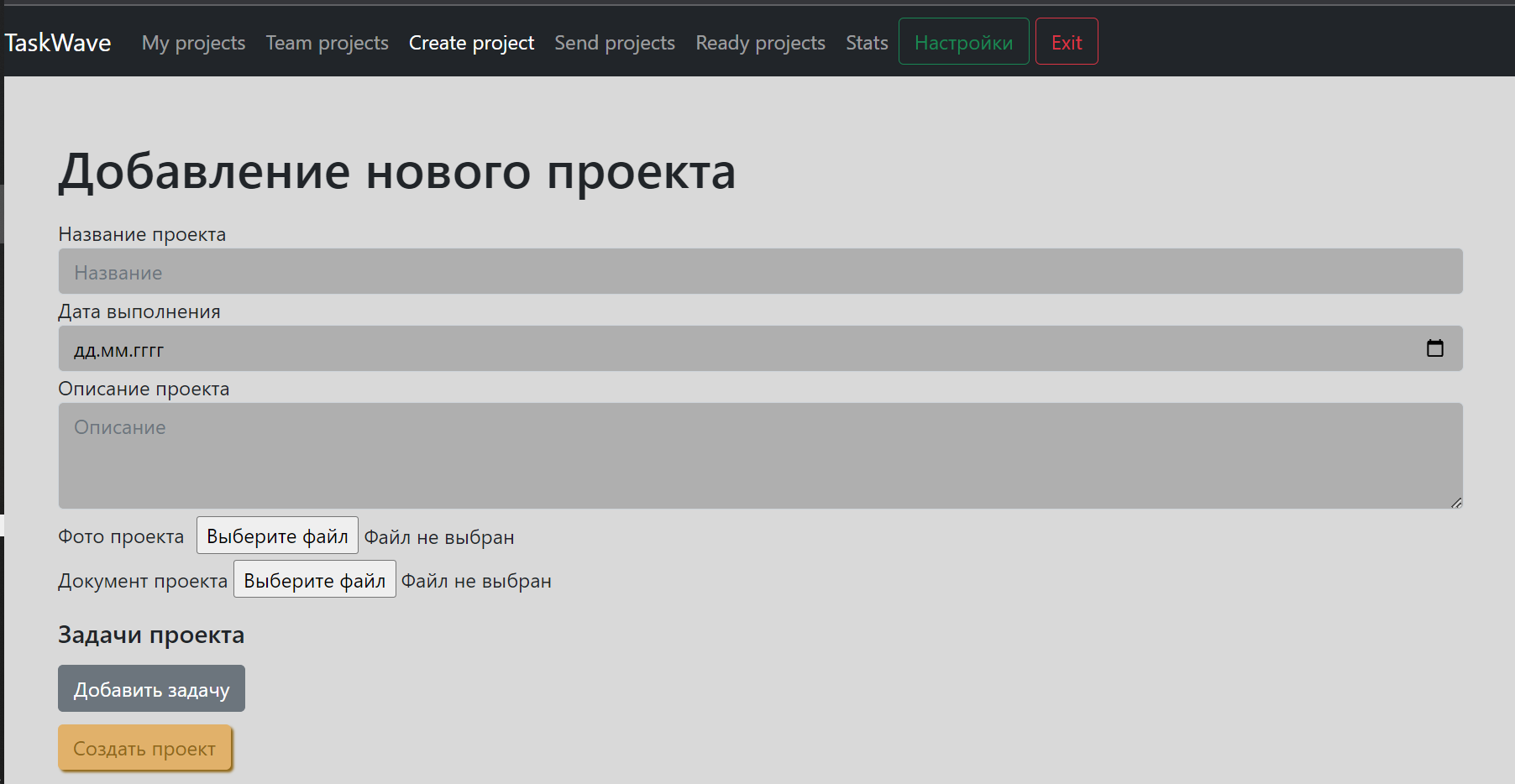


Рисунок 7.4 – Форма создания нового проекта

В поле названия проекта можем ввести любое строковое значение, ограниченное длинной 100 символов. Дату выполнения допускается назначать только для даты, которой ещё не было. Мы можем добавить файлы в качестве фото и документа. Допустимо добавления нескольких задач (связь один ко многим). После создания проекта он появляется в списке «моих проектов», как это было показано на рисунке 7.3.

Все панели с проектами (отправленные, выполненные и командные) выполнены аналогично со вкладкой личных проектов. За тем исключением, что нельзя вносить изменений в проект. В командных проектах можно лишь выполнить его (добавив при этом комментарии), в отправленных и выполненных просмотреть информацию. Так же важно учесть, что отправленные проекты в случае одобрения от создателя этого проекта, перемещаются в раздел выполненных не у исполнителя, а у того, кто назначал этот проект конечному пользователю. Если же проект не принимается, то он снова помещается в раздел командных проектов у пользователя.

Далее рассмотрим настройки аккаунта пользователя, вся информация демонстрируется на рисунке 7.5.

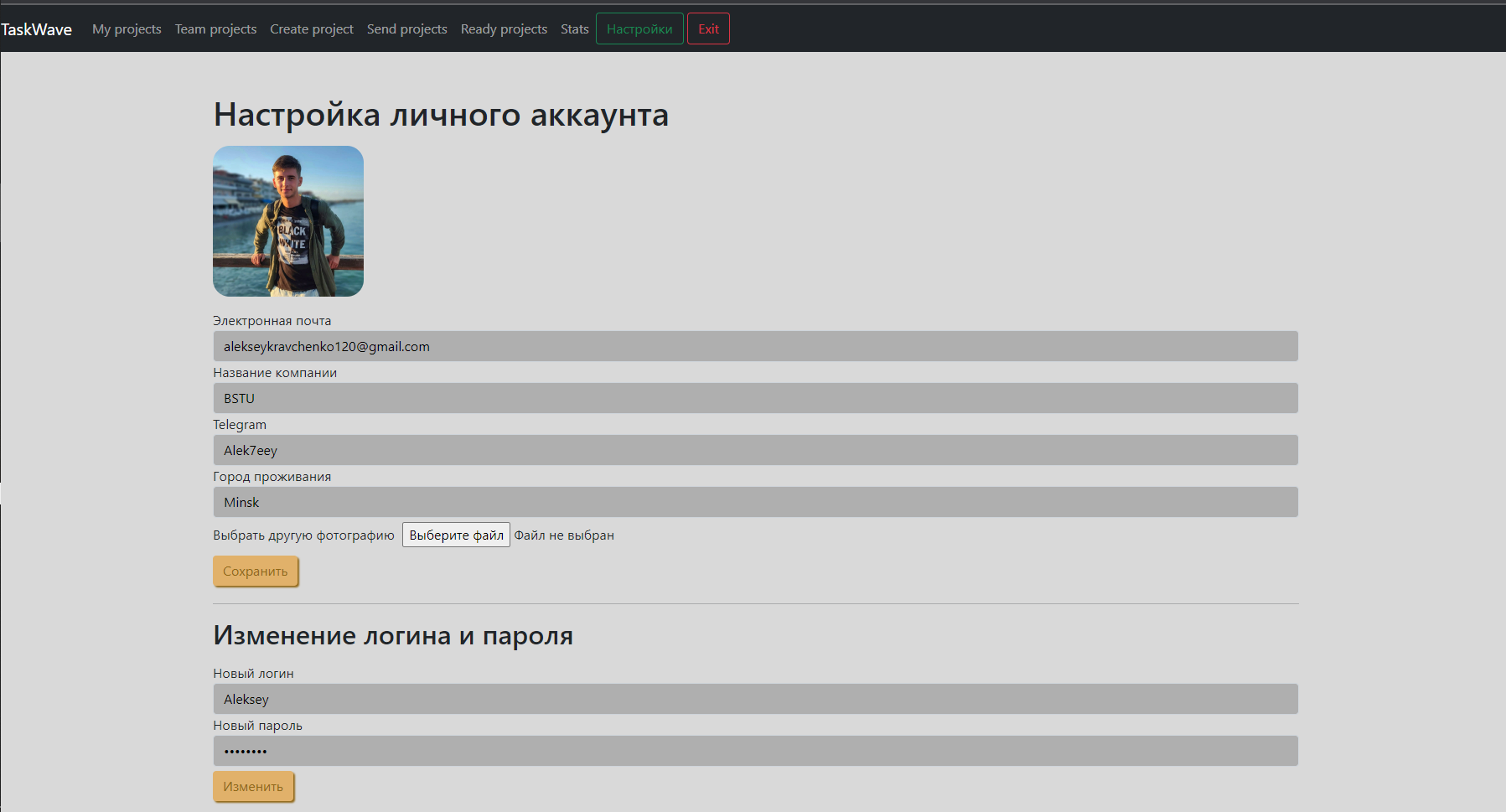


Рисунок 7.5 – Настройка личного аккаунта

Стоит обратить внимание, что на рисунке уже заполнена информация про пользователя и добавлена фотография. По умолчанию, после регистрации, вся эта информация пустая. Также в этом разделе можно изменить логин пользователя и его пароль. После нажатия на кнопки «Сохранить» или «Изменить», у нас обновляется информация и страница автоматически перезагружается.

Далее рассмотрим статистику. В приложении «TaskWave» она представляет из себя график, который показывает информацию про последние 6 месяцев (если аккаунт зарегистрирован не менее 6 месяцев назад) с количеством выполненных проектов каждый месяц. На рисунке 7.6 приводится такой пример для одного из пользователей. В данном примере можно заметить, что в ноябре пользователь выполнил 2 проекта, а в текущем месяце, т.е. в декабре, выполнена одна задача. Эти данные обрабатываются в БД и приходит непосредственно два массива, один из которых соответствует месяцам выполнения, а второй количеству выполненных задач.

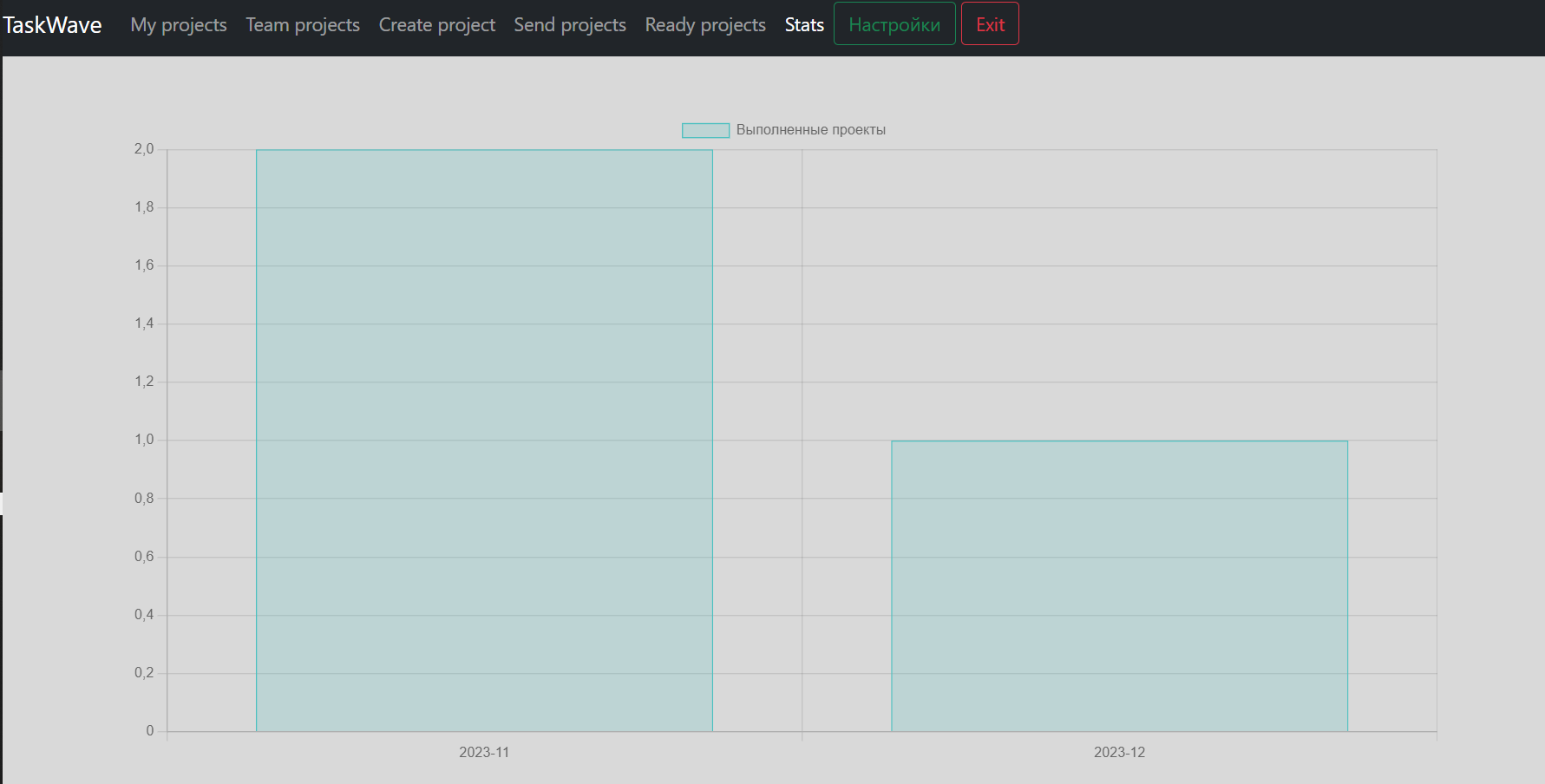


Рисунок 7.6 – Статистика

Далее перейдем к рассмотрению пользователя с расширенными правами. При аутентификации появляется аналогичный интерфейс со стандартным пользователем, но с дополнительными возможностями. На рисунке 7.7 демонстрируется страница.

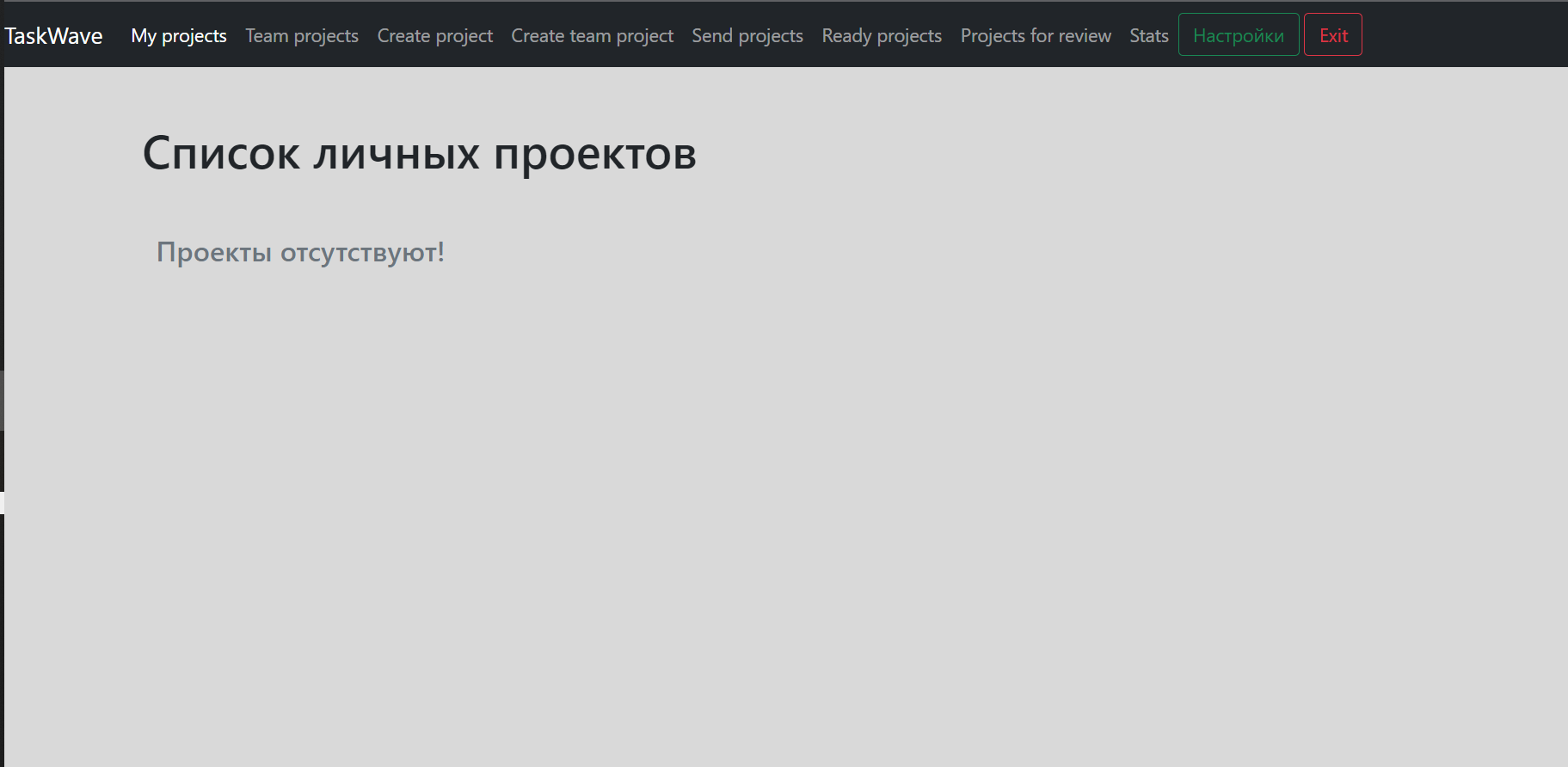


Рисунок 7.7 – Интерфейс пользователя с расширенными правами

Можно заметить, что добавлены пункты меню «Create team project» и «Projects for review». Создание командного проекта аналогично созданию личного проекта лишь за одним исключением, у пользователя с расширенными правами появляется возможность указать логин исполнителя. Список проектов на проверку представляет собой те проекты, которые пользователи отправили и пометили, как выполненные. Они содержат информацию про первоначальный проект и комментарии, которые пользователи оставили при отправки. Поэтому у пользователя с расширенными правами есть возможность принять или отклонить проект.

Далее рассмотрим возможности администратора. Данный пользователь регистрируется автоматически при развёртывании БД. По умолчанию он имеет логин и пароль «admin». На рисунке 7.8 демонстрируется интерфейс администратора.

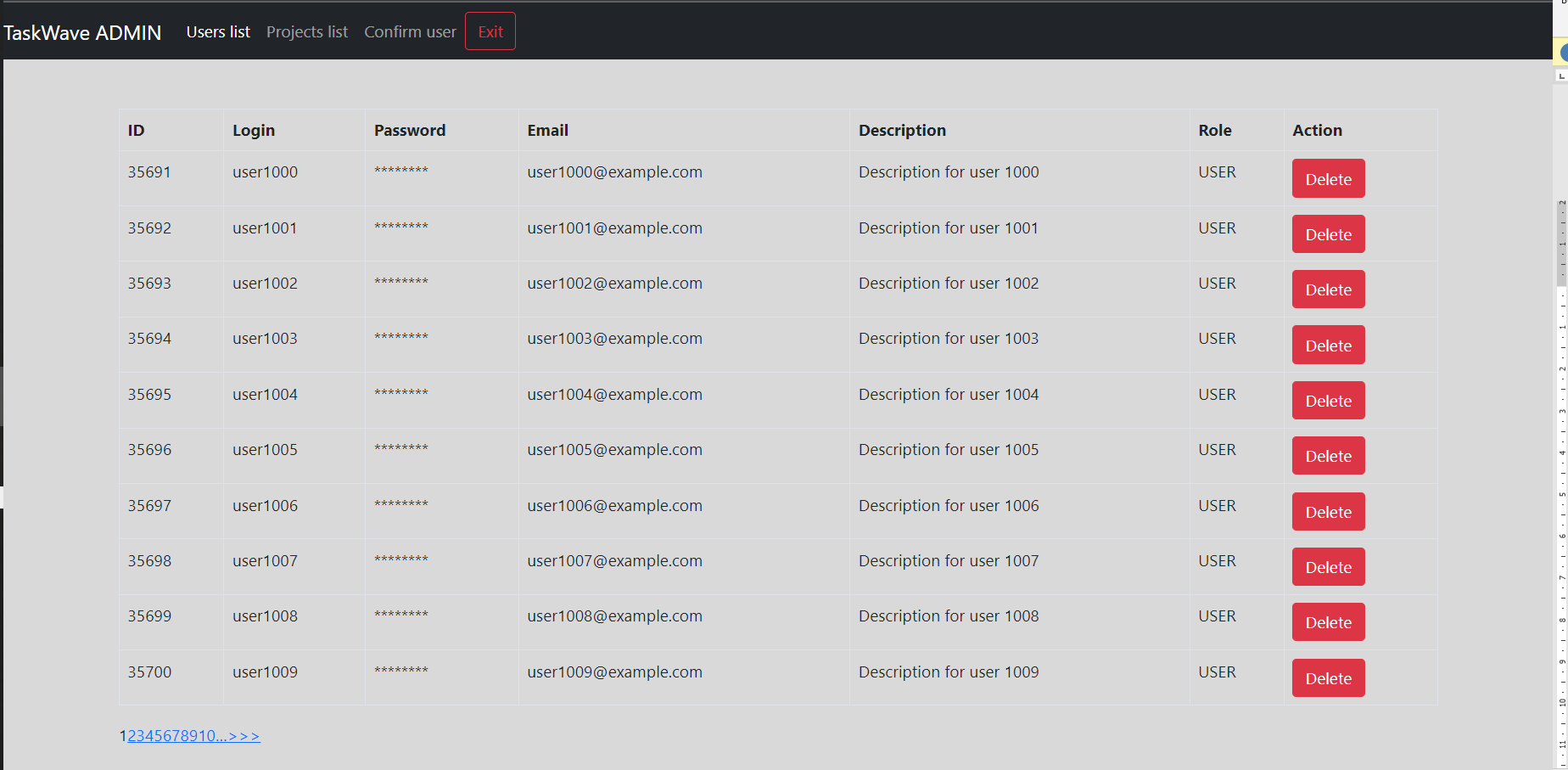


Рисунок 7.8 – Интерфейс администратора

Как сразу можно заметить, у администратора существует три вкладки меню. В users list содержится список пользователей с их описанием (пароль замаскирован) и возможностью удалить каждого из них. Реализована пагинация, которая позволяет выводить 10 пользователей на одной странице. На рисунке 7.9 демонстрируется содержимое Projects list.

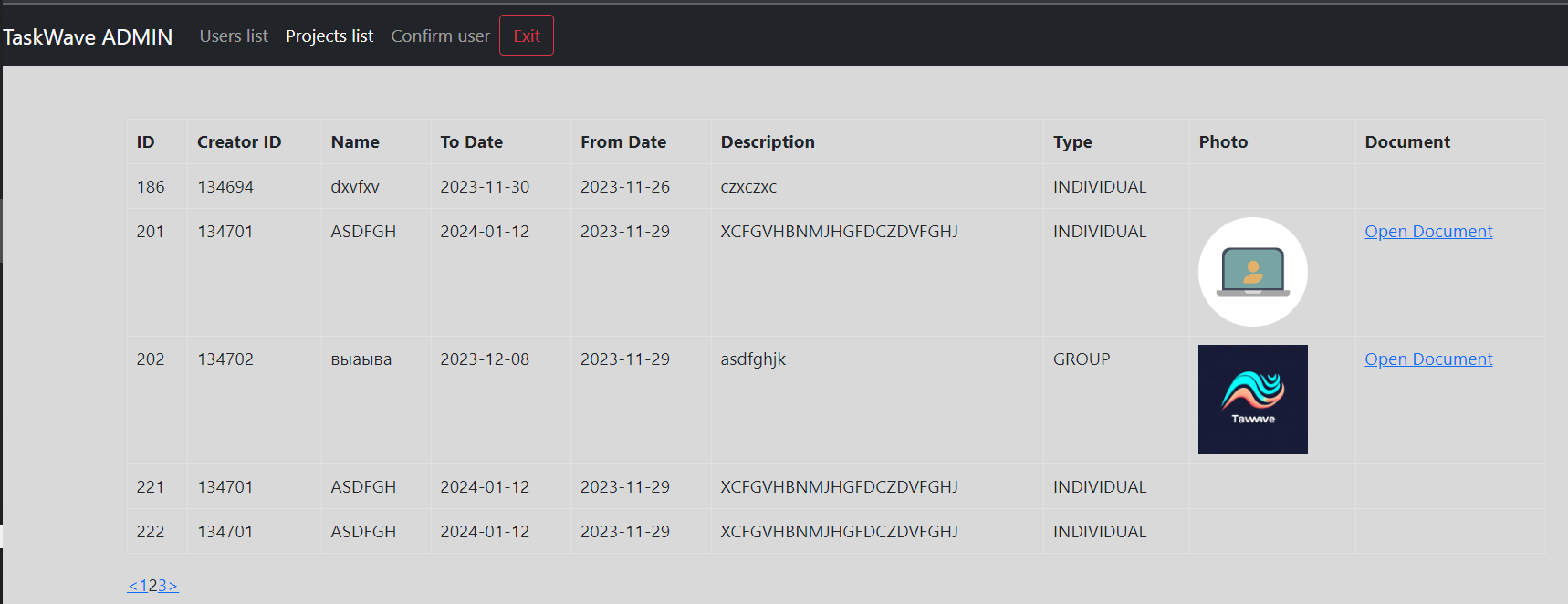


Рисунок 7.9 – Список проектов

В данном разделе возможен просмотр всех проектов с их описанием. Также реализована пагинация по 5 проектов на страницу.

Следующая вкладка «Confirm user» позволяет подтвердить (или нет) регистрацию пользователя с расширенными правами.

# ЗАКЛЮЧЕНИЕ

В ходе выполнения курсовой работы была спроектирована база данных для работы приложения «Task Wave» на базе СУБД Oracle 12c. Приложение было разработано с использованием технологии С# ASP.Net Core MVVM. Во время выполнения было изучено немало теоретического материала, а также просмотрено и разобрано большое количество уже готовых решений тех или иных задач. Были разработаны, описаны и применены на практике процедуры экспорта и импорта данных; разработаны функции шифрования данных пользователей; дополнительно изучена технология хранения мультимедийных данных, было описано руководство пользователя для приложения.

Функционально были выполнены следующие задачи:

* + - администратор управляет пользователями;
    - администратор подтверждает (или нет) регистрацию пользователей с расширенными правами;
    - стандартный пользователь может создавать личные проекты;
    - стандартный пользователь может управлять личными проектами;
    - стандартный пользователь может просматривать информацию про проекты;
    - стандартный пользователь может отправлять командные проекты на проверку;
    - стандартный пользователь может просматривать статистику;
    - стандартный пользователь может управлять личным аккаунтам;
    - стандартный пользователь может изменять пароль и логин;
    - стандартный пользователь может изменять личные данные;
    - Пользователь с расширенными правами дополнительно может создавать командные проекты и принимать (или нет) выполненные преокты от других пользователей;
    - были реализованы технологии шифрования на базе DBMS\_CRYPTO, а также технология хранения мультимедийных данных на базе DBMS\_LOB.

Данное программное средство имеет удобный и понятный пользователю интерфейс, корректно обрабатывает ошибочный ввод данных (например, ввод данных некорректного формата, попытка оставить поля пустыми), а также некоторые внутренние ошибки.

В соответствии с полученным результатом работы программы, можно сделать вывод, что разработанная программа работает верно, а требования технического задания выполнены в полном объёме.

# Список использованной литературы

[1] METANIT.COM Сайт о программировании [Электронный ресурс] / Режим доступа: https://metanit.com Дата доступа: 10.12.2019

[2] Oracle-dba.ru [Электронный ресурс] / Режим доступа: https://oracle-dba.ru Дата доступа 10.12.2023

[3] stackoverflow.com [Электронный ресурс] / Режим доступа: https://stackoverflow.com Дата доступа: 12.12.2023

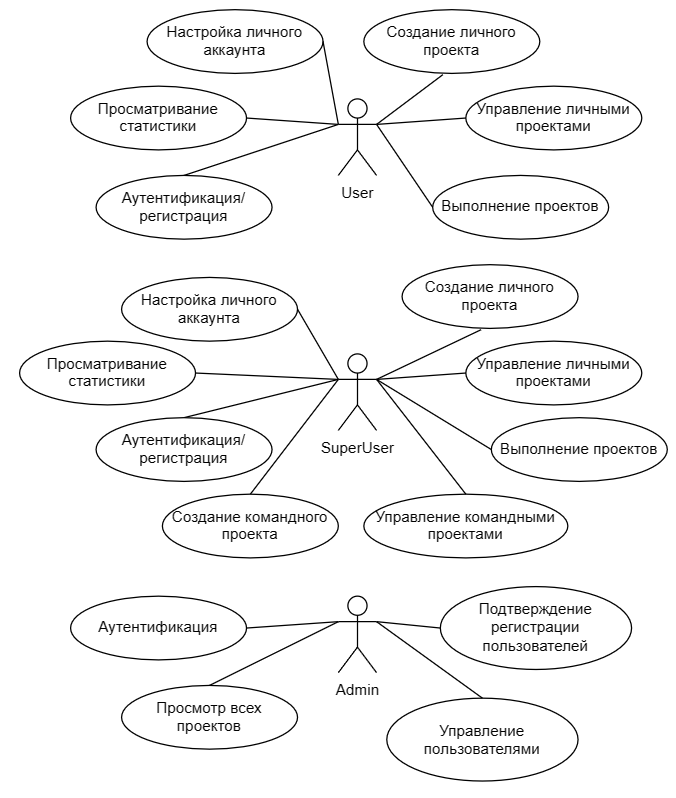
[4] Шифрование в PL/SQL: уроки программирования [Электронный ресурс] / Режим доступа: <https://oracle-patches.com/db/sql/3939-shifrovanie-v-pl-sql-uroki-programmirovaniya> Дата доступа: 13.12.2023

[5] DBMS\_SCHEDULER [Электронный ресурс] / Режим доступа: <https://docs.oracle.com/database/121/ARPLS/d_sched.htm#ARPLS72235> Дата доступа: 13.12.2023

[6] DBMS\_LOB [Электронный ресурс] / Режим доступа: <https://docs.oracle.com/database/121/ARPLS/d_lob.htm#ARPLS600> Дата доступа: 13.12.2023

[7] DBMS\_XMLDOM [Электронный ресурс] / Режим доступа: <https://docs.oracle.com/cd/B19306_01/appdev.102/b14258/d_xmldom.htm#i1076719>Дата доступа: 03.12.2023.

# Приложение A



# Приложение Б

-------------USERS-------------  
CREATE TABLE USERS (  
 id NUMBER(10) GENERATED AS IDENTITY (START WITH 1 INCREMENT BY 1),  
 login NVARCHAR2(60) NOT NULL,  
 password NVARCHAR2(100) NOT NULL,  
 gmail\_address NVARCHAR2(100) not null,  
 description NVARCHAR2(100) default null,  
 type NVARCHAR2(10) CHECK ( type IN ('user', 'admin', 'superUser')) NOT NULL,  
 is\_register NUMBER(1) CHECK ( is\_register IN (0, 1)) NOT NULL,  
 CONSTRAINT user\_pk PRIMARY KEY (id)  
);  
  
insert into USERS (login, password, gmail\_address, description, type, is\_register)  
values ('admin', *ENCRYPT\_PASSWORD*('admin'), 'alekseykravchenko120@gmail.com', 'This is admin', 'admin', 1);  
  
-------------USER\_PHOTO-------------  
CREATE TABLE USER\_PHOTO (  
 id NUMBER(10) GENERATED AS IDENTITY (START WITH 1 INCREMENT BY 1),  
 user\_id NUMBER(10) NOT NULL,  
 title NVARCHAR2(100) default null,  
 image BLOB DEFAULT *EMPTY\_BLOB*(),  
 CONSTRAINT user\_photo\_pk PRIMARY KEY (id),  
 CONSTRAINT user\_id\_fk FOREIGN KEY (user\_id) REFERENCES USERS(id)  
);  
drop table USER\_PHOTO;  
  
-------------USER\_DESCRIPTION-------------  
CREATE TABLE USER\_DESCRIPTION (  
 id NUMBER(10) GENERATED AS IDENTITY (START WITH 1 INCREMENT BY 1),  
 user\_id NUMBER(10) NOT NULL,  
 telegram\_URL NVARCHAR2(100) default null,  
 gmail\_address NVARCHAR2(100) default null,  
 company\_name NVARCHAR2(100) default null,  
 city\_name NVARCHAR2(100) default null,  
 CONSTRAINT user\_description\_pk PRIMARY KEY (id),  
 CONSTRAINT user\_description\_id\_fk FOREIGN KEY (user\_id) REFERENCES USERS(id)  
);

-------------PROJECT-------------  
drop table PROJECT;  
CREATE TABLE PROJECT(  
 id NUMBER(10) GENERATED AS IDENTITY (START WITH 1 INCREMENT BY 1),  
 name NVARCHAR2(100) NOT NULL,  
 from\_date DATE DEFAULT *CURRENT\_TIMESTAMP* NOT NULL,  
 to\_date DATE NOT NULL,  
 description NVARCHAR2(400),  
 type NVARCHAR2(12) CHECK (type in ('group', 'individual')) NOT NULL,  
 artist\_id NUMBER(10) default null, --id исполнителя  
 is\_ready NUMBER(1) CHECK ( is\_ready IN (0, 1)) NOT NULL,  
 is\_send NUMBER(1) CHECK ( is\_send IN (0, 1)) NOT NULL,  
 creator\_id NUMBER(10) NOT NULL,  
 CONSTRAINT project\_pk PRIMARY KEY (id),  
 CONSTRAINT project\_artist\_id\_fk FOREIGN KEY (artist\_id) REFERENCES USERS(id),  
 CONSTRAINT project\_creator\_id\_fk FOREIGN KEY (creator\_id) REFERENCES USERS(id)  
);  
  
-------------READY\_PROJECT-------------  
drop table READY\_PROJECT;  
CREATE TABLE READY\_PROJECT(  
 id NUMBER(10) GENERATED AS IDENTITY (START WITH 1 INCREMENT BY 1),  
 project\_id NUMBER(10) NOT NULL,  
 date\_complete DATE DEFAULT *CURRENT\_TIMESTAMP* NOT NULL,  
 performer\_user\_id NUMBER(10) NOT NULL,  
 CONSTRAINT ready\_project\_pk PRIMARY KEY (id),  
 CONSTRAINT project\_id\_fk FOREIGN KEY (project\_id) REFERENCES PROJECT(id),  
 CONSTRAINT performer\_user\_fk FOREIGN KEY (performer\_user\_id) REFERENCES USERS(id)  
);  
  
-------------SEND\_PROJECT-------------  
drop table SEND\_PROJECT;  
CREATE TABLE SEND\_PROJECT(  
 id NUMBER(10) GENERATED AS IDENTITY (START WITH 1 INCREMENT BY 1),  
 project\_id NUMBER(10) NOT NULL,  
 date\_send DATE DEFAULT *CURRENT\_TIMESTAMP* not null,  
 sender\_user\_id NUMBER(10) NOT NULL,  
 description NVARCHAR2(400),  
 CONSTRAINT send\_project\_pk PRIMARY KEY (id),  
 CONSTRAINT send\_project\_id\_fk FOREIGN KEY (project\_id) REFERENCES PROJECT(id),

CONSTRAINT sender\_user\_fk FOREIGN KEY (sender\_user\_id) REFERENCES USERS(id)  
);  
  
-------------PROJECT\_PHOTO-------------  
drop table PROJECT\_PHOTO;  
CREATE TABLE PROJECT\_PHOTO(  
 id NUMBER(10) GENERATED AS IDENTITY (START WITH 1 INCREMENT BY 1),  
 project\_id NUMBER(10) NOT NULL,  
 title NVARCHAR2(100) default null,  
 photo BLOB DEFAULT *EMPTY\_BLOB*(),  
 CONSTRAINT photo\_pk PRIMARY KEY (id),  
 CONSTRAINT project\_photo\_id\_fk FOREIGN KEY (project\_id) REFERENCES PROJECT(id)  
);  
  
-------------PROJECT\_DOCUMENT-------------  
drop table PROJECT\_DOCUMENT;  
CREATE TABLE PROJECT\_DOCUMENT(  
 id NUMBER(10) GENERATED AS IDENTITY (START WITH 1 INCREMENT BY 1),  
 project\_id NUMBER(10) NOT NULL,  
 title NVARCHAR2(100) default null,  
 type NVARCHAR2(10) default null,  
 data BLOB DEFAULT *EMPTY\_BLOB*(),  
 CONSTRAINT project\_document\_pk PRIMARY KEY (id),  
 CONSTRAINT project\_document\_id\_fk FOREIGN KEY (project\_id) REFERENCES PROJECT(id)  
);  
  
-------------TASK\_IN\_PROJECT-------------  
drop table TASK\_IN\_PROJECT;  
CREATE TABLE TASK\_IN\_PROJECT(  
 id NUMBER(10) GENERATED AS IDENTITY (START WITH 1 INCREMENT BY 1),  
 project\_id NUMBER(10) NOT NULL,  
 description NVARCHAR2(400),  
 CONSTRAINT task\_pk PRIMARY KEY (id),  
 CONSTRAINT project\_task\_id\_fk FOREIGN KEY (project\_id) REFERENCES PROJECT(id)  
);

# Приложение В

--PRIV Super User  
 GRANT CREATE SESSION, ALTER SESSION  
 TO SUPER\_USER\_TASK\_WAVE;  
 GRANT RESTRICTED SESSION TO SUPER\_USER\_TASK\_WAVE;  
 grant EXECUTE ON sys.dbms\_crypto to SUPER\_USER\_TASK\_WAVE;  
 GRANT EXECUTE on ADMIN\_TASK\_WAVE.*CreateStandartUser* to SUPER\_USER\_TASK\_WAVE;  
 GRANT EXECUTE on ADMIN\_TASK\_WAVE.*encrypt\_password* to SUPER\_USER\_TASK\_WAVE;  
 GRANT EXECUTE on ADMIN\_TASK\_WAVE.*getUser* to SUPER\_USER\_TASK\_WAVE;  
 GRANT EXECUTE on ADMIN\_TASK\_WAVE.*CreateProject* to SUPER\_USER\_TASK\_WAVE;  
 GRANT EXECUTE on ADMIN\_TASK\_WAVE.*CREATETASK* to SUPER\_USER\_TASK\_WAVE;  
 GRANT EXECUTE on ADMIN\_TASK\_WAVE.*getUserById* to SUPER\_USER\_TASK\_WAVE;  
 GRANT EXECUTE on ADMIN\_TASK\_WAVE.*GetOwnProjects* to SUPER\_USER\_TASK\_WAVE;  
 GRANT EXECUTE on ADMIN\_TASK\_WAVE.*GETTEAMPROJECTS* to SUPER\_USER\_TASK\_WAVE;  
 GRANT EXECUTE on ADMIN\_TASK\_WAVE.*GetSendProjects* to SUPER\_USER\_TASK\_WAVE;  
 GRANT EXECUTE on ADMIN\_TASK\_WAVE.*GetReadyProjects* to SUPER\_USER\_TASK\_WAVE;  
 GRANT EXECUTE on ADMIN\_TASK\_WAVE.*GetStat* to SUPER\_USER\_TASK\_WAVE;  
 GRANT EXECUTE on ADMIN\_TASK\_WAVE.*GetDescription* to SUPER\_USER\_TASK\_WAVE;  
 GRANT EXECUTE on ADMIN\_TASK\_WAVE.*CreateDefaultDescription* to SUPER\_USER\_TASK\_WAVE;  
 GRANT EXECUTE on ADMIN\_TASK\_WAVE.*ChangeUserInformation* to SUPER\_USER\_TASK\_WAVE;  
 GRANT EXECUTE on ADMIN\_TASK\_WAVE.*getUserByLogin* to SUPER\_USER\_TASK\_WAVE;  
 GRANT EXECUTE on ADMIN\_TASK\_WAVE.*ChangeDescriptionAccount* to SUPER\_USER\_TASK\_WAVE;  
 GRANT EXECUTE on ADMIN\_TASK\_WAVE.*GETCURRENTPROJECT* to SUPER\_USER\_TASK\_WAVE;  
 GRANT EXECUTE on ADMIN\_TASK\_WAVE.*GetCurrentTasks* to SUPER\_USER\_TASK\_WAVE;  
 GRANT EXECUTE on ADMIN\_TASK\_WAVE.*CompleteProject* to SUPER\_USER\_TASK\_WAVE;  
 GRANT EXECUTE on ADMIN\_TASK\_WAVE.*DeleteProject* to SUPER\_USER\_TASK\_WAVE;  
 GRANT EXECUTE on ADMIN\_TASK\_WAVE.*UpdateProject* to SUPER\_USER\_TASK\_WAVE;  
 GRANT EXECUTE on ADMIN\_TASK\_WAVE.*UPDATETASK* to SUPER\_USER\_TASK\_WAVE;

GRANT EXECUTE on ADMIN\_TASK\_WAVE.*GetCurrentTeamProject* to SUPER\_USER\_TASK\_WAVE;  
 GRANT EXECUTE on ADMIN\_TASK\_WAVE.*SendProject* to SUPER\_USER\_TASK\_WAVE;  
 GRANT EXECUTE on ADMIN\_TASK\_WAVE.*GetCurrentSendProject* to SUPER\_USER\_TASK\_WAVE;  
 GRANT EXECUTE on ADMIN\_TASK\_WAVE.*GETCURRENTREADYPROJECT* to SUPER\_USER\_TASK\_WAVE;  
 GRANT EXECUTE on ADMIN\_TASK\_WAVE.*CreateTeamProject* to SUPER\_USER\_TASK\_WAVE;  
 GRANT EXECUTE on ADMIN\_TASK\_WAVE.*GetReviewProjects* to SUPER\_USER\_TASK\_WAVE;  
 GRANT EXECUTE on ADMIN\_TASK\_WAVE.*AcceptProject* to SUPER\_USER\_TASK\_WAVE;  
 GRANT EXECUTE on ADMIN\_TASK\_WAVE.*RejectProject* to SUPER\_USER\_TASK\_WAVE;

--PRIV User  
 GRANT CREATE SESSION, ALTER SESSION  
 TO USER\_TASK\_WAVE;  
 GRANT RESTRICTED SESSION TO USER\_TASK\_WAVE;  
 grant EXECUTE ON sys.dbms\_crypto to USER\_TASK\_WAVE;  
 GRANT EXECUTE on ADMIN\_TASK\_WAVE.*CreateStandartUser* to USER\_TASK\_WAVE;  
 GRANT EXECUTE on ADMIN\_TASK\_WAVE.*encrypt\_password* to USER\_TASK\_WAVE;  
 GRANT EXECUTE on ADMIN\_TASK\_WAVE.*getUser* to USER\_TASK\_WAVE;  
 GRANT EXECUTE on ADMIN\_TASK\_WAVE.*CreateProject* to USER\_TASK\_WAVE;  
 GRANT EXECUTE on ADMIN\_TASK\_WAVE.*CREATETASK* to USER\_TASK\_WAVE;  
 GRANT EXECUTE on ADMIN\_TASK\_WAVE.*getUserById* to USER\_TASK\_WAVE;  
 GRANT EXECUTE on ADMIN\_TASK\_WAVE.*GetOwnProjects* to USER\_TASK\_WAVE;  
 GRANT EXECUTE on ADMIN\_TASK\_WAVE.*GETTEAMPROJECTS* to USER\_TASK\_WAVE;  
 GRANT EXECUTE on ADMIN\_TASK\_WAVE.*GetSendProjects* to USER\_TASK\_WAVE;  
 GRANT EXECUTE on ADMIN\_TASK\_WAVE.*GetReadyProjects* to USER\_TASK\_WAVE;  
 GRANT EXECUTE on ADMIN\_TASK\_WAVE.*GetStat* to USER\_TASK\_WAVE;  
 GRANT EXECUTE on ADMIN\_TASK\_WAVE.*GetDescription* to USER\_TASK\_WAVE;  
 GRANT EXECUTE on ADMIN\_TASK\_WAVE.*CreateDefaultDescription* to USER\_TASK\_WAVE;  
 GRANT EXECUTE on ADMIN\_TASK\_WAVE.*ChangeUserInformation* to USER\_TASK\_WAVE;  
 GRANT EXECUTE on ADMIN\_TASK\_WAVE.*getUserByLogin* to USER\_TASK\_WAVE;

GRANT EXECUTE on ADMIN\_TASK\_WAVE.*ChangeDescriptionAccount* to USER\_TASK\_WAVE;  
 GRANT EXECUTE on ADMIN\_TASK\_WAVE.*GETCURRENTPROJECT* to USER\_TASK\_WAVE;  
 GRANT EXECUTE on ADMIN\_TASK\_WAVE.*GetCurrentTasks* to USER\_TASK\_WAVE;  
 GRANT EXECUTE on ADMIN\_TASK\_WAVE.*CompleteProject* to USER\_TASK\_WAVE;  
 GRANT EXECUTE on ADMIN\_TASK\_WAVE.*DeleteProject* to USER\_TASK\_WAVE;  
 GRANT EXECUTE on ADMIN\_TASK\_WAVE.*UpdateProject* to USER\_TASK\_WAVE;  
 GRANT EXECUTE on ADMIN\_TASK\_WAVE.*UPDATETASK* to USER\_TASK\_WAVE;  
 GRANT EXECUTE on ADMIN\_TASK\_WAVE.*GetCurrentTeamProject* to USER\_TASK\_WAVE;  
 GRANT EXECUTE on ADMIN\_TASK\_WAVE.*SendProject* to USER\_TASK\_WAVE;  
 GRANT EXECUTE on ADMIN\_TASK\_WAVE.*GetCurrentSendProject* to USER\_TASK\_WAVE;  
 GRANT EXECUTE on ADMIN\_TASK\_WAVE.*GETCURRENTREADYPROJECT* to USER\_TASK\_WAVE;

--Priv admin  
 GRANT CREATE SESSION, ALTER SESSION, CREATE PLUGGABLE DATABASE  
 TO ADMIN\_TASK\_WAVE;  
  
 GRANT RESTRICTED SESSION TO ADMIN\_TASK\_WAVE;  
  
 GRANT CREATE TABLE, ALTER TABLE, DROP TABLE,  
 COMMENT TABLE, SELECT TABLE, INSERT TABLE,  
 UPDATE TABLE, DELETE TABLE,  
 FLASHBACK TABLE, ALTER INDEX, DROP INDEX,  
 CREATE INDEX, CREATE TRIGGER, ALTER TRIGGER,  
 DROP TRIGGER, EXECUTE PROCEDURE, CREATE PROCEDURE,  
 ALTER ANY PROCEDURE, DROP PROCEDURE, CREATE SEQUENCE,  
 ALTER SEQUENCE, DROP SEQUENCE, CREATE VIEW,  
 ALTER ANALYTIC VIEW,  
 DROP VIEW TO ADMIN\_TASK\_WAVE;  
 grant EXECUTE ON sys.dbms\_crypto to ADMIN\_TASK\_WAVE;

# Приложение Г

------------encrypt\_password--------------  
--drop function encrypt\_password;  
create or replace  
FUNCTION *encrypt\_password*(i\_password in users.password%type)  
 RETURN USERS.password%type  
 IS  
 my\_key VARCHAR2(2000) := '2023110909112023';  
 my\_val VARCHAR2(2000) := i\_password;  
 my\_mod NUMBER := sys.DBMS\_CRYPTO.encrypt\_aes128  
+ sys.DBMS\_CRYPTO.chain\_cbc  
+ sys.DBMS\_CRYPTO.pad\_pkcs5;  
 encrypted\_password RAW(2000);  
BEGIN  
 encrypted\_password := sys.DBMS\_CRYPTO.*encrypt*(utl\_i18n.*string\_to\_raw*(my\_val, 'AL32UTF8'), my\_mod,  
utl\_i18n.*string\_to\_raw*(my\_key, 'AL32UTF8'));  
 RETURN *RAWTOHEX*(encrypted\_password);  
END *encrypt\_password*;  
  
  
------------getUserById--------------  
CREATE OR REPLACE FUNCTION *getUserById*(  
 idUser in USERS.id%type  
)  
RETURN SYS\_REFCURSOR  
IS  
 user\_cursor SYS\_REFCURSOR;  
BEGIN  
 OPEN user\_cursor FOR  
 SELECT ID, LOGIN, PASSWORD, GMAIL\_ADDRESS, DESCRIPTION, TYPE  
 FROM users  
 WHERE USERS.ID = idUser  
 and  
 users.IS\_REGISTER = 1;  
  
 RETURN user\_cursor;  
END *getUserById*;  
  
------------getUser--------------  
CREATE OR REPLACE FUNCTION *getUser*(  
passwor IN users.password%TYPE,  
logi IN users.login%TYPE)  
RETURN SYS\_REFCURSOR  
IS

user\_cursor SYS\_REFCURSOR;  
BEGIN  
 OPEN user\_cursor FOR  
 SELECT ID, LOGIN, PASSWORD, GMAIL\_ADDRESS, DESCRIPTION, TYPE  
 FROM users  
 WHERE users.login = logi and  
 users.PASSWORD = *encrypt\_password*(passwor) and  
 users.IS\_REGISTER = 1;  
  
 RETURN user\_cursor;  
END *getUser*;  
  
------------getMyProjects--------------  
--drop function GetOwnProjects;  
CREATE OR REPLACE FUNCTION *GetOwnProjects*(  
userId IN PROJECT.CREATOR\_ID%TYPE  
)  
RETURN SYS\_REFCURSOR  
IS  
 project\_cursor SYS\_REFCURSOR;  
BEGIN  
 OPEN project\_cursor FOR  
 SELECT PROJECT.ID as ID, PROJECT.CREATOR\_ID as CREATOR\_ID,  
 PROJECT.NAME as NAME, PROJECT.TO\_DATE as TO\_DATE,  
 PROJECT.FROM\_DATE as FROM\_DATE,  
 PROJECT.DESCRIPTION as DESCRIPTION, PP.ID as idPh,  
 PP.TITLE as titlePh, PP.PHOTO as photoPh,  
 PD.ID as idDoc, PD.TITLE as titleDoc,  
 PD.TYPE as typeDoc, PD.DATA as dataDoc  
  
 FROM  
 PROJECT  
 LEFT JOIN PROJECT\_DOCUMENT PD ON PROJECT.ID = PD.PROJECT\_ID  
 LEFT JOIN PROJECT\_PHOTO PP ON PROJECT.ID = PP.PROJECT\_ID  
 WHERE  
 PROJECT.CREATOR\_ID = userId and  
 PROJECT.TYPE = 'individual'  
 and  
 PROJECT.IS\_SEND = 0  
 and  
 PROJECT.IS\_READY = 0;  
 RETURN project\_cursor;  
END *GetOwnProjects*;  
  
------------GetTeamProjects--------------  
CREATE OR REPLACE FUNCTION *GetTeamProjects*(  
userId IN PROJECT.ARTIST\_ID%TYPE

)  
RETURN SYS\_REFCURSOR  
IS  
 project\_cursor SYS\_REFCURSOR;  
BEGIN  
 OPEN project\_cursor FOR  
 SELECT PROJECT.ID as ID, PROJECT.CREATOR\_ID as CREATOR\_ID,  
 PROJECT.NAME as NAME, PROJECT.TO\_DATE as TO\_DATE,  
 PROJECT.FROM\_DATE as FROM\_DATE, PROJECT.ARTIST\_ID as ARTIST\_ID,  
 PROJECT.DESCRIPTION as DESCRIPTION,  
 PP.ID as idPh,  
 PP.TITLE as titlePh, PP.PHOTO as photoPh,  
 PD.ID as idDoc, PD.TITLE as titleDoc,  
 PD.TYPE as typeDoc, PD.DATA as dataDoc  
  
 FROM  
 PROJECT  
 LEFT JOIN PROJECT\_DOCUMENT PD ON PROJECT.ID = PD.PROJECT\_ID  
 LEFT JOIN PROJECT\_PHOTO PP ON PROJECT.ID = PP.PROJECT\_ID  
 WHERE  
 PROJECT.ARTIST\_ID = userId  
 and  
 PROJECT.TYPE = 'group'  
 and  
 PROJECT.IS\_SEND = 0  
 and  
 PROJECT.IS\_READY = 0;  
 RETURN project\_cursor;  
END *GetTeamProjects*;  
  
------------GetSendProjects--------------  
CREATE OR REPLACE FUNCTION *GetSendProjects*(  
userId IN PROJECT.ARTIST\_ID%TYPE  
)  
RETURN SYS\_REFCURSOR  
IS  
 project\_cursor SYS\_REFCURSOR;  
BEGIN  
 OPEN project\_cursor FOR  
 SELECT PROJECT.ID as ID, PROJECT.CREATOR\_ID as CREATOR\_ID,  
 PROJECT.NAME as NAME, PROJECT.TO\_DATE as TO\_DATE,  
 PROJECT.FROM\_DATE as FROM\_DATE, PROJECT.ARTIST\_ID as ARTIST\_ID,  
 PROJECT.DESCRIPTION as DESCRIPTION,  
 PP.ID as idPh,  
 PP.TITLE as titlePh, PP.PHOTO as photoPh,  
 PD.ID as idDoc, PD.TITLE as titleDoc,

PD.TYPE as typeDoc, PD.DATA as dataDoc,  
  
 SP.DESCRIPTION as SEND\_PROJECT\_DESCRIPTION,  
 SP.DATE\_SEND AS DATE\_SEND\_PROJECT,  
 SP.SENDER\_USER\_ID AS SENDER\_USER\_ID,  
 SP.ID AS ID\_SEND\_PROJECT  
  
 FROM  
 PROJECT  
 LEFT JOIN PROJECT\_DOCUMENT PD ON PROJECT.ID = PD.PROJECT\_ID  
 LEFT JOIN PROJECT\_PHOTO PP ON PROJECT.ID = PP.PROJECT\_ID  
 left join SEND\_PROJECT SP on PROJECT.ID = SP.PROJECT\_ID  
 WHERE  
 SP.SENDER\_USER\_ID = userId  
 and  
 PROJECT.TYPE = 'group'  
 and  
 PROJECT.IS\_SEND = '1';  
 RETURN project\_cursor;  
END *GetSendProjects*;  
  
------------GetReadyProjects--------------  
CREATE OR REPLACE FUNCTION *GetReadyProjects*(  
userId IN PROJECT.ARTIST\_ID%TYPE  
)  
RETURN SYS\_REFCURSOR  
IS  
 project\_cursor SYS\_REFCURSOR;  
BEGIN  
 OPEN project\_cursor FOR  
 SELECT PROJECT.ID as ID, PROJECT.CREATOR\_ID as CREATOR\_ID,  
 PROJECT.NAME as NAME, PROJECT.TO\_DATE as TO\_DATE,  
 PROJECT.FROM\_DATE as FROM\_DATE, PROJECT.ARTIST\_ID as ARTIST\_ID,  
 PROJECT.DESCRIPTION as DESCRIPTION,  
 PP.ID as idPh,  
 PP.TITLE as titlePh, PP.PHOTO as photoPh,  
 PD.ID as idDoc, PD.TITLE as titleDoc,  
 PD.TYPE as typeDoc, PD.DATA as dataDoc,  
  
 RP.DATE\_COMPLETE AS DATE\_COMPLETE\_PROJECT,  
 RP.performer\_user\_id AS perfomer\_USER\_ID,  
 RP.ID AS ID\_READY\_PROJECT  
  
 FROM  
 PROJECT  
 LEFT JOIN PROJECT\_DOCUMENT PD ON PROJECT.ID = PD.PROJECT\_ID  
 LEFT JOIN PROJECT\_PHOTO PP ON PROJECT.ID = PP.PROJECT\_ID

left join READY\_PROJECT RP on PROJECT.ID = RP.PROJECT\_ID  
 WHERE  
 (RP.performer\_user\_id = userId or PROJECT.CREATOR\_ID = userId)  
 and  
 PROJECT.IS\_READY = '1';  
 RETURN project\_cursor;  
END *GetReadyProjects*;  
  
------------GetStat--------------  
CREATE OR REPLACE FUNCTION *GetStat*(  
userId IN PROJECT.ARTIST\_ID%TYPE  
)  
RETURN SYS\_REFCURSOR  
IS  
 project\_cursor SYS\_REFCURSOR;  
BEGIN  
 OPEN project\_cursor FOR  
 SELECT  
 *TO\_CHAR*(RP.DATE\_COMPLETE, 'YYYY-MM') AS Month,  
 *COUNT*(\*) AS ProjectCount  
 FROM  
 READY\_PROJECT RP  
 WHERE  
 RP.performer\_user\_id = userId  
  
 GROUP BY *TO\_CHAR*(RP.DATE\_COMPLETE, 'YYYY-MM')  
 ORDER BY *TO\_CHAR*(RP.DATE\_COMPLETE, 'YYYY-MM') DESC;  
  
 RETURN project\_cursor;  
END *GetStat*;  
  
------------GetDescription--------------  
CREATE OR REPLACE FUNCTION *GetDescription*(  
 userId IN USERS.ID%TYPE  
)  
RETURN SYS\_REFCURSOR  
IS  
 user\_description\_cursor SYS\_REFCURSOR;  
BEGIN  
 OPEN user\_description\_cursor FOR  
 SELECT  
 ud.ID as ID,  
 ud.GMAIL\_ADDRESS AS GMAIL,  
 ud.TELEGRAM\_URL AS TELEGRAM,  
 ud.COMPANY\_NAME AS COMPANY,

ud.city\_name AS CITY,  
 u.LOGIN AS LOGIN,  
 CASE  
 WHEN u.PASSWORD IS NOT NULL THEN '\*\*\*\*\*\*\*\*'  
 ELSE NULL  
 END AS PASSWORD,  
  
 UP.TITLE AS TITLEPHOTO,  
 UP.IMAGE AS IMAGE  
 FROM  
 USERS U  
 LEFT JOIN USER\_DESCRIPTION ud on U.ID = ud.USER\_ID  
 LEFT JOIN USER\_PHOTO UP on U.ID = UP.USER\_ID  
  
 WHERE  
 U.ID = userId;  
  
 RETURN user\_description\_cursor;  
END *GetDescription*;  
  
------------getUserByLogin--------------  
CREATE OR REPLACE FUNCTION *getUserByLogin*(  
 loginUser in USERS.LOGIN%type  
)  
RETURN SYS\_REFCURSOR  
IS  
 user\_cursor SYS\_REFCURSOR;  
BEGIN  
 OPEN user\_cursor FOR  
 SELECT ID, LOGIN, PASSWORD, GMAIL\_ADDRESS, DESCRIPTION, TYPE  
 FROM users  
 WHERE USERS.LOGIN = loginUser  
 and  
 users.IS\_REGISTER = 1;  
  
 RETURN user\_cursor;  
END *getUserByLogin*;  
  
------------GetCurrentProject--------------  
--drop function GetCurrentProject;  
CREATE OR REPLACE FUNCTION *GETCURRENTPROJECT*(  
 projectId IN PROJECT.ID%TYPE  
)  
RETURN SYS\_REFCURSOR  
IS  
 project\_cursor SYS\_REFCURSOR;  
BEGIN  
 OPEN project\_cursor FOR

SELECT PROJECT.ID as ID, PROJECT.CREATOR\_ID as CREATOR\_ID,  
 PROJECT.NAME as NAME, PROJECT.TO\_DATE as TO\_DATE,  
 PROJECT.FROM\_DATE as FROM\_DATE,  
 PROJECT.DESCRIPTION as DESCRIPTION,  
  
 PP.ID as idPh,  
 PP.TITLE as titlePh, PP.PHOTO as photoPh,  
  
 PD.ID as idDoc, PD.TITLE as titleDoc,  
 PD.TYPE as typeDoc, PD.DATA as dataDoc  
 FROM  
 PROJECT  
 LEFT JOIN PROJECT\_DOCUMENT PD ON PROJECT.ID = PD.PROJECT\_ID  
 LEFT JOIN PROJECT\_PHOTO PP ON PROJECT.ID = PP.PROJECT\_ID  
 WHERE  
 PROJECT.ID = projectId and  
 PROJECT.TYPE = 'individual'  
 and  
 PROJECT.IS\_SEND = 0  
 and  
 PROJECT.IS\_READY = 0;  
 RETURN project\_cursor;  
END *GETCURRENTPROJECT*;  
  
------------GetCurrentTasks--------------  
--drop function GetCurrentTasks;  
CREATE OR REPLACE FUNCTION *GetCurrentTasks*(  
 projectId IN PROJECT.ID%TYPE  
)  
RETURN SYS\_REFCURSOR  
IS  
 project\_cursor SYS\_REFCURSOR;  
BEGIN  
 OPEN project\_cursor FOR  
 SELECT TIP.ID as idTask, TIP.DESCRIPTION as TaskDescription  
 FROM  
 TASK\_IN\_PROJECT TIP  
 WHERE  
 TIP.PROJECT\_ID = projectId;  
  
 RETURN project\_cursor;  
END *GetCurrentTasks*;  
  
------------GetCurrentTeamProject--------------  
--drop function GetCurrentTeamProject;  
CREATE OR REPLACE FUNCTION *GetCurrentTeamProject*(  
 projectId IN PROJECT.ID%TYPE  
)

RETURN SYS\_REFCURSOR  
IS  
 project\_cursor SYS\_REFCURSOR;  
BEGIN  
 OPEN project\_cursor FOR  
 SELECT PROJECT.ID as ID, PROJECT.CREATOR\_ID as CREATOR\_ID,  
 PROJECT.NAME as NAME, PROJECT.TO\_DATE as TO\_DATE,  
 PROJECT.FROM\_DATE as FROM\_DATE,  
 PROJECT.DESCRIPTION as DESCRIPTION,  
  
 PP.ID as idPh,  
 PP.TITLE as titlePh, PP.PHOTO as photoPh,  
  
 PD.ID as idDoc, PD.TITLE as titleDoc,  
 PD.TYPE as typeDoc, PD.DATA as dataDoc  
 FROM  
 PROJECT  
 LEFT JOIN PROJECT\_DOCUMENT PD ON PROJECT.ID = PD.PROJECT\_ID  
 LEFT JOIN PROJECT\_PHOTO PP ON PROJECT.ID = PP.PROJECT\_ID  
 WHERE  
 PROJECT.ID = projectId and  
 PROJECT.TYPE = 'group'  
 and  
 PROJECT.IS\_SEND = 0  
 and  
 PROJECT.IS\_READY = 0;  
 RETURN project\_cursor;  
END *GetCurrentTeamProject*;  
  
------------GetCurrentSendProject--------------  
--drop function GetCurrentSendProject;  
CREATE OR REPLACE FUNCTION *GetCurrentSendProject*(  
 projectId IN PROJECT.ID%TYPE  
)  
RETURN SYS\_REFCURSOR  
IS  
 project\_cursor SYS\_REFCURSOR;  
BEGIN  
OPEN project\_cursor FOR  
 SELECT PROJECT.ID as ID, PROJECT.CREATOR\_ID as CREATOR\_ID,  
 PROJECT.NAME as NAME, PROJECT.TO\_DATE as TO\_DATE,  
 PROJECT.FROM\_DATE as FROM\_DATE, PROJECT.ARTIST\_ID as ARTIST\_ID,  
 PROJECT.DESCRIPTION as DESCRIPTION,  
 PP.ID as idPh,  
 PP.TITLE as titlePh, PP.PHOTO as photoPh,  
 PD.ID as idDoc, PD.TITLE as titleDoc,

PD.TYPE as typeDoc, PD.DATA as dataDoc,  
 SP.DESCRIPTION as SEND\_PROJECT\_DESCRIPTION,  
 SP.DATE\_SEND AS DATE\_SEND\_PROJECT,  
 SP.SENDER\_USER\_ID AS SENDER\_USER\_ID,  
 SP.ID AS ID\_SEND\_PROJECT  
  
 FROM  
 PROJECT  
 LEFT JOIN PROJECT\_DOCUMENT PD ON PROJECT.ID = PD.PROJECT\_ID  
 LEFT JOIN PROJECT\_PHOTO PP ON PROJECT.ID = PP.PROJECT\_ID  
 left join SEND\_PROJECT SP on PROJECT.ID = SP.PROJECT\_ID  
 WHERE  
 SP.PROJECT\_ID = projectId  
 and  
 PROJECT.TYPE = 'group'  
 and  
 PROJECT.IS\_SEND = '1';  
 RETURN project\_cursor;  
END *GetCurrentSendProject*;  
  
------------GetCurrentReadyProject--------------  
--drop function GetCurrentReadyProject;  
CREATE OR REPLACE FUNCTION *GETCURRENTREADYPROJECT*(  
 projectId IN PROJECT.ID%TYPE  
)  
RETURN SYS\_REFCURSOR  
IS  
 project\_cursor SYS\_REFCURSOR;  
BEGIN  
OPEN project\_cursor FOR  
 SELECT PROJECT.ID as ID, PROJECT.CREATOR\_ID as CREATOR\_ID,  
 PROJECT.NAME as NAME, PROJECT.TO\_DATE as TO\_DATE,  
 PROJECT.FROM\_DATE as FROM\_DATE, PROJECT.ARTIST\_ID as ARTIST\_ID,  
 PROJECT.DESCRIPTION as DESCRIPTION,  
 PROJECT.IS\_READY AS ISREADY,  
 PROJECT.TYPE AS TYPE\_PROJECT,  
  
 PP.ID as idPh,  
 PP.TITLE as titlePh, PP.PHOTO as photoPh,  
 PD.ID as idDoc, PD.TITLE as titleDoc,  
 PD.TYPE as typeDoc, PD.DATA as dataDoc,  
  
 RP.DATE\_COMPLETE AS DATE\_COMPLETE\_PROJECT,  
 RP.PERFORMER\_USER\_ID AS COMPLETE\_USER\_ID,  
 RP.ID AS ID\_READY\_PROJECT  
  
 FROM

PROJECT  
 LEFT JOIN PROJECT\_DOCUMENT PD ON PROJECT.ID = PD.PROJECT\_ID  
 LEFT JOIN PROJECT\_PHOTO PP ON PROJECT.ID = PP.PROJECT\_ID  
 left join READY\_PROJECT RP on PROJECT.ID = RP.PROJECT\_ID  
 WHERE  
 PROJECT.ID = projectId  
 and  
 PROJECT.IS\_READY = '1';  
 RETURN project\_cursor;  
END *GetCurrentReadyProject*;  
  
------------getUnaffectedUser--------------  
CREATE OR REPLACE FUNCTION *getUnaffectedUser*(  
passwor IN users.password%TYPE,  
logi IN users.login%TYPE)  
RETURN SYS\_REFCURSOR  
IS  
 user\_cursor SYS\_REFCURSOR;  
BEGIN  
 OPEN user\_cursor FOR  
 SELECT ID, LOGIN, PASSWORD, GMAIL\_ADDRESS, DESCRIPTION, TYPE  
 FROM users  
 WHERE users.login = logi and  
 users.PASSWORD = *encrypt\_password*(passwor);  
  
 RETURN user\_cursor;  
END *getUnaffectedUser*;  
  
------------GetReviewProjects--------------  
CREATE OR REPLACE FUNCTION *GetReviewProjects*(  
userId IN PROJECT.ARTIST\_ID%TYPE  
)  
RETURN SYS\_REFCURSOR  
IS  
 project\_cursor SYS\_REFCURSOR;  
BEGIN  
 OPEN project\_cursor FOR  
 SELECT PROJECT.ID as ID, PROJECT.CREATOR\_ID as CREATOR\_ID,  
 PROJECT.NAME as NAME, PROJECT.TO\_DATE as TO\_DATE,  
 PROJECT.FROM\_DATE as FROM\_DATE, PROJECT.ARTIST\_ID as ARTIST\_ID,  
 PROJECT.DESCRIPTION as DESCRIPTION,  
 PP.ID as idPh,  
 PP.TITLE as titlePh, PP.PHOTO as photoPh,  
 PD.ID as idDoc, PD.TITLE as titleDoc,  
 PD.TYPE as typeDoc, PD.DATA as dataDoc,  
  
 SP.DESCRIPTION as SEND\_PROJECT\_DESCRIPTION,

SP.DATE\_SEND AS DATE\_SEND\_PROJECT,  
 SP.SENDER\_USER\_ID AS SENDER\_USER\_ID,  
 SP.ID AS ID\_SEND\_PROJECT  
  
 FROM  
 PROJECT  
 LEFT JOIN PROJECT\_DOCUMENT PD ON PROJECT.ID = PD.PROJECT\_ID  
 LEFT JOIN PROJECT\_PHOTO PP ON PROJECT.ID = PP.PROJECT\_ID  
 left join SEND\_PROJECT SP on PROJECT.ID = SP.PROJECT\_ID  
 WHERE  
 PROJECT.CREATOR\_ID = userId  
 and  
 PROJECT.TYPE = 'group'  
 and  
 PROJECT.IS\_SEND = '1';  
 RETURN project\_cursor;  
END *GetReviewProjects*;  
  
------------GetAllUsers--------------  
CREATE OR REPLACE FUNCTION *GetAllUsers*RETURN SYS\_REFCURSOR  
IS  
 user\_cursor SYS\_REFCURSOR;  
BEGIN  
 OPEN user\_cursor FOR  
 SELECT ID, LOGIN,  
 CASE WHEN users.PASSWORD IS NOT NULL THEN '\*\*\*\*\*\*\*\*'  
 ELSE NULL  
 END AS PASSWORD,  
 GMAIL\_ADDRESS, DESCRIPTION, TYPE  
 FROM users;  
  
 RETURN user\_cursor;  
END *GetAllUsers*;  
  
------------GetAllProjects--------------  
CREATE OR REPLACE FUNCTION *GetAllProjects*RETURN SYS\_REFCURSOR  
IS  
 project\_cursor SYS\_REFCURSOR;  
BEGIN  
 OPEN project\_cursor FOR  
 SELECT PROJECT.ID as ID, PROJECT.CREATOR\_ID as CREATOR\_ID,  
 PROJECT.NAME as NAME, PROJECT.TO\_DATE as TO\_DATE,  
 PROJECT.FROM\_DATE as FROM\_DATE, PROJECT.ARTIST\_ID as ARTIST\_ID,  
 PROJECT.DESCRIPTION as DESCRIPTION,  
 PROJECT.TYPE AS TYPE,  
 PP.ID as idPh,  
 PP.TITLE as titlePh, PP.PHOTO as photoPh,

PD.ID as idDoc, PD.TITLE as titleDoc,  
 PD.TYPE as typeDoc, PD.DATA as dataDoc  
 FROM  
 PROJECT  
 LEFT JOIN PROJECT\_DOCUMENT PD ON PROJECT.ID = PD.PROJECT\_ID  
 LEFT JOIN PROJECT\_PHOTO PP ON PROJECT.ID = PP.PROJECT\_ID;  
  
 RETURN project\_cursor;  
END *GetAllProjects*;  
  
------------GetAllUnaffectedUser--------------  
CREATE OR REPLACE FUNCTION *GetAllUnaffectedUser*RETURN SYS\_REFCURSOR  
IS  
 user\_cursor SYS\_REFCURSOR;  
BEGIN  
 OPEN user\_cursor FOR  
 SELECT ID, LOGIN,  
 CASE WHEN users.PASSWORD IS NOT NULL THEN '\*\*\*\*\*\*\*\*'  
 ELSE NULL  
 END AS PASSWORD,  
 GMAIL\_ADDRESS, DESCRIPTION, TYPE  
 FROM users  
 WHERE IS\_REGISTER = 0;  
  
 RETURN user\_cursor;  
END *GetAllUnaffectedUser*;

# Приложение Д

-------------CreateStandardUser-------------  
--drop procedure CreateStandartUser;  
CREATE OR REPLACE PROCEDURE *CreateStandartUser*(  
 user\_login IN USERS.login%TYPE,  
 user\_password IN USERS.password%TYPE,  
 user\_email IN USERS.gmail\_address%TYPE,  
 user\_description IN USERS.description%TYPE := null  
)  
IS  
 cnt NUMBER;  
 encrypted\_password\_result USERS.password%type;  
 user\_type USERS.type%TYPE := 'user';  
 user\_is\_register USERS.is\_register%TYPE := 1;  
  
 BEGIN  
 SELECT *COUNT*(\*) INTO cnt FROM USERS where *UPPER*(LOGIN) = *UPPER*(user\_login);  
 IF(cnt=0) then  
 encrypted\_password\_result := *encrypt\_password*(user\_password);  
 INSERT INTO USERS(login, password, GMAIL\_ADDRESS,DESCRIPTION, type, is\_register)  
 values(  
 user\_login, encrypted\_password\_result,user\_email,user\_description, user\_type, user\_is\_register  
 );  
 COMMIT;  
 ELSE  
 *RAISE\_APPLICATION\_ERROR*(-20001, 'This person already exists');  
 END IF;  
end *CreateStandartUser*;  
  
-------------CreateSuperUser-------------  
drop procedure *CreateSuperUser*;  
CREATE OR REPLACE PROCEDURE *CreateSuperUser*(  
 user\_login IN USERS.login%TYPE,  
 user\_password IN USERS.password%TYPE,  
 user\_email IN USERS.gmail\_address%TYPE,  
 user\_description IN USERS.description%TYPE := null  
)  
IS  
 cnt NUMBER;  
 encrypted\_password\_result USERS.password%type;  
 user\_type USERS.type%TYPE := 'superUser';  
 user\_is\_register USERS.is\_register%TYPE := 0;

current\_id USERS.id%TYPE;  
  
 BEGIN  
 SELECT *COUNT*(\*) INTO cnt FROM USERS where *UPPER*(LOGIN) = *UPPER*(user\_login);  
 IF(cnt=0) then  
 encrypted\_password\_result := *encrypt\_password*(user\_password);  
 INSERT INTO USERS(login, password, GMAIL\_ADDRESS,DESCRIPTION, type, is\_register)  
 values(  
 user\_login, encrypted\_password\_result,user\_email,user\_description, user\_type, user\_is\_register  
 );  
 COMMIT;  
 ELSE  
 *RAISE\_APPLICATION\_ERROR*(-20001, 'This person already exists');  
 END IF;  
 end *CreateSuperUser*;  
  
-------------CreateProject-------------  
drop procedure *CreateProject*;  
CREATE OR REPLACE PROCEDURE *CreateProject*(  
 creatorid IN PROJECT.creator\_id%TYPE,  
 project\_name IN PROJECT.name%TYPE,  
 toDate IN PROJECT.to\_date%TYPE,  
 project\_description IN PROJECT.description%TYPE,  
 photo\_name IN PROJECT\_PHOTO.title%TYPE DEFAULT NULL,  
 photo\_data IN PROJECT\_PHOTO.photo%TYPE DEFAULT NULL,  
 document\_name IN PROJECT\_DOCUMENT.title%TYPE DEFAULT NULL,  
 document\_type IN PROJECT\_DOCUMENT.type%TYPE DEFAULT NULL,  
 document\_data IN PROJECT\_DOCUMENT.data%TYPE DEFAULT NULL,  
 current\_id OUT PROJECT.id%TYPE  
)  
IS  
 cnt NUMBER;  
  
BEGIN  
 -- Insert data into the PROJECT table  
 INSERT INTO PROJECT (  
 name,  
 to\_date,  
 description,  
 creator\_id,  
 type,  
 is\_ready,  
 is\_send

) VALUES (  
 project\_name,  
 toDate,  
 project\_description,  
 creatorid,  
 'individual',  
 0,  
 0  
 ) RETURNING id INTO cnt;  
  
 -- Insert data into the PROJECT\_PHOTO table if values are provided  
 IF photo\_name IS NOT NULL AND photo\_data IS NOT NULL THEN  
 INSERT INTO PROJECT\_PHOTO (  
 project\_id,  
 title,  
 photo  
 ) VALUES (  
 cnt,  
 photo\_name,  
 photo\_data  
 );  
 END IF;  
  
 -- Insert data into the PROJECT\_DOCUMENT table if values are provided  
 IF document\_name IS NOT NULL AND document\_type IS NOT NULL AND document\_data IS NOT NULL THEN  
 INSERT INTO PROJECT\_DOCUMENT (  
 project\_id,  
 title,  
 type,  
 data  
 ) VALUES (  
 cnt,  
 document\_name,  
 document\_type,  
 document\_data  
 );  
 END IF;  
  
 current\_id := cnt;  
  
 COMMIT;  
END *CreateProject*;  
  
-------------CreateTask-------------  
drop procedure *CreateTask*;  
CREATE OR REPLACE PROCEDURE *CreateTask*(

project\_id IN TASK\_IN\_PROJECT.PROJECT\_ID%TYPE,  
 description IN TASK\_IN\_PROJECT.description%TYPE  
)  
IS  
 BEGIN  
 insert into TASK\_IN\_PROJECT (PROJECT\_ID, DESCRIPTION)  
 VALUES (PROJECT\_ID, DESCRIPTION);  
 COMMIT;  
END *CreateTask*;  
  
-------------CreateDefaultDescription-------------  
drop procedure *CreateDefaultDescription*;  
CREATE OR REPLACE PROCEDURE *CreateDefaultDescription*(  
 user\_id IN USERS.ID%TYPE  
)  
IS  
 BEGIN  
 insert into USER\_DESCRIPTION (user\_id, telegram\_url, gmail\_address, company\_name, city\_name)  
 VALUES (USER\_ID, '','','','');  
 insert into USER\_PHOTO (user\_id, TITLE, IMAGE)  
 VALUES (USER\_ID, '','');  
 COMMIT;  
  
END *CreateDefaultDescription*;  
  
-------------ChangeUserInformation-------------  
drop procedure *ChangeUserInformation*;  
CREATE OR REPLACE PROCEDURE *ChangeUserInformation*(  
 user\_login IN USERS.LOGIN%TYPE,  
 user\_password IN USERS.PASSWORD%TYPE,  
 user\_id IN USERS.ID%TYPE  
)  
IS  
 current\_login USERS.LOGIN%TYPE;  
 current\_password USERS.PASSWORD%TYPE;  
  
BEGIN  
 -- Получаем текущие значения логина и пароля для пользователя с указанным ID  
 SELECT login, password INTO current\_login, current\_password  
 FROM USERS U  
 WHERE U.ID = user\_id;  
  
 -- Проверяем, совпадают ли переданные значения с текущими  
 IF user\_password = '\*\*\*\*\*\*\*\*' AND user\_login != current\_login then  
 UPDATE USERS

SET LOGIN = user\_login  
  
 WHERE ID = user\_id;  
  
 COMMIT; -- Фиксация изменений в БД  
  
 elsif user\_login != current\_login OR user\_password != current\_password THEN  
 -- Если не совпадают, обновляем значения в БД  
 UPDATE USERS  
 SET LOGIN = user\_login,  
 PASSWORD = *encrypt\_password*(user\_password)  
 WHERE ID = user\_id;  
  
 COMMIT; -- Фиксация изменений в БД  
  
 END IF;  
  
END *ChangeUserInformation*;  
  
-------------ChangeDescriptionAccount-------------  
drop procedure *ChangeDescriptionAccount*;  
CREATE OR REPLACE PROCEDURE *ChangeDescriptionAccount*(  
 userid IN USERS.ID%TYPE,  
 user\_email IN USER\_DESCRIPTION.GMAIL\_ADDRESS%TYPE,  
 user\_company IN USER\_DESCRIPTION.COMPANY\_NAME%TYPE,  
 user\_telegram IN USER\_DESCRIPTION.TELEGRAM\_URL%TYPE,  
 user\_city IN USER\_DESCRIPTION.CITY\_NAME%TYPE,  
 user\_image IN USER\_PHOTO.IMAGE%TYPE,  
 user\_title IN USER\_PHOTO.TITLE%TYPE  
)  
IS  
 user\_description\_count INTEGER;  
 user\_photo\_count INTEGER;  
  
BEGIN  
 -- Проверяем существование записи в USER\_DESCRIPTION для заданного user\_id  
 SELECT *COUNT*(\*)  
 INTO user\_description\_count  
 FROM USER\_DESCRIPTION  
 WHERE USER\_ID = userid;  
  
 -- Если запись уже существует, обновляем значения  
 IF user\_description\_count > 0 THEN  
 UPDATE USER\_DESCRIPTION  
 SET  
 TELEGRAM\_URL = user\_telegram,  
 GMAIL\_ADDRESS = user\_email,

COMPANY\_NAME = user\_company,  
 CITY\_NAME = user\_city  
 WHERE USER\_ID = userid;  
 ELSE  
 -- Вставляем новую запись в USER\_DESCRIPTION  
 INSERT INTO USER\_DESCRIPTION(USER\_ID, TELEGRAM\_URL, GMAIL\_ADDRESS, COMPANY\_NAME, CITY\_NAME)  
 VALUES (userid, user\_telegram, user\_email, user\_company, user\_city);  
 END IF;  
  
 -- Проверяем существование записи в USER\_PHOTO для заданного user\_id  
 SELECT *COUNT*(\*)  
 INTO user\_photo\_count  
 FROM USER\_PHOTO  
 WHERE USER\_ID = userid;  
  
 -- Если запись уже существует и передано значение для user\_title, обновляем значения  
 IF user\_photo\_count > 0 AND user\_title IS NOT NULL THEN  
 UPDATE USER\_PHOTO  
 SET  
 TITLE = user\_title,  
 IMAGE = user\_image  
 WHERE USER\_ID = userid;  
 ELSIF user\_title IS NOT NULL THEN  
 -- Вставляем новую запись в USER\_PHOTO  
 INSERT INTO USER\_PHOTO(USER\_ID, TITLE, IMAGE)  
 VALUES (userid, user\_title, user\_image);  
 END IF;  
  
 COMMIT;  
  
END *ChangeDescriptionAccount*;  
  
-------------CompleteProject-------------  
drop procedure *CompleteProject*;  
CREATE OR REPLACE PROCEDURE *CompleteProject*(  
 projectId in PROJECT.ID%TYPE  
)  
IS  
 creatorID PROJECT.CREATOR\_ID%type;  
 BEGIN  
 UPDATE PROJECT  
 set IS\_READY = 1  
 where PROJECT.ID = projectId  
 returning PROJECT.CREATOR\_ID into creatorID;

insert into READY\_PROJECT(PROJECT\_ID, PERFORMER\_USER\_ID)  
 VALUES (projectId, creatorID);  
  
END *CompleteProject*;  
  
-------------DeleteProject-------------  
drop procedure *DeleteProject*;  
CREATE OR REPLACE PROCEDURE *DeleteProject*(  
 projectId in PROJECT.ID%TYPE  
)  
IS  
 creatorID PROJECT.CREATOR\_ID%type;  
BEGIN  
 delete TASK\_IN\_PROJECT  
 where TASK\_IN\_PROJECT.PROJECT\_ID = projectId;  
  
 delete PROJECT\_DOCUMENT  
 where PROJECT\_DOCUMENT.PROJECT\_ID = projectId;  
  
 delete PROJECT\_PHOTO  
 where PROJECT\_PHOTO.PROJECT\_ID = projectId;  
  
 delete PROJECT  
 where PROJECT.ID = projectId;  
END *DeleteProject*;  
  
-------------UpdateProject-------------  
drop procedure *UpdateProject*;  
CREATE OR REPLACE PROCEDURE *UpdateProject*(  
 project\_id IN PROJECT.id%TYPE,  
 project\_name IN PROJECT.name%TYPE,  
 toDate IN PROJECT.to\_date%TYPE,  
 project\_description IN PROJECT.description%TYPE,  
 photo\_name IN PROJECT\_PHOTO.title%TYPE DEFAULT NULL,  
 photo\_data IN PROJECT\_PHOTO.photo%TYPE DEFAULT NULL,  
 document\_name IN PROJECT\_DOCUMENT.title%TYPE DEFAULT NULL,  
 document\_type IN PROJECT\_DOCUMENT.type%TYPE DEFAULT NULL,  
 document\_data IN PROJECT\_DOCUMENT.data%TYPE DEFAULT NULL  
)  
IS  
BEGIN  
 -- Update data in the PROJECT table  
 UPDATE PROJECT  
 SET  
 name = project\_name,  
 to\_date = toDate,  
 description = project\_description

WHERE id = project\_id;  
  
 -- Update data in the PROJECT\_PHOTO table if values are provided  
 IF photo\_name IS NOT NULL AND photo\_data IS NOT NULL THEN  
 UPDATE PROJECT\_PHOTO  
 SET  
 title = photo\_name,  
 photo = photo\_data  
 WHERE project\_id = project\_id;  
 END IF;  
  
 -- Update data in the PROJECT\_DOCUMENT table if values are provided  
 IF document\_name IS NOT NULL AND document\_type IS NOT NULL AND document\_data IS NOT NULL THEN  
 UPDATE PROJECT\_DOCUMENT  
 SET  
 title = document\_name,  
 type = document\_type,  
 data = document\_data  
 WHERE project\_id = project\_id;  
 END IF;  
  
 COMMIT;  
END *UpdateProject*;  
  
-------------UpdateTask-------------  
drop procedure *UpdateTask*;  
CREATE OR REPLACE PROCEDURE *UpdateTask*(  
 task\_id IN TASK\_IN\_PROJECT.ID%TYPE,  
 descript IN TASK\_IN\_PROJECT.description%TYPE  
)  
IS  
BEGIN  
 -- Update data in the TASK\_IN\_PROJECT table  
 UPDATE TASK\_IN\_PROJECT  
 SET  
 description = descript  
 WHERE ID = task\_id;  
  
 COMMIT;  
END *UpdateTask*;  
  
-------------SendProject-------------  
drop procedure *SendProject*;  
CREATE OR REPLACE PROCEDURE *SendProject*(  
 projectId in PROJECT.ID%TYPE,

descrip in SEND\_PROJECT.DESCRIPTION%TYPE  
)  
IS  
 artistID PROJECT.CREATOR\_ID%type;  
 BEGIN  
 UPDATE PROJECT  
 set IS\_SEND = 1  
 where PROJECT.ID = projectId  
 returning PROJECT.ARTIST\_ID into artistID;  
  
 insert into SEND\_PROJECT(PROJECT\_ID, SENDER\_USER\_ID, DESCRIPTION)  
 VALUES (projectId, artistID, descrip);  
  
END *SendProject*;  
  
-------------CreateProject-------------  
drop procedure *CreateTeamProject*;  
CREATE OR REPLACE PROCEDURE *CreateTeamProject*(  
 creatorid IN PROJECT.creator\_id%TYPE,  
 project\_name IN PROJECT.name%TYPE,  
 toDate IN PROJECT.to\_date%TYPE,  
 project\_description IN PROJECT.description%TYPE,  
 photo\_name IN PROJECT\_PHOTO.title%TYPE DEFAULT NULL,  
 photo\_data IN PROJECT\_PHOTO.photo%TYPE DEFAULT NULL,  
 document\_name IN PROJECT\_DOCUMENT.title%TYPE DEFAULT NULL,  
 document\_type IN PROJECT\_DOCUMENT.type%TYPE DEFAULT NULL,  
 document\_data IN PROJECT\_DOCUMENT.data%TYPE DEFAULT NULL,  
 artistId in PROJECT.ARTIST\_ID%type,  
 current\_id OUT PROJECT.id%TYPE  
)  
IS  
 cnt NUMBER;  
  
BEGIN  
  
 SELECT *COUNT*(\*) INTO cnt FROM USERS where ID = artistId;  
 IF(cnt=0) then  
 *RAISE\_APPLICATION\_ERROR*(-20001, 'This person already exists');  
 END IF;  
  
 -- Insert data into the PROJECT table  
 INSERT INTO PROJECT (  
 name,  
 to\_date,  
 description,  
 creator\_id,  
 type,

is\_ready,  
 is\_send,  
 ARTIST\_ID  
 ) VALUES (  
 project\_name,  
 toDate,  
 project\_description,  
 creatorid,  
 'group',  
 0,  
 0,  
 artistId  
 ) RETURNING id INTO cnt;  
  
 -- Insert data into the PROJECT\_PHOTO table if values are provided  
 IF photo\_name IS NOT NULL AND photo\_data IS NOT NULL THEN  
 INSERT INTO PROJECT\_PHOTO (  
 project\_id,  
 title,  
 photo  
 ) VALUES (  
 cnt,  
 photo\_name,  
 photo\_data  
 );  
 END IF;  
  
 -- Insert data into the PROJECT\_DOCUMENT table if values are provided  
 IF document\_name IS NOT NULL AND document\_type IS NOT NULL AND document\_data IS NOT NULL THEN  
 INSERT INTO PROJECT\_DOCUMENT (  
 project\_id,  
 title,  
 type,  
 data  
 ) VALUES (  
 cnt,  
 document\_name,  
 document\_type,  
 document\_data  
 );  
 END IF;  
  
 current\_id := cnt;  
  
 COMMIT;  
END *CreateTeamProject*;

-------------AcceptProject-------------  
drop procedure *AcceptProject*;  
CREATE OR REPLACE PROCEDURE *AcceptProject*(  
 projectId in PROJECT.ID%TYPE  
)  
IS  
 creatorID PROJECT.CREATOR\_ID%type;  
 BEGIN  
  
 UPDATE PROJECT  
 set IS\_READY = 1, IS\_SEND = 0  
 where PROJECT.ID = projectId  
 returning PROJECT.CREATOR\_ID into creatorID;  
  
 insert into READY\_PROJECT(PROJECT\_ID, PERFORMER\_USER\_ID)  
 VALUES (projectId, creatorID);  
  
 delete from SEND\_PROJECT where PROJECT\_ID = projectId;  
  
END *AcceptProject*;  
  
-------------RejectProject-------------  
drop procedure *RejectProject*;  
CREATE OR REPLACE PROCEDURE *RejectProject*(  
 projectId in PROJECT.ID%TYPE  
)  
IS  
 creatorID PROJECT.CREATOR\_ID%type;  
 BEGIN  
  
 UPDATE PROJECT  
 set IS\_READY = 0, IS\_SEND = 0  
 where PROJECT.ID = projectId  
 returning PROJECT.CREATOR\_ID into creatorID;  
  
 delete from SEND\_PROJECT where PROJECT\_ID = projectId;  
  
END *RejectProject*;  
  
-------------DeleteUser-------------  
drop procedure *DeleteUser*;  
CREATE OR REPLACE PROCEDURE *DeleteUser*(  
 userId in users.ID%TYPE  
)  
IS BEGIN  
 delete from USERS where id=userId;  
END *DeleteUser*;

-------------ConfirmUser-------------  
drop procedure *ConfirmUser*;  
CREATE OR REPLACE PROCEDURE *ConfirmUser*(  
 user\_id IN USERS.id%TYPE  
)  
IS  
 cnt NUMBER;  
  
 BEGIN  
 SELECT *COUNT*(\*) INTO cnt FROM USERS where id = user\_id;  
 IF(cnt=1) then  
 UPDATE USERS  
 SET IS\_REGISTER = 1  
 WHERE USERS.ID = user\_id;  
  
 COMMIT;  
 ELSE  
 *RAISE\_APPLICATION\_ERROR*(-20007, 'There is no such user');  
 END IF;  
  
end *ConfirmUser*;  
  
select \*  
from USERS;