МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ

ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет Информационных технологий

Кафедра Информационные системы и технологии

Специальность 1–40 01 01 Программное обеспечение информационных технологий

Специализация 1–40 01 01 Программное обеспечение информационных технологий (программирование интернет – изданий)

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**К КУРСОВОМУ ПРОЕКТУ НА ТЕМУ:**

«Контроль и управление задачами малых групп»

Выполнил студент Артём Елизавета Владимировна

(Ф.И.О.)

Руководитель проекта асс., Нистюк О.А.

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

Заведующий кафедрой к.т.н., доц. Смелов В.В .

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

Консультант:  асс., Нистюк О.А.

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

Нормоконтролер: асс., Нистюк О.А.

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

Курсовой проект защищен с оценкой

Минск 2022

**Реферат**

Пояснительная записка курсового проекта содержит 49 страниц пояснительной записки, 13 иллюстраций, 5 источников литературы, 2 приложение.

C#, .NET FRAMEWORK 4.7.2, Windows presentation foundation (WPF), ORACLE 21XE, ORACLE.MANAGEDDATAACCESS FOR C#, ORACLE DATA REPLICATION, XML IMPORT, XML EXPORT.

Цель курсового проекта: разработка базы данных для контроля задач и управление проектами небольших групп с применением технологии с технологией репликации данных между серверами СУБД.

Первая глава описывает приложения, которыми был вдохновлен курсовой проект, и пункты, которые следует имплементировать.

Вторая глава описывает процесс создания и настройки базы данных, с которой будет происходить взаимодействие.

Третья глава описывает средства взаимодействия с базой данных, предоставленные платформой .NET и драйвером Oracle.ManagedDataAccess.

Четвёртая глава описывает изученные и использованные в процессе разработки технологии.

В пятой главе приведено руководство пользователя по взаимодействию с приложением и описаны базовые тестовые случаи.

В заключении приведены результаты проделанной работы.

**Abstract**

The course project explanatory note consists of 49 pages of explanatory note, 13 illustrations, 5 literature sources, 2 appendix.

C#, .NET FRAMEWORK 4.7.2, Windows presentation foundation (WPF), ORACLE 21XE, ORACLE.MANAGEDDATAACCESS FOR C#, ORACLE DATA REPLICATION, XML IMPORT, XML EXPORT.

The goal of course project is to design database for task control and small group project management using technology with data replication technology between DBMS servers.

The first chapter describes the main inspirations behind the project’s theme and also some reference points.

The second chapter describes the process of building and setting up the main database.

The third chapter describes methods of interaction with database using tools provided by .NET platform and Oracle.ManagedDataAcess driver.

The fourth chapter describes the technologies used or studied during development.

The fifth chapter is a users’ guide to the app combined with test cases.

Conclusion states the results of this project.

**Содержание**

[Введение 5](#_Toc122383038)

[1 Аналитический обзор литературы 6](#_Toc122383039)

[2 Архитектура базы данных 8](#_Toc122383040)

[2.1 Таблицы базы данных 8](#_Toc122383041)

[2.2 Процедуры и функции для решения поставленных задач 9](#_Toc122383042)

[2.2.1 Идентификация пользователей 9](#_Toc122383043)

[2.2.2 Процедуры для различных манипуляций на страницах приложения 13](#_Toc122383044)

[2.2.3 Процедуры для добавления данных в таблицы 14](#_Toc122383045)

[2.2.4 Процедуры для изменения данных в таблицах 16](#_Toc122383046)

[2.2.5 Процедуры для удаления данных из таблиц 20](#_Toc122383047)

[2.2.6 Процедуры для добавления 100 000 строк 22](#_Toc122383048)

[2.2.7 Процедура для экспорта и импорта таблиц в XML 23](#_Toc122383049)

[2.3 Представления базы данных 24](#_Toc122383050)

[2.4 Пользователи базы данных 24](#_Toc122383051)

[3 Обоснование технических приемов программирования 26](#_Toc122383052)

[4 Описание технологий 28](#_Toc122383053)

[4.1 Репликация данных 28](#_Toc122383054)

[4.2 Мультимедийные типы данных 36](#_Toc122383055)

[4.3 Средства диагностики 38](#_Toc122383056)

[5 Руководство пользователя, тестирование 39](#_Toc122383057)

[Заключение 46](#_Toc122383058)

[Список литературы 47](#_Toc122383059)

[Приложение А 48](#_Toc122383060)

[Приложение Б 49](#_Toc122383061)

Введение

Целью данной работы была разработка реляционной базы данных на тему «Контроль задач и управление проектами небольших групп». База данных должна быть составлена для контроля задач и управление проектами небольших групп. Так же было необходимо разработать соответствующее приложение, для демонстрации её работы.

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 СУБД для базы данных была выбрана Oracle 21XE, в связи с ее высокой производительностью и надежностью.

Так же было необходимо разработать приложение для демонстрации работы базы данных, взаимодействия с ней. Приложение было написано на языке программирования C# с графическим интерфейсом, выполненным с помощью WPF. Для взаимодействия с сервером базы данных Oracle использовался пакет Oracle Managed Data Access для .NET.

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.

Для обеспечения безопасности пользователей приложения в моем курсовом проекте используется технология шифрования паролей от аккаунта перед записью их в базу данных. А также использовалась технологией репликации данных между серверами СУБД для резервирования в случае выхода из строя одного из серверов.

Основные требования к приложению:

* Создание команд пользователей.
* Вход и регистрация в приложение.
* Добавление пользователей в команды.
* Реализация ролей администратора и рядового пользователя для команд.
* Добавление задач и управление ими.
* Отображение статистики по выполнению задач.
* Взаимодействие с базой данных при помощи хранимых процедур.

В пояснительной записке вы сможете найти краткую информацию о похожих продуктах, архитектуре, реализации проекта, руководстве пользователя.

1. Аналитический обзор литературы

В современном мире большая часть работы осуществляется в команде людей. Для более продуктивной коммуникации и трудоспособности необходимо отслеживать задачи и результаты их выполнения.

Одним из примеров использования приложений для контроля и отслеживания задач является IT-сфера, где над разработкой работают люди в команде и каждый выполняет какие-то свои задачи, которые зачастую ограничиваются крайним сроком выполнения работы.

Одним из основных требований к подобным приложениям является надежность, возможность аналитики, отслеживание задач и контроль их выполнения.

При разработке приложения было обращено внимание на некоторые популярные приложения с подобной задачей. Они рассмотрены ниже.

Trello — облачная программа для управления проектами небольших групп, разработанная Fog Creek Software. Trello использует парадигму для управления проектами, известную как канбан, метод, который первоначально был популяризирован Toyota в 1980-х для управления цепочками поставок. Интерфейс этого сервиса можно увидеть на рисунке 1.1.

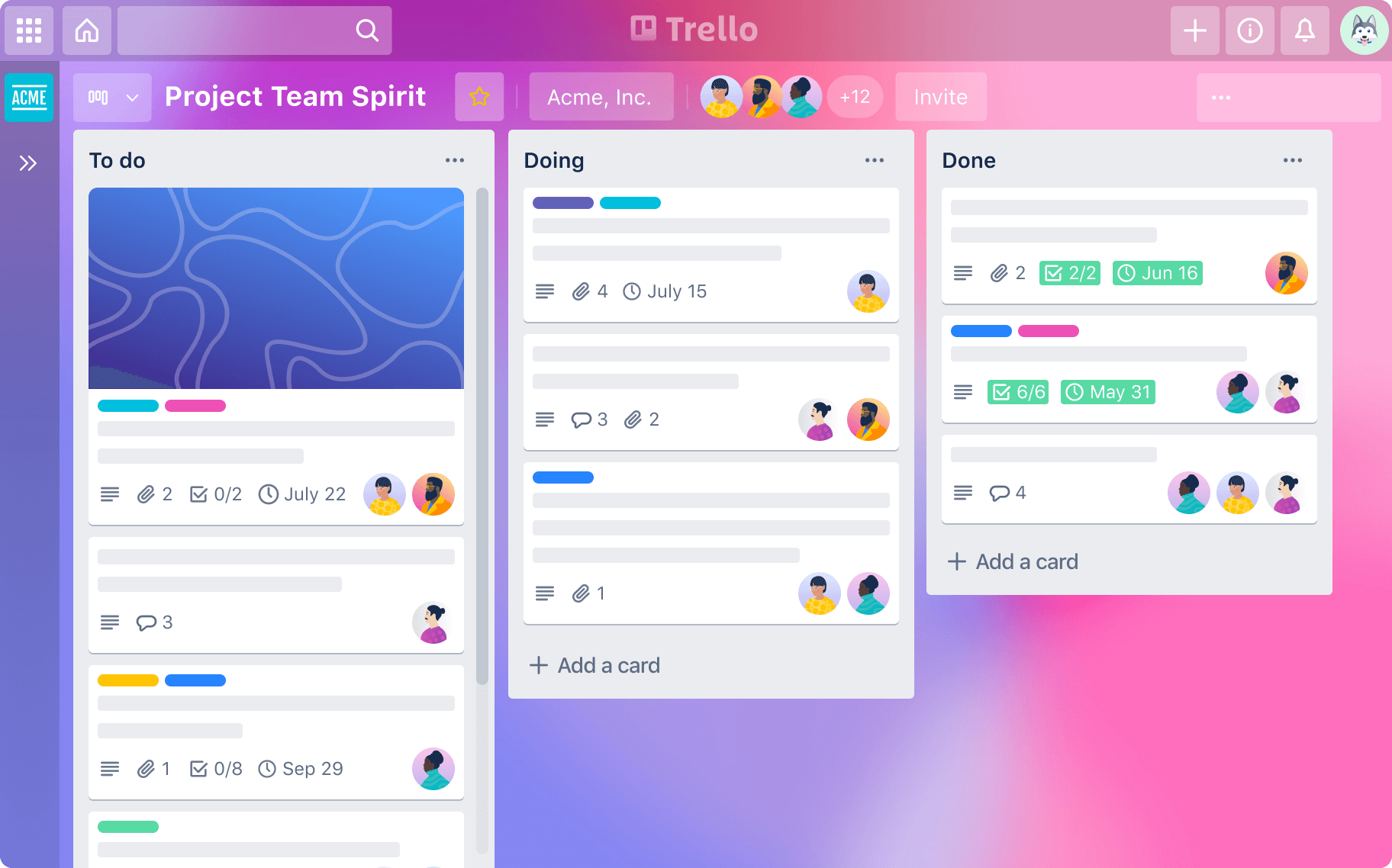


Рисунок 1.1 – Страница с задачами Trello

На странице с задачами в Trello четко видны блоки с задачами, пользователи с доступом к данным доскам. Также можно отметить, что у одного пользователя есть множество досок, как и команд, участником которых он может являться. В Trello есть возможность изменения профиля пользователя и возможность изменения дизайна для определенных досок.

Jira — коммерческая система отслеживания ошибок, предназначена для организации взаимодействия с пользователями, хотя в некоторых случаях используется и для управления проектами. Разработана компанией Atlassian, является одним из двух её основных продуктов. Имеет веб-интерфейс. Рассмотрим интерфейс на рисунке 1.2.

В Jira на странице с задачами также можно заметить список с необходимыми к выполнению задачами, их приоритет, срок исполнения, статус. В боковом меню можно перейти к самой доске с задачами, календарю, графику выполнения задач.

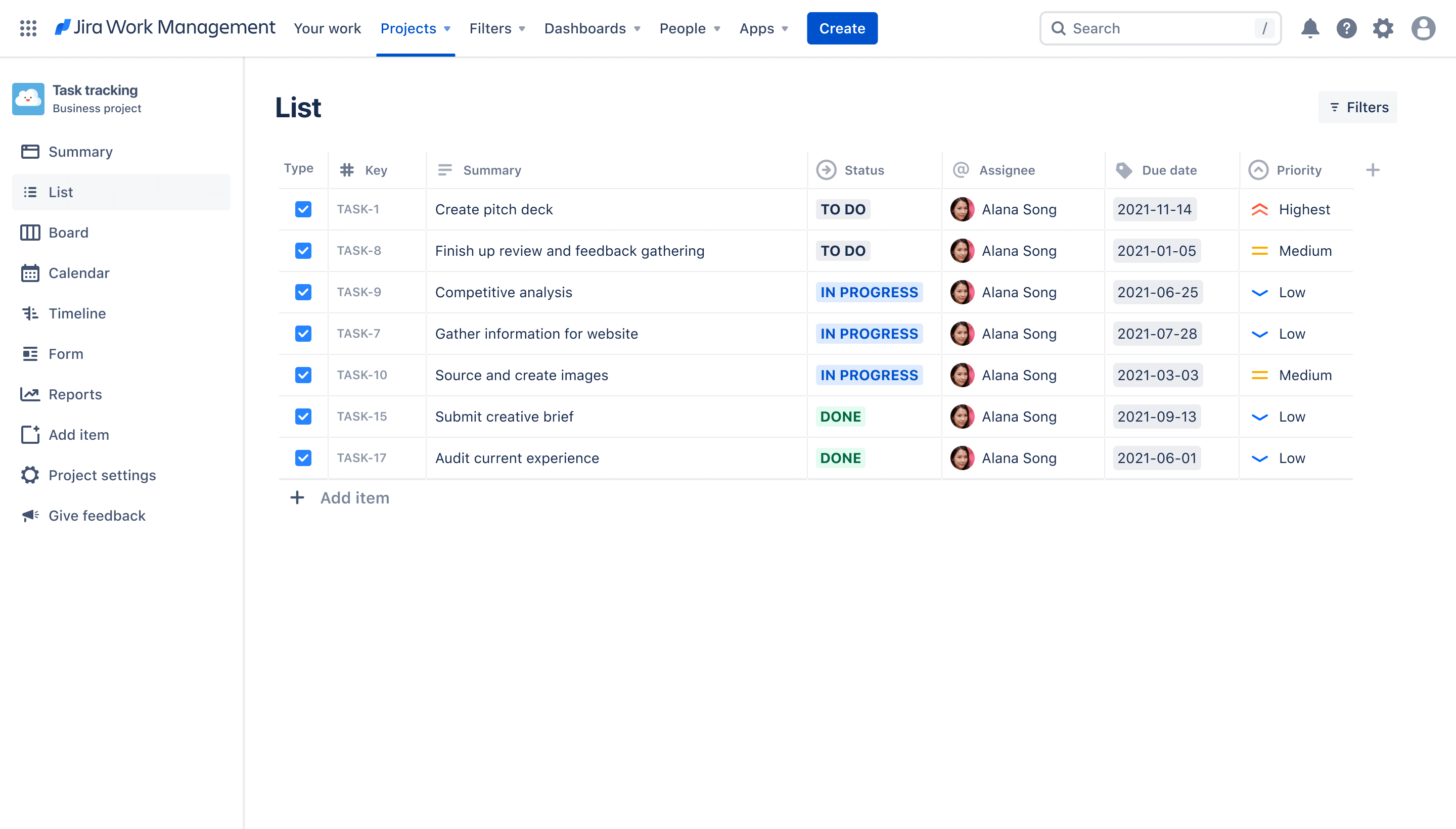


Рисунок 1.2 – Страница с задачами Jira

Разобрав два приведенных выше сервиса для контроля задач, можно сформировать общий функционал необходимый для работы приложения.

Прежде всего, естественно, база данных должна корректно работать с большим количеством задач и информации о них. Также стоит ввести меню настроек для пользователя, чтобы он мог изменить свой пароль или логин, оформление профиля по желанию.

1. Архитектура базы данных

При разработке приложения для курсового проекта была использована база данных Oracle 21XE.

* 1. Таблицы базы данных

Для реализации базы данных было разработано 9 таблиц. Диаграмма базы данных со структурой связей представлена в приложении А.

При разработке приложение получило рабочее название «Noto», соответственно схема базы данных называется DBNoto. В ее структуру входят следующие таблицы: AppRoleTable, UserTable, TaskStatuses, TaskPriorities, TaskComments, TaskTable, TeamTable, UserTeamPrivs, UserTeamPrivTable. Ниже мы разберем каждую из них более подробно.

Таблица AppRoleTable представляет собой перечень ролей, которые могут иметь пользователи приложения. Имеющиеся столбцы: RoleID (идентификатор роли, типа number(10), первичный ключ), RoleName (текстовое название роли, тип varchar2(30)). Имеющиеся по умолчанию роли: USER (рядовой пользователь) и ADMIN (пользователь с правами администратора).

Таблица UserTable содержит информацию о пользователях, а именно: UserID (идентификатор пользователя, тип number(10), первичный ключ), UserLogin (логин пользователя, типа varchar2(30)), UserPassword (пароль пользователя, тип varchar2(200), шифруется в процессе регистрации и расшифровывается при проверке во время авторизации), UserName (имя пользователя, типа varchar2(30)), UserLastName (фамилия пользователя, типа varchar2(30)), UserPhoneNumber (номер телефона пользователя, типа varchar2(13)), UserEmail (электронная почта пользователя, типа varchar2(30)), UserIcon (изображение профиля пользователя), UserRole (идентификатор роли, тип number(10), внешний ключ).

Таблица TaskStatuses содержит информацию о статусе задач: TaskStatusID (идентификатор статуса задачи, типа number(10), первичный ключ), TaskStatusName (название статуса задачи, тип varchar2(30)). Имеющиеся по умолчанию статусы: NOT STARTED (выполнение задачи не начато), STARTED (выполнение задачи начато), IN PROCESS (задача выполняется) и READY (задача выполнена).

Таблица TaskPriorities содержит информацию приоритете выполнения задачи: TaskPriorityID (идентификатор команды, типа number(10), первичный ключ), TaskPriorityName (название приоритета задачи, тип varchar2(30)). Имеющиеся по умолчанию приоритеты: HIGH (высокий приоритет), MEDIUM (средний приоритет) и LOW (низкий приоритет).

Таблица TaskComments содержит информацию о комментариях к задачам: ComID (идентификатор комментария, типа number(10), первичный ключ), ComDate (дата создания комментария, типа varchar2(30)), ComText (текст комментария, тип varchar2(100)), ComUser (автор комментария, типа number(10), внешний ключ).

Таблица TaskTable содержит задачи и информацию о них: TaskID (идентификатор задачи, тип number(10), первичный ключ), TaskStatus (идентификатор состояния задачи, тип number(10), внешний ключ), TaskPriority (идентификатор приоритета задачи, тип number(10), внешний ключ), TaskTitle (название задачи, тип varchar2(30)), TaskDescription (текстовое описание задачи, тип varchar2(200)), CreationDate (дата создания задачи, типа varchar2(30)), DeadlineDate (дата завершения задачи, типа varchar2(30)), Comments (идентификатор комментария пользователя к задаче, типа varchar2(10), внешний ключ).

Таблица TeamTable содержит информацию о командах: TeamID (идентификатор команды, типа number(10), первичный ключ), TeamName (название команды, тип varchar2(10)), TeamIcon (изображение команды), TeamTasks (идентификатор задач команды, типа number(10), внешний ключ).

Таблица UserTeamPrivs отвечает за хранение информации о привилегиях пользователя в команде: UserTeamPrivID (идентификатор привилегии, тип number(10), первичный ключ), UserTeamPrivName (описание привилегии, типа varchar2(10)). Имеющиеся по умолчанию привилегии пользователя в команде: OWNER (создатель команды), STARTED (выполнение задачи начато) и TEAMMATE (участник команды).

Таблица UserTeamPrivTable отвечает за связь таблиц с пользователями, командами и привилегиями пользователей в конкретной команде: PrivUser (идентификатор пользователя, тип number(10), внешний ключ), PrivTeam (идентификатор команды, тип number(10), внешний ключ), Privelegy (идентификатор привилегии пользователя, тип number(10), внешний ключ).

* 1. Процедуры и функции для решения поставленных задач

Для управления данными через приложение администратор использует хранимые процедуры. Хранимая процедура – объект базы данных, представляющий собой набор SQL-инструкций, который компилируется один раз и хранится на сервере.

Написанные мной в ходе разработки курсового проекта процедуры и функции можно разбить на несколько категорий. Их мы рассмотрим ниже.

* + 1. Идентификация пользователей

Во-первых, был написан блок процедур и функций, отвечающих за идентификацию пользователей.

Прежде всего, рассмотрим функцию encryption\_password. Ее реализацию можно увидеть на рисунке 2.1. Эта функция позволяет зашифровать пароль пользователя. Она принимает один входной параметр: пароль пользователя, который следует зашифровать. В моем курсовом проекте для шифрования используется алгоритм AES с 128-разрядным ключом, метод сцепления CBC и механизм заполнения PKCS#5.

Для шифрования необходимо выбрать ключ. В приложении в качестве ключа используется строка «2201200314775132». 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.

CREATE OR REPLACE FUNCTION encryption\_password

(p\_user\_password IN UserTable.UserPassword%TYPE)

RETURN UserTable.UserPassword%TYPE

IS

l\_key VARCHAR2(2000) := '2201200314775132';

l\_in\_val VARCHAR2(2000) := p\_user\_password;

l\_mod NUMBER := DBMS\_CRYPTO.encrypt\_aes128 + DBMS\_CRYPTO.chain\_cbc + DBMS\_CRYPTO.pad\_pkcs5;

l\_enc RAW(2000);

BEGIN

l\_enc := DBMS\_CRYPTO.encrypt(utl\_i18n.string\_to\_raw(l\_in\_val, 'AL32UTF8'), l\_mod, utl\_i18n.string\_to\_raw(l\_key, 'AL32UTF8'));

RETURN RAWTOHEX(l\_enc);

END encryption\_password;

Листинг 2.1 – Функция encryption\_password

В обратных целях была разработана функция decryption\_password. Она в качестве параметра принимает зашифрованный пароль и должна в результате своей работе вернуть его в расшифрованном виде. Используя тот же самый ключ, передаем его в качестве одного из параметров в функцию DECRYPT, перед этим преобразовав его в RAW при помощи функции STRING\_TO\_RAW. Так же не забываем передать в функцию параметр type и зашифрованный пароль типа RAW.

Результат функции преобразуем к строке при помощи функции RAW\_TO\_CHAR пакета UTL\_I18N. Реализацию этой функции можно увидеть на листинге 2.2.

CREATE OR REPLACE FUNCTION encryption\_password

(p\_user\_password IN UserTable.UserPassword%TYPE)

RETURN UserTable.UserPassword%TYPE

IS

l\_key VARCHAR2(2000) := '2201200314775132';

l\_in\_val VARCHAR2(2000) := p\_user\_password;

l\_mod NUMBER := DBMS\_CRYPTO.encrypt\_aes128 + DBMS\_CRYPTO.chain\_cbc + DBMS\_CRYPTO.pad\_pkcs5;

l\_enc RAW(2000);

BEGIN

Продолжение листинга 2.2

l\_enc := DBMS\_CRYPTO.encrypt(utl\_i18n.string\_to\_raw(l\_in\_val, 'AL32UTF8'), l\_mod, utl\_i18n.string\_to\_raw(l\_key, 'AL32UTF8'));

RETURN RAWTOHEX(l\_enc);

END encryption\_password;

Листинг 2.2 – Функция decryption\_password

Далее была разработана хранимая процедура register\_user, с помощью которой происходит регистрация пользователей в приложении. Эта процедура принимает два параметра: p\_user\_login (логин пользователя, которого нужно зарегистрировать) и p\_user\_password (пароль от аккаунта этого пользователя). В случае, если указанный логин уже есть в базе данных, поднимается исключение. Если исключения не произошло, то строка успешно вставляется в таблицу UserTable, зашифровав пароль в процессе.

CREATE OR REPLACE PROCEDURE register\_user

(p\_user\_login IN UserTable.UserLogin%TYPE,

p\_user\_password IN UserTable.UserPassword%TYPE,

p\_user\_name IN UserTable.UserName%TYPE,

p\_user\_lastname IN UserTable.UserLastName%TYPE,

p\_user\_phonenumber IN UserTable.UserPhoneNumber%TYPE,

p\_user\_email IN UserTable.UserEmail%TYPE)

IS

cnt NUMBER;

BEGIN

SELECT COUNT(\*) INTO cnt FROM UserTable WHERE UPPER(UserLogin) = UPPER(p\_user\_login);

IF (cnt = 0) THEN

INSERT INTO UserTable(UserLogin, UserPassword, UserName, UserLastName, UserPhoneNumber, UserEmail, UserRole)

VALUES (UPPER(p\_user\_login), encryption\_password(UPPER(p\_user\_password)), p\_user\_name, p\_user\_lastname, p\_user\_phonenumber, p\_user\_email, 1);

COMMIT;

ELSE

RAISE\_APPLICATION\_ERROR(-20001, 'This login is already taken');

END IF;

END register\_user;

Листинг 2.3 – Процедура register\_user

Для авторизации пользователя в приложении разработана процедура log\_in\_user. У нее есть два входных параметра: p\_user\_login (логин пользователя) и p\_user\_password (пароль пользователя). Если указанная комбинация логина и пароля отсутствует в таблице user\_table, то поднимается исключение. Если исключения не произошло, то в приложение передается три выходных параметра, которые пригодятся далее в разработке логики приложения и будут занесены в поля статического класса DataWorker: o\_user\_id (идентификатор пользователя, прошедшего авторизацию), o\_user\_login (логин пользователя), o\_user\_role (роль пользователя), o\_user\_name (имя пользователя), o\_user\_lastname (фамилия пользователя), o\_user\_phonenumber (номер телефона пользователя), o\_user\_email (электронная почта пользователя).

CREATE OR REPLACE PROCEDURE log\_in\_user

(p\_user\_login IN UserTable.UserLogin%TYPE,

p\_user\_password IN UserTable.UserPassword%TYPE,

o\_user\_id OUT UserTable.UserID%TYPE,

o\_user\_login OUT UserTable.UserLogin%TYPE,

o\_user\_role OUT AppRoleTable.RoleName%TYPE,

o\_user\_name OUT UserTable.UserName%TYPE,

o\_user\_lastname OUT UserTable.UserLastName%TYPE,

o\_user\_phonenumber OUT UserTable.UserPhoneNumber%TYPE,

o\_user\_email OUT UserTable.UserEmail%TYPE)

IS

cnt NUMBER;

uri AppRoleTable.RoleID%TYPE;

BEGIN

SELECT COUNT(\*) INTO cnt FROM UserTable WHERE UPPER(UserLogin) = UPPER(p\_user\_login) AND UserPassword = encryption\_password(UPPER(p\_user\_password));

IF (cnt != 0) THEN

SELECT UserID INTO o\_user\_id FROM UserTable WHERE UPPER(UserLogin) = UPPER(p\_user\_login);

SELECT UserLogin INTO o\_user\_login FROM UserTable WHERE UPPER(UserLogin) = UPPER(p\_user\_login);

SELECT UserRole INTO uri FROM UserTable WHERE UPPER(UserLogin) = UPPER(p\_user\_login);

SELECT RoleName INTO o\_user\_role FROM AppRoleTable WHERE RoleID = uri;

SELECT UserName INTO o\_user\_name FROM UserTable WHERE UPPER(UserLogin) = UPPER(p\_user\_login);

SELECT UserLastName INTO o\_user\_lastname FROM UserTable WHERE UPPER(UserLogin) = UPPER(p\_user\_login);

SELECT UserEmail INTO o\_user\_email FROM UserTable WHERE UPPER(UserLogin) = UPPER(p\_user\_login);

SELECT UserPhoneNumber INTO o\_user\_phonenumber FROM UserTable WHERE UPPER(UserLogin) = UPPER(p\_user\_login);

ELSE

RAISE\_APPLICATION\_ERROR(-20002, 'User is not found');

END IF;

Продолжение листинга 2.4

END log\_in\_user;

Листинг 2.4 – Процедура log\_in\_user

Процедура check\_role позволяет найти роль пользователя и выгружает её для дальнейших манипуляций. Принимает один входной параметр, а именно: логин пользователя (p\_user\_login), а также один выходной параметр: название роли пользователя (o\_user\_role).

CREATE OR REPLACE PROCEDURE check\_role

(p\_user\_login IN UserTable.UserLogin%TYPE,

o\_user\_role OUT AppRoleTable.RoleName%TYPE)

IS

CURSOR role\_cursor IS SELECT RoleName FROM AppRoleTable\_view WHERE UPPER(UserLogin) = UPPER(p\_user\_login);

BEGIN

OPEN role\_cursor;

FETCH role\_cursor INTO o\_user\_role;

IF role\_cursor%NOTFOUND THEN

RAISE\_APPLICATION\_ERROR(-20004, 'App role error');

END IF;

CLOSE role\_cursor;

END check\_role;

Листинг 2.5 – Процедура log\_in\_user

Также были разработаны другие процедуры для взаимодействия и связи пользователя с командами и задачами.

* + 1. Процедуры для различных манипуляций на страницах приложения

Процедура search\_user позволяет найти информацию о пользователе и выгружает данные о нем из представления UserTeam\_view. Она используется для отображения данных на странице «Профиль» и данных о пользователях, состоящих в определенной команде. Процедура принимает один входной параметр, а именно: логин пользователя (p\_user\_login), а также несколько выходных параметра, а именно: идентификатор пользователя (p\_user\_id), имя пользователя (o\_user\_name), фамилию пользователя (o\_user\_lastname), номер телефона пользователя (o\_user\_phonenumber) и электронную почту пользователя (o\_user\_email).

Процедура update\_user\_login позволяет изменить логин пользователя. Данная процедура принимает два входных параметра: старый логин пользователя (p\_user\_login) и новый логин пользователя (p\_new\_user\_login), которым следует заменить старый.

Процедура update\_user\_password позволяет изменить пароль пользователя. Эта процедура принимает два входных параметра, а именно: старый пароль пользователя (p\_user\_password) и расшифрованный новый пароль пользователя (p\_new\_user\_password), который заменит в таблице старый и будет зашифрован в процессе.

Процедура update\_user\_email позволяет изменить электронную почту пользователя. Она принимает два входных параметра, а именно: логин пользователя (p\_user\_login) и новую электронную почту пользователя (p\_new\_user\_email), которая заменит в таблице старое значение.

Процедура update\_user\_phonenumberпозволяет изменить номер телефона пользователя. Эта процедура принимает два входных параметра, а именно: логин пользователя (p\_user\_login) и новый номер телефона пользователя (p\_new\_user\_phonenumber), который заменит в таблице старое значение.

Процедура update\_user\_name позволяет изменить имя пользователя. Она принимает два входных параметра, а именно: логин пользователя (p\_user\_login) и новое имя пользователя (p\_new\_user\_name), которое заменит в таблице старое значение.

Процедура update\_user\_last\_name позволяет изменить фамилию пользователя. Эта принимает два входных параметра, а именно: логин пользователя (p\_user\_login) и новую фамилию пользователя (p\_new\_user\_last\_name), которая заменит в таблице старое значение.

Процедура count\_user\_in\_team позволяет определить количество пользователей в команде. Она принимает два параметра, а именно: идентификатор команды (p\_team\_id) и количество пользователей (o\_count\_users).

* + 1. Процедуры для добавления данных в таблицы

Процедура create\_team служит для добавления нового команды в базу данных. Данная процедура принимает два параметра: p\_user\_login (логин пользователя, который будет владельцем данной команды), p\_team\_name (название команды, которое будет добавлено в базу данных). Поскольку все процедуры, описанные в данном пункте, аналогичны по своему алгоритму, имеет смысл показать в качестве примера только пару из их них. Таким образом, на листинге 2.6 можно увидеть реализацию функции сreate\_team.

create or replace PROCEDURE create\_team

(p\_user\_login IN UserTable.UserLogin%TYPE,

p\_team\_name IN TeamTable.TeamName%TYPE)

IS

cnt NUMBER;

usr\_id UserTable.UserID%TYPE;

tm\_id TeamTable.TeamID%TYPE;

BEGIN

SELECT COUNT(\*) INTO cnt FROM TeamTable WHERE UPPER(TeamName) = UPPER(p\_team\_name);

IF (cnt = 0) THEN

INSERT INTO TeamTable (TeamName) VALUES(p\_team\_name);

Продолжение листинга 2.6

COMMIT;

SELECT TeamID INTO tm\_id FROM TeamTable WHERE UPPER(TeamName) = UPPER(p\_team\_name);

SELECT UserID INTO usr\_id FROM UserTable WHERE UPPER(UserLogin) = UPPER(p\_user\_login);

INSERT INTO UserTeamPrivTable(PrivUser,PrivTeam,Privelegy) VALUES(usr\_id,tm\_id,1);

COMMIT;

ELSE

RAISE\_APPLICATION\_ERROR(-20003, 'Team with this name already exists');

END IF;

END create\_team;

Листинг 2.6 – Процедура create\_team

Процедура create\_task служит для добавления нового задания в базу данных. Данная процедура принимает параметры: p\_user\_login (логин пользователя), p\_team\_name (название команды, в которое будет добавлено задание), p\_task\_title (заголовок задания), p\_task\_status (статус задания), p\_task\_priority (приоритет задания), p\_task\_description (описание задания), p\_task\_deadlineDate (крайний срок выполнения задачи). На листинге 2.7 можно увидеть реализацию функции сreate\_task.

create or replace PROCEDURE create\_task

(p\_user\_login IN UserTable.UserLogin%TYPE,

p\_team\_name IN TeamTable.TeamName%TYPE,

p\_task\_title IN TaskTable.TaskTitle%TYPE,

p\_task\_status IN TaskStatuses.TaskStatusID%TYPE,

p\_task\_priority IN TaskPriorities.TaskPriorityID%TYPE,

p\_task\_description IN TaskTable.TaskDescription%TYPE,

p\_task\_deadlineDate IN TaskTable.DeadlineDate%TYPE)

IS

tm\_id TeamTable.TeamID%TYPE;

BEGIN

SELECT TeamID INTO tm\_id FROM TeamTable WHERE UPPER(TeamName) = UPPER(p\_team\_name);

INSERT INTO TaskTable(TaskTeamID, TaskStatus, TaskPriority, TaskTitle, TaskDescription, CreationDate, DeadlineDate)

VALUES(tm\_id, p\_task\_status, p\_task\_priority, p\_task\_title, p\_task\_description, TO\_CHAR(SYSDATE, 'DD.MM.YYYY'), p\_task\_deadlineDate);

COMMIT;

END create\_task;

Листинг 2.7 – Процедура create\_task

Процедура add\_user\_to\_team служит для добавления уже созданного пользователя в команду. Данная процедура принимает параметры: p\_user\_id (идентификатор пользователя), p\_team\_id (идентификатор команды, в которую будет добавлен пользователь). На листинге 2.8 можно увидеть реализацию функции create\_task.

create or replace PROCEDURE add\_user\_to\_team

(p\_user\_login IN UserTable.UserLogin%TYPE,

p\_team\_id IN TeamTable.TeamID%TYPE)

IS

user\_id UserTable.UserID%TYPE;

cnt NUMBER;

BEGIN

SELECT COUNT(\*) INTO CNT FROM UserTable WHERE upper(UserLogin) = upper(p\_user\_login);

SELECT UserID INTO user\_id FROM UserTable WHERE upper(UserLogin) = upper(p\_user\_login);

IF (cnt != 0) THEN

SELECT COUNT(\*) INTO cnt FROM UserTeamPrivTable WHERE PrivUser = user\_id AND PrivTeam = p\_team\_id;

IF (cnt = 0) THEN

INSERT INTO UserTeamPrivTable(PrivUser, PrivTeam, Privelegy) VALUES(user\_id, p\_team\_id, 2);

COMMIT;

ELSE

RAISE\_APPLICATION\_ERROR(-20011, 'This user is already add');

END IF;

ELSE

RAISE\_APPLICATION\_ERROR(-20002, 'This user is not find');

END IF;

END add\_user\_to\_team;

Листинг 2.8 – Процедура add\_user\_to\_team

* + 1. Процедуры для изменения данных в таблицах

В этом блоке будут описаны процедуры, с помощью которых изменяются данные в таблицах.

Процедура update\_user\_login отвечает за замену старого имени пользователя на новое. Она соответственно принимает эти имена как параметры: p\_user\_login (текущий логин пользователя) и p\_new\_user\_login (новый логин пользователя). Она изображена на листинге 2.9.

create or replace NONEDITIONABLE PROCEDURE update\_user\_login

(p\_user\_login IN UserTable.UserLogin%TYPE,

p\_new\_user\_login IN UserTable.UserLogin%TYPE)

IS

cnt NUMBER;

Продолжение листинга 2.9

BEGIN

SELECT COUNT(\*) INTO cnt FROM UserTable WHERE UPPER(UserLogin) = UPPER(p\_user\_login);

IF (cnt != 0) THEN

SELECT COUNT(\*) INTO cnt FROM UserTable WHERE UPPER(UserLogin) = UPPER(p\_new\_user\_login);

IF (cnt = 0) THEN

UPDATE UserTable SET UserLogin = UPPER(p\_new\_user\_login) WHERE UPPER(UserLogin) = UPPER(p\_user\_login);

COMMIT;

ELSE

RAISE\_APPLICATION\_ERROR(-20001, 'This login is already taken');

END IF;

ELSE

RAISE\_APPLICATION\_ERROR(-20002, 'User is not found');

END IF;

END update\_user\_login;

Листинг 2.9 – Процедура update\_user\_login

Процедура update\_team\_owner отвечает за изменение владельца команды. Она соответственно принимает эти имена как параметры: p\_team\_name (название команды) и p\_new\_owner (идентификатор пользователя - нового владельца команды). Она изображена на листинге 2.10.

create or replace NONEDITIONABLE PROCEDURE update\_team\_owner

(p\_team\_name IN TeamTable.TeamName%TYPE,

p\_new\_owner IN UserTable.UserLogin%TYPE)

IS

old\_owner\_id UserTable.UserID%TYPE;

new\_owner\_id UserTable.UserID%TYPE;

team\_id TeamTable.TeamID%TYPE;

cnt\_user NUMBER;

cnt\_team NUMBER;

BEGIN

SELECT COUNT(\*) INTO cnt\_user FROM UserTable WHERE UPPER(UserLogin) = UPPER(p\_new\_owner);

IF (cnt\_user != 0) THEN

SELECT COUNT(\*) INTO cnt\_team FROM TeamTable WHERE UPPER(TeamName) = UPPER(p\_team\_name);

IF (cnt\_team != 0) THEN

SELECT TeamID INTO team\_id FROM TeamTable WHERE UPPER(TeamName) = UPPER(p\_team\_name);

SELECT PrivUser INTO old\_owner\_id FROM UserTeamPrivTable WHERE Privelegy = 1 AND PrivTeam = team\_id;

SELECT UserID INTO new\_owner\_id FROM UserTable WHERE UPPER(UserLogin) = UPPER(p\_new\_owner);

Продолжение листинга 2.10

UPDATE UserTeamPrivTable SET Privelegy = 1 WHERE PrivUser = new\_owner\_id AND PrivTeam = team\_id;

UPDATE UserTeamPrivTable SET Privelegy = 2 WHERE PrivUser = old\_owner\_id AND PrivTeam = team\_id;

COMMIT;

ELSE

RAISE\_APPLICATION\_ERROR(-20002, 'Team is not found');

END IF;

ELSE

RAISE\_APPLICATION\_ERROR(-20002, 'User is not found');

END IF;

END update\_team\_owner;

Листинг 2.10 – Процедура update\_team\_owner

Процедура update\_task\_title отвечает за замену старого заголовка задания на новое. Принимает следующие параметры: p\_task\_id (идентификатор задания) и p\_new\_task\_title (новый заголовок задания). Она изображена на листинге 2.11.

create or replace PROCEDURE update\_task\_title

(p\_task\_id IN TaskTable.TaskID%TYPE,

p\_new\_task\_title IN TaskTable.TaskTitle%TYPE)

IS

cnt NUMBER;

BEGIN

SELECT COUNT(\*) INTO cnt FROM TaskTable WHERE TaskID = p\_task\_id;

IF (cnt != 0) THEN

UPDATE TaskTable SET TaskTitle = p\_new\_task\_title WHERE TaskID = p\_task\_id;

ELSE

RAISE\_APPLICATION\_ERROR(-20002, 'Task is not found');

END IF;

END update\_task\_title;

Листинг 2.11 – Процедура update\_task\_title

Процедура update\_task\_priority отвечает за замену старого приоритета задания на новое. Принимает следующие параметры: p\_task\_id (идентификатор задания) и p\_new\_task\_priority (приоритет задания) Она изображена на листинге 2.12.

create or replace NONEDITIONABLE PROCEDURE update\_task\_priority

(p\_task\_id IN TaskTable.TaskID %TYPE,

p\_new\_task\_priority IN TaskTable.TaskPriority%TYPE)

IS

cnt NUMBER;

BEGIN

SELECT COUNT(\*) INTO cnt FROM TaskTable WHERE TaskID = p\_task\_id;

Продолжение листинга 2.12

IF (cnt != 0) THEN

UPDATE TaskTable SET TaskPriority = p\_new\_task\_priority WHERE TaskID = p\_task\_id;

ELSE

RAISE\_APPLICATION\_ERROR(-20002, 'Task is not found');

END IF;

END update\_task\_priority;

Листинг 2.12 – Процедура update\_task\_priority

Процедура update\_task\_description отвечает за замену старого заголовка задания на новое. Принимает следующие параметры: p\_task\_id (идентификатор задания) и p\_new\_task\_description (новое описание задания). Она изображена на листинге 2.13.

create or replace NONEDITIONABLE PROCEDURE update\_task\_description

(p\_task\_id IN TaskTable.TaskID %TYPE,

p\_new\_task\_description IN TaskTable.TaskDescription%TYPE)

IS

cnt NUMBER;

BEGIN

SELECT COUNT(\*) INTO cnt FROM TaskTable WHERE TaskID = p\_task\_id;

IF (cnt != 0) THEN

UPDATE TaskTable SET TaskDescription = p\_new\_task\_description WHERE TaskID = p\_task\_id;

ELSE

RAISE\_APPLICATION\_ERROR(-20002, 'Task is not found');

END IF;

END update\_task\_description;

Листинг 2.13 – Процедура update\_task\_description

Процедура update\_task\_deadlinedate отвечает за замену старого дедлайна задания на новое. Принимает следующие параметры: p\_task\_id (идентификатор задания) и p\_new\_task\_deadlinedate (новая крайняя дата завершения задания). Она изображена на листинге 2.14.

create or replace NONEDITIONABLE PROCEDURE update\_task\_deadlinedate

(p\_task\_id IN TaskTable.TaskID %TYPE,

p\_new\_task\_deadlinedate IN TaskTable.DeadlineDate%TYPE)

IS

cnt NUMBER;

BEGIN

SELECT COUNT(\*) INTO cnt FROM TaskTable WHERE TaskID = p\_task\_id;

IF (cnt != 0) THEN

UPDATE TaskTable SET DeadlineDate = p\_new\_task\_deadlinedate WHERE TaskID = p\_task\_id;

Продолжение листинга 2.14

ELSE

RAISE\_APPLICATION\_ERROR(-20002, 'Task is not found');

END IF;

END update\_task\_deadlinedate;

Листинг 2.14 – Процедура update\_task\_deadlinedate

* + 1. Процедуры для удаления данных из таблиц

В этом блоке будут описаны процедуры, с помощью которых удаляются данных из таблиц.

Процедура delete\_user отвечает за удаление пользователя из базы данных. Она принимает один параметр: p\_login (логин удаляемого пользователя). Она изображена на листинге 2.15. При удалении пользователя удалятся все связанные с ним записи в БД.

create or replace NONEDITIONABLE PROCEDURE delete\_user

(p\_login IN UserTable.UserLogin%TYPE)

IS

user\_id UserTable.UserID%TYPE;

cnt NUMBER;

BEGIN

SELECT COUNT(\*) INTO cnt FROM UserTable WHERE upper(UserLogin) = upper(p\_login);

IF (cnt != 0) THEN

SELECT UserID INTO user\_id FROM UserTable WHERE upper(UserLogin) = upper(p\_login);

DELETE TaskComments WHERE ComUser = user\_id;

DELETE UserTeamPrivTable WHERE PrivUser = user\_id;

DELETE UserTable WHERE UserID = user\_id;

ELSE

RAISE\_APPLICATION\_ERROR(-20003, 'User is not found');

END IF;

END delete\_user;

Листинг 2.15 – Процедура delete\_user

Процедура delete\_team отвечает за удаление команды из базы данных. Она принимает один параметр: p\_id (идентификатор удаляемой команды). При удалении команды удалятся все связанные данные.

create or replace NONEDITIONABLE PROCEDURE delete\_team

(p\_id IN TeamTable.TeamID%TYPE)

IS

cnt NUMBER;

task\_id TaskTable.TaskID%TYPE;

BEGIN

Продолжение листинга 2.16

SELECT COUNT(\*) INTO cnt FROM TeamTable WHERE TeamID = p\_id;

IF (cnt != 0) THEN

DELETE TeamTable WHERE TeamID = p\_id;

ELSE

RAISE\_APPLICATION\_ERROR(-20009, 'Team is not found');

END IF;

END delete\_team;

Листинг 2.16 – Процедура delete\_team

Процедура delete\_task отвечает за удаление задания из базы данных. Она принимает один параметр: p\_id (идентификатор удаляемого задания). При удалении задания удалятся все связанные данные.

create or replace NONEDITIONABLE PROCEDURE delete\_task

(p\_id IN TaskTable.TaskID%TYPE)

IS

cnt NUMBER;

BEGIN

SELECT COUNT(\*) INTO cnt FROM TaskTable WHERE TaskID = p\_id;

IF (cnt != 0) THEN

DELETE TaskComments WHERE ComTask = p\_id;

DELETE TaskTable WHERE TaskID = p\_id;

commit;

ELSE

RAISE\_APPLICATION\_ERROR(-20008, 'Task is not found');

END IF;

END delete\_task;

Листинг 2.17 – Процедура delete\_task

Процедура remove\_user\_from\_team отвечает за удаление пользователя из команды. Она принимает один параметр: p\_user\_id (идентификатор пользователя).

create or replace NONEDITIONABLE PROCEDURE remove\_user\_from\_team

(p\_user\_id IN UserTable.UserID%TYPE,

p\_team\_id IN TeamTable.TeamID%TYPE)

IS

cnt NUMBER;

BEGIN

SELECT COUNT(\*) INTO cnt FROM UserTeamPrivTable WHERE PrivUser = p\_user\_id AND PrivTeam = p\_team\_id;

IF (cnt != 0) THEN

DELETE FROM UserTeamPrivTable WHERE PrivUser = p\_user\_id AND PrivTeam = p\_team\_id;

COMMIT;

ELSE

Продолжение листинга 2.18

RAISE\_APPLICATION\_ERROR(-20012, 'This user is not found in this team');

END IF;

END remove\_user\_from\_team;

Листинг 2.18 – Процедура remove\_user\_from\_team

* + 1. Процедуры для добавления 100 000 строк

С помощью процедуры insert\_100k\_users было зарегистрировано 100 000 пользователей. Она изображена на листинге 2.19.

CREATE OR REPLACE PROCEDURE insert\_100k\_users

IS

BEGIN

FOR i IN 1 .. 100000 LOOP

REGISTER\_USER('user' || i, 'pass' || i, 'UserName'|| i, 'UserLastName'|| i, '+375291234567', 'user@mail.ru');

END LOOP;

END insert\_100k\_users;

Листинг 2.19 – Процедура insert\_100k\_users

Поскольку процесс получения администратором информации о всех пользователях занимал достаточно большое время, было решено отображать их в окне администратора по 4.

С помощью процедуры insert\_100k\_tasks было создано 100 000 заданий. Она изображена на листинге 2.20.

CREATE OR REPLACE PROCEDURE insert\_100k\_tasks

IS

BEGIN

BEGIN create\_team('qusest','team\_demo'); commit; END;

FOR i IN 1 .. 100000 LOOP

create\_task('qusest', 'team\_demo', 'task\_title' || i, 1, 1, 'task\_desc' || i, TO\_CHAR(SYSDATE, 'DD.MM.YYYY'));

END LOOP;

END insert\_100k\_tasks;

Листинг 2.20 – Процедура insert\_100k\_tasks

Поскольку процесс получения информации о всех задачах занимал достаточно большое время, было решено отображать их в окне по 3.

* + 1. Процедура для экспорта и импорта таблиц в XML

Для реализации обоих процедур потребовалось создать директорий с желаемым расположением при подключении под пользователем SYS к инстансу и выдать пользователю, владеющему базой данных, гран на чтение и запись из этого директория. Это изображено на рисунке 2.21.

CREATE PUBLIC DATABASE LINK CWCON

CONNECT TO DBNoto

IDENTIFIED BY Pa$$w0rd

USING '25.49.158.45:1521/xe';

Листинг 2.21 – Создание директория CW\_DIR

В рамках задания нужно было разработать процедуру, экспортирующую данные из таблицы, содержащей 100 000 строк, в XML документ. С этой целью разработана процедура users\_export. Она изображена на рисунке 2.22.

CREATE OR REPLACE PROCEDURE users\_export

IS

rc sys\_refcursor;

doc DBMS\_XMLDOM.DOMDocument;

BEGIN

open rc for select USERLOGIN,USERPASSWORD,USERNAME,USERLASTNAME,USERPHONENUMBER,USEREMAIL,USERROLE, USERICON from UserTable;

doc := DBMS\_XMLDOM.NewDOMDocument(XMLTYPE(rc));

DBMS\_XMLDOM.WRITETOFILE(doc, 'CW\_DIR/users\_export.xml');

END users\_export;

Листинг 2.22 – Процедура users\_export

Также нужно было написать процедуру, которая бы, наоборот, импортировала существующие данные из XML документа в таблицу базы данных. В этом случае данные вставляются в таблицу UserTable. Процедура user\_import изображена на рисунке 2.23.

CREATE OR REPLACE PROCEDURE users\_import

IS

BEGIN

INSERT INTO UserTable(USERLOGIN,USERPASSWORD,USERNAME,USERLASTNAME,USERPHONENUMBER,USEREMAIL,USERROLE)

SELECT ExtractValue(VALUE(Users), '//LOGIN') AS UserLogin,

ExtractValue(VALUE(Users), '//PASSWORD') AS UserPassword,

ExtractValue(VALUE(Users), '//NAME') AS UserName,

ExtractValue(VALUE(Users), '//LASTNAME') AS UserLastName,

ExtractValue(VALUE(Users), '//PHONENUMBER') AS UserPhoneNumber,

Продолжение листинга 2.23

ExtractValue(VALUE(Users), '//EMAIL') AS UserEmail,

ExtractValue(VALUE(Users), '//ROLE') AS UserRole

FROM TABLE(XMLSequence(EXTRACT(XMLTYPE(bfilename('CW\_DIR', 'users\_export.xml'),

nls\_charset\_id('UTF-8')),'/ROWSET/ROW'))) Users;

END users\_import;

Листинг 2.23 – Процедура users\_import

* 1. Представления базы данных

Для удобства и повышения производительности при обращении с базой данных были разработаны следующие представления.

Представление AppRoleTable\_view используется для выхода в приложение под пользователем или администратором и связки таблиц UserTable и AppRoleTable и возвращает следующие столбцы: UserLogin (логин пользователя), UserPassword (пароль пользователя), RoleName (роль пользователя в приложении).

Представление UserTeam\_view объединяет таблицы TeamTable, UserTable и UserTeamPrivs. Возвращает следующие столбцы: TeamID, TeamName, TeamIcon, UserID, UserLogin, UserTeamPrivName. Оно используется для отображения команд пользователя.

Представление TaskTeam\_view объединяет таблицы TaskTable, TaskPriorities и TeamTable. Возвращает следующие столбцы: TaskID, TaskTitle, CreationDate, DeadlineDate, TaskPriorityName, TaskStatusName, TaskDescription, TeamID, TeamName. Оно содержит информацию о каждой задачах для команды.

* 1. Пользователи базы данных

В разработанной базе данных присутствует один пользователь, который является хозяином схемы – DBNoto. Создание данного пользователя описано в листинге 2.24.

CREATE USER DBNoto IDENTIFIED BY Pa$$w0rd

DEFAULT TABLESPACE Users QUOTA UNLIMITED ON Users;

GRANT CONNECT TO DBNoto;

GRANT CREATE TABLE TO DBNoto;

GRANT CREATE SEQUENCE TO DBNoto;

GRANT CREATE VIEW TO DBNoto;

GRANT CREATE INDEXTYPE TO DBNoto;

GRANT CREATE PROCEDURE TO DBNoto;

GRANT CREATE TRIGGER TO DBNoto;

GRANT CREATE SESSION TO DBNoto;

GRANT CREATE JOB TO DBNoto;

GRANT EXECUTE ON SYS.dbms\_crypto TO dbnoto;

Листинг 2.24 – Создание пользователя базы данных DBNoto

Он имеет грант на подключение к базе данных, грант на выборку, добавление, изменение и удаление данных из всех таблиц, описанных в пункте 2.1, и из всех представлений, описанных в пункте 2.3. Он может создавать таблицы, процедуры, представления и сессии. Также ему выдан грант на выполнение процедур и функций стандартного пакета dbms\_crypto.

1. Обоснование технических приемов программирования

При разработке программного средстве в качестве программной платформы был выбран язык программирования C#. Проект разработан с использованием .NET Framework 4.7.2. Для взаимодействия с сервером Oracle 21XE нужно было с помощью диспетчера пакетов NuGet скачать пакет Oracle.ManagedDataAccess. Интерфейс представлен с использованием Windows Presentation Foundation (WPF). Все эти средства используются для реализации поставленных задач.

Пакет Oracle.ManagedDataAccess предоставляет драйвер, позволяющий подключаться к серверу. В листинге 3.1. изображена конфигурация подключения к серверу.

OracleConnection con = new OracleConnection();

public Authorization(){

InitializeComponent();

con.ConnectionString = "DATA SOURCE=localhost:1521/xe; PERSIST SECURITY INFO=True; USER ID=DBNoto; PASSWORD=Pa$$w0rd";

}

Листинг 3.1 – Строка подключения

Большая часть взаимодействия с базой данных происходит через хранимые процедуры. На листинге 3.2 можно увидеть пример того, как в C# вызываются хранимые на сервере Oracle процедуры. В качестве примера изображена процедура авторизации пользователя в приложении.

OracleCommand cmd = con.CreateCommand();

cmd.CommandText = "DBNoto.LOG\_IN\_USER";

cmd.CommandType = CommandType.StoredProcedure;

cmd.Parameters.Add("p\_user\_login", OracleDbType.Varchar2, 30).Value = textBoxLogin.Text.Trim();

cmd.Parameters.Add("p\_user\_password", OracleDbType.Varchar2, 30).Value = textBoxPassword.Text.Trim();

cmd.Parameters.Add("o\_user\_id", OracleDbType.Int32, 10);

cmd.Parameters["o\_user\_id"].Direction = ParameterDirection.Output;

cmd.Parameters.Add("o\_user\_login", OracleDbType.Varchar2, 30);

cmd.Parameters["o\_user\_login"].Direction = ParameterDirection.Output;

cmd.Parameters.Add("o\_user\_role", OracleDbType.Varchar2, 30);

cmd.Parameters["o\_user\_role"].Direction = ParameterDirection.Output;

cmd.Parameters.Add("o\_user\_name", OracleDbType.Varchar2, 30);

cmd.Parameters["o\_user\_name"].Direction = ParameterDirection.Output;

cmd.Parameters.Add("o\_user\_lastname", OracleDbType.Varchar2, 30);

cmd.Parameters["o\_user\_lastname"].Direction = ParameterDirection.Output;

cmd.Parameters.Add("o\_user\_phonenumber", OracleDbType.Varchar2, 13);

cmd.Parameters["o\_user\_phonenumber"].Direction = ParameterDirection.Output;

Продолжение листинга 3.2

cmd.Parameters.Add("o\_user\_email", OracleDbType.Varchar2, 30);

cmd.Parameters["o\_user\_email"].Direction = ParameterDirection.Output;

cmd.ExecuteNonQuery();

Листинг 3.2 – Вызов процедуры авторизации

В случаях, когда используются непосредственно SELECT запросы, используются конструкции подготовленных заявлений. Они позволяют передать в запрос параметры, не допустив SQL-инъекций.

Рассмотрим этот сценарий на примере запроса, используемого для вывода плиток с задачами. Данную конструкцию можно увидеть на рисунке 3.3.

con.Open();

OracleCommand cmd = con.CreateCommand();

cmd.CommandText = "SELECT \* FROM (select TaskID, TaskTitle, CreationDate, DeadlineDate, TaskPriorityName, TaskStatusName, TaskDescription, row\_number() over (ORDER BY TaskID ASC) rn from DBNoto.TaskTeam\_view WHERE TeamID = " + DataWorker.CurrentTeam.teamId + ") where rn between :n and :m ORDER BY TaskID ASC";

cmd.Parameters.Add(new OracleParameter("n", rowCounter));

cmd.Parameters.Add(new OracleParameter("m", rowCounter + rowMargin));

cmd.CommandType = CommandType.Text;

OracleDataReader reader = cmd.ExecuteReader();

taskList.Children.Clear();

while (reader.Read())

{

TaskUC task = new TaskUC(reader.GetInt16(0), reader.GetString(1), reader.GetString(2), reader.GetString(3), reader.GetString(4), reader.GetString(5), reader.GetString(6));

taskList.Children.Add(task);

}

con.Close();

Листинг 3.3 – Защищенный от SQL-инъекций SELECT-запрос

Сценарий с командой UPDATE будет рассмотрен в разделе с технологией базы данных.

1. Описание технологий

В данном разделе будут описаны технологии, использованные в процессе разработки приложения.

* 1. Репликация данных

В процессе разработки проекта была использована технология: репликации данных между серверами СУБД

Словарь определяет репликацию как процесс поддержания двух (или более) наборов данных в согласованном состоянии. Что такое «согласованное состояние наборов данных» – отдельный большой вопрос, поэтому переформулируем определение проще: процесс изменения одного набора данных, называемого репликой, в ответ на изменения другого набора данных, называемого основным. Совсем не обязательно наборы при этом будут одинаковыми.

Поддержка репликации баз данных – одна из важнейших задач администратора: почти у каждой сколько-нибудь важной базы данных есть реплика, а то и не одна.

Среди задач, решаемых репликацией, можно назвать как минимум:

* поддержку резервной базы данных на случай потери основной;
* снижение нагрузки на базу за счёт переноса части запросов на реплики;
* перенос данных в архивные или аналитические системы.

Можно выделить три подхода к репликации:

* блочная репликация на уровне системы хранения данных;
* физическая репликация на уровне СУБД;
* логическая репликация на уровне СУБД.

**Блочная репликация**

При блочной репликации каждая операция записи выполняется не только на основном диске, но и на резервном. Таким образом тому на одном массиве соответствует зеркальный том на другом массиве, с точностью до байта повторяющий основной том.

К достоинствам такой репликации можно отнести простоту настройки и надёжность. Записывать данные на удалённый диск может либо дисковый массив, либо нечто (устройство или программное обеспечение), стоящее между хостом и диском.

Дисковые массивы могут быть дополнены опциями, позволяющими включить репликацию. Название опции зависит от производителя массива.

Если дисковый массив не способен реплицировать данные, между хостом и массивом может быть установлен агент, осуществляющей запись на два массива сразу. Агент может быть, как отдельным устройством (EMC VPLEX), так и программным компонентом (HPE PeerPersistence, Windows Server Storage Replica, DRBD). В отличие от дискового массива, который может работать только с таким же массивом или, как минимум, с массивом того же производителя, агент может работать с совершенно разными дисковыми устройствами.

Главное назначение блочной репликации – обеспечение отказоустойчивости. Если база данных потеряна, то можно перезапустить её с использованием зеркального тома.

Блочная репликация хороша своей универсальностью, но за универсальность приходится платить.

Во-первых, никакой сервер не может работать с зеркальным томом, поскольку его операционная система не может управлять записью на него; с точки зрения наблюдателя данные на зеркальном томе появляются сами собой. В случае аварии (отказ основного сервера или всего ЦОДа, где находится основной сервер) следует остановить репликацию, размонтировать основной том и смонтировать зеркальный том. Как только появится возможность, следует перезапустить репликацию в обратном направлении.

В случае использования агента все эти действия выполнит агент, что упрощает настройку, но не уменьшает время переключения.

Во-вторых, сама СУБД на резервном сервере может быть запущена только после монтирования диска. В некоторых операционных системах, например, в Solaris, память под кеш при выделении размечается, и время разметки пропорционально объёму выделяемой памяти, то есть старт экземпляра будет отнюдь не мгновенным. Плюс ко всему кеш после рестарта будет пуст.

В-третьих, после запуска на резервном сервере СУБД обнаружит, что данные на диске неконсистентны, и нужно потратить значительное время на восстановление с применением журналов повторного выполнения: сначала повторить те транзакции, результаты которых сохранились в журнале, но не успели сохраниться в файлы данных, а потом откатить транзакции, которые к моменту сбоя не успели завершиться.

Блочная репликация не может использоваться для распределения нагрузки, а для обновления хранилища данных используется похожая схема, когда зеркальный том находится в том же массиве, что и основной. У EMC и HP эта схема называется BCV, только EMC расшифровывает аббревиатуру как Business Continuance Volume, а HP – как Business Copy Volume. У IBM на этот случай нет специальной торговой марки, эта схема так и называется – «mirrored volume».

В массиве создаются два тома, и операции записи синхронно выполняются на обоих (A). В определённое время зеркало разрывается (B), то есть тома становятся независимыми. Зеркальный том монтируется к серверу, выделенному для обновления хранилища, и на этом сервере поднимается экземпляр базы данных. Экземпляр будет подниматься так же долго, как и при восстановлении с помощью блочной репликации, но это время может быть существенно уменьшено за счёт разрыва зеркала в период минимальной нагрузки. Дело в том, что разрыв зеркала по своим последствиям эквивалентен аварийному завершению СУБД, а время восстановление при аварийном завершении существенно зависит от количества активных транзакций в момент аварии. База данных, предназначенная для выгрузки, доступна как на чтение, так и на запись. Идентификаторы всех блоков, изменённых после разрыва зеркала как на основном, так и на зеркальном томе, сохраняются в специальной области Block Change Tracking – BCT.

После окончания выгрузки зеркальный том размонтируется (С), зеркало восстанавливается, и через некоторое время зеркальный том вновь догоняет основной и становится его копией.

**Физическая репликация**

Журналы (redo log или write-ahead log) содержат все изменения, которые вносятся в файлы базы данных. Идея физической репликации состоит в том, что изменения из журналов повторно выполняются в другой базе (реплике), и таким образом данные в реплике повторяют данные в основной базе байт-в-байт.

Возможность использовать журналы базы данных для обновления реплики появилась в релизе Oracle 7.3, который вышел в 1996 году, а уже в релизе Oracle 8i доставка журналов с основной базы в реплику была автоматизирована и получила название DataGuard. Технология оказалась настолько востребованной, что сегодня механизм физической репликации есть практически во всех современных СУБД.

Опыт показывает, что если использовать сервер только для поддержания реплики в актуальном состоянии, то ему достаточно примерно 10% процессорной мощности сервера, на котором работает основная база.

Журналы СУБД не предназначены для использования вне этой платформы, их формат не документируется и может меняться без предупреждения. Отсюда совершенно естественное требование, что физическая репликация возможна только между экземплярами одной и той же версии одной той же СУБД. Отсюда же возможные ограничения на операционную систему и архитектуру процессора, которые тоже могут влиять на формат журнала.

Естественно, никаких ограничений на модели СХД физическая репликация не накладывает. Более того, файлы в базе-реплике могут располагаться совсем по-другому, чем на базе-источнике – надо лишь описать соответствие между томами, на которых лежат эти файлы.

Oracle DataGuard позволяет удалить часть файлов из базы-реплики – в этом случае изменения в журналах, относящиеся к этим файлам, будут проигнорированы.

Физическая репликация базы данных имеет множество преимуществ перед репликацией средствами СХД:

объём передаваемых данных меньше за счёт того, что передаются только журналы, но не файлы с данными; эксперименты показывают уменьшение трафика в 5-7 раз;

переключение на резервную базу происходит значительно быстрее: экземпляр-реплика уже поднят, поэтому при переключении ему нужно лишь откатить активные транзакции; более того, к моменту сбоя кеш реплики уже прогрет;

на реплике можно выполнять запросы, сняв тем самым часть нагрузки с основной базы. В частности, реплику можно использовать для создания резервных копий.

Возможность читать данные с реплики появилась в 2007 году в релизе Oracle 11g – именно на это указывает эпитет «active», добавленный к названию технологии DataGuard. В других СУБД возможность чтения с реплики также есть, но в названии это никак не отражено.

Запись данных в реплику невозможна, поскольку изменения в неё приходят побайтно, и реплика не может обеспечить конкурентное исполнение своих запросов. Oracle Active DataGuard в последних релизах разрешает запись в реплику, но это не более чем «сахар»: на самом деле изменения выполняются на основной базе, а клиент ждёт, пока они докатятся до реплики.

В случае повреждения файла в основной базе можно просто скопировать соответствующий файл с реплики (прежде, чем делать такое со своей базой, внимательно изучите руководство администратора!). Файл на реплике может быть не идентичен файлу в основной базе: дело в том, что когда файл расширяется, новые блоки в целях ускорения ничем не заполняются, и их содержимое случайно. База может использовать не всё пространство блока (например, в блоке может оставаться свободное место), но содержимое использованного пространства совпадает с точностью до байта.

Физическая репликация может быть как синхронной, так и асинхронной. При асинхронной репликации всегда есть некий набор транзакций, которые завершены на основной базе, но ещё не дошли до резервной, и в случае перехода на резервную базу при сбое основной эти транзакции будут потеряны. При синхронной репликации завершение операции commit означает, что все журнальные записи, относящиеся к данной транзакции, переданы на реплику. Важно понимать, что получение репликой журнала не означает применения изменений к данным. При потере основной базы транзакции не будут потеряны, но если приложение пишет данные в основную базу и считывает их из реплики, то у него есть шанс получить старую версию этих данных.

В PostgreSQL есть возможность сконфигурировать репликацию так, чтобы commit завершался только после применения изменений к данным реплики (опция synchronous\_commit = remote\_apply), а в Oracle можно сконфигурировать всю реплику или отдельные сессии, чтобы запросы выполнялись только если реплика не отстаёт от основной базы (STANDBY\_MAX\_DATA\_DELAY=0). Однако всё же лучше проектировать приложение так, чтобы запись в основную базу и чтение из реплик выполнялись в разных модулях.

При поиске ответа на вопрос, какой режим выбрать, синхронный или асинхронный, нам на помощь приходят маркетологи Oracle. DataGuard предусматривает три режима, каждый из которых максимизирует один из параметров – сохранность данных, производительность, доступность – за счёт остальных:

* Maximum performance: репликация всегда асинхронная;
* Maximum protection: репликация синхронная; если реплика не отвечает, commit на основной базе не завершается;
* Maximum availability: репликация синхронная; если реплика не отвечает, то репликация переключается в асинхронный режим и, как только связь восстанавливается, реплика догоняет основную базу и репликация снова становится синхронной.

Несмотря на бесспорные преимущества репликации средствами БД над блочной репликацией, администраторы во многих компаниях, особенно со старыми традициями надёжности, до сих пор очень неохотно отказываются от блочной репликации. Причин у этого две.

Во-первых, в случае репликации средствами дискового массива трафик идёт не по сети передачи данных (LAN), а по сети хранения данных (Storage Area Network). Зачастую в инфраструктурах, построенных давно, SAN гораздо надёжнее и производительнее, чем сеть передачи данных.

Во-вторых, синхронная репликация средствами СУБД стала надёжной относительно недавно. В Oracle прорыв произошёл в релизе 11g, который вышел в 2007 году, а в других СУБД синхронная репликация появилась ещё позже. Конечно, 10 лет по меркам сферы информационных технологий – срок не такой уж маленький, но когда речь идёт о сохранности данных, некоторые администраторы до сих пор руководствуются принципом «как бы чего не вышло».

**Логическая репликация**

Все изменения в базе данных происходят в результате вызовов её API – например, в результате выполнения SQL-запросов. Очень заманчивой кажется идея выполнять одну и ту же последовательность запросов на двух разных базах. Для репликации необходимо придерживаться двух правил:

Нельзя начинать транзакцию, пока не завершены все транзакции, которые должны закончиться раньше. Так на рисунке ниже нельзя запускать транзакцию D, пока не завершены транзакции A и B.

Нельзя завершать транзакцию, пока не начаты все транзакции, которые должны закончиться до завершения текущей транзакции. Так на рисунке ниже даже если транзакция B выполнилась мгновенно, завершить её можно только после того, как начнётся транзакция C.

Репликация команд (statement-based replication) реализована, например, в MySQL. К сожалению, эта простая схема не приводит к появлению идентичных наборов данных – тому есть две причины.

Во-первых, не все API детерминированы. Например, если в SQL-запросе встречается функция now() или sysdate(), возвращающая текущее время, то на разных серверах она вернёт разный результат – из-за того, что запросы выполняются не одновременно. Кроме того, к различиям могут привести разные состояния триггеров и хранимых функций, разные национальные настройки, влияющие на порядок сортировки, и многое другое.

Во-вторых, репликацию, основанную на параллельном исполнении команд, невозможно корректно приостановить и перезапустить.

Если репликация остановлена в момент T1 транзакция B должна быть прервана и откачена. При перезапуске репликации исполнение транзакции B может привести реплику к состоянию, отличному от состояния базы-источника: на источнике транзакция B началась до того, как закончилась транзакция A, а значит, она не видела изменений, сделанных транзакцией A.

Репликация запросов может быть остановлена и перезапущена только в момент T2, когда в базе нет ни одной активной транзакции. Разумеется, на сколько-нибудь нагруженной промышленной базе таких моментов не бывает.

Обычно для логической репликации используют детерминированные запросы. Детерминированность запроса обеспечивается двумя свойствами:

запрос обновляет (или вставляет, или удаляет) единственную запись, идентифицируя её по первичному (или уникальному) ключу;

все параметры запроса явно заданы в самом запросе.

В отличие от репликации команд (statement-based replication) такой подход называется репликацией записей (row-based replication).

**Репликация триггерами**

Триггер – хранимая процедура, которая исполняется автоматически при каком-либо действии по модификации данных. Триггеру, который вызывается при изменении каждой записи, доступны ключ этой записи, а также старые и новые значения полей. При необходимости триггер может сохранять новые значения строк в специальную таблицу, откуда специальный процесс на стороне реплики будет их вычитывать. Объём кода в триггерах велик, поэтому существуют специальное программное обеспечение, генерирующее такие триггеры, например, «Репликация слиянием» (merge replication) – компонент Microsoft SQL Server или Slony-I – отдельный продукт для репликации PostgreSQL.

Сильные стороны репликации триггерами:

* независимость от версий основной базы и реплики;
* широкие возможности преобразования данных.

Недостатки:

* нагрузка на основную базу;
* большая задержка при репликации.

**Использование CDC**

Существует целый класс программного обеспечения, предназначенного для организации логической репликации. Это ПО называется CDC, change data capture. Вот список наиболее известных платформ этого класса:

* Oracle GoldenGate (компания GoldenGate приобретена в 2009 году);
* IBM InfoSphere Data Replication (ранее – InfoSphere CDC; ещё ранее – DataMirror Transformation Server, компания DataMirror приобретена в 2007 году);
* VisionSolutions DoubleTake/MIMIX (ранее – Vision Replicate1);
* Qlik Data Integration Platform (ранее – Attunity);
* Informatica PowerExchange CDC;
* Debezium;
* StreamSets Data Collector

В задачу платформы входит чтение журналов базы данных, преобразование информации, передача информации на реплику и применение. Как и в случае репликации средствами самой СУБД, журнал должен содержать информацию об изменённых полях. Использование дополнительного приложения позволяет «на лету» выполнять сложные преобразования реплицируемых данных и строить достаточно сложные топологии репликации.

Сильные стороны:

* возможность репликации между разными СУБД, в том числе загрузка данных в отчётные системы;
* широчайшие возможности обработки и преобразования данных;
* минимальный трафик между узлами – платформа отсекает ненужные данные и может сжимать трафик;
* встроенные возможности мониторинга состояния репликации.

Недостатков не так много:

* увеличение объёма журналов, как при логической репликации средствами СУБД;
* новое ПО – сложное в настройке и/или с дорогими лицензиями.

Именно CDC-платформы традиционно используются для обновления корпоративных хранилищ данных в режиме, близком к реальному времени.

В нашем случае была использована репликация триггерами. Подключение к другой базе данных осуществлялось с помощью dblink. Скрипт создания объекта dblink показан в листинге 4.1.

CREATE PUBLIC DATABASE LINK CWCON

CONNECT TO DBNoto

IDENTIFIED BY Pa$$w0rd

USING '25.49.158.45:1521/xe';

Листинг 4.1 – Скрипт создания объекта dblink

Далее были созданы триггеры для каждой таблицы, которые при удалении, изменении или добавлении строки производили эти же изменение в удалённом сервере. Пример такого триггера для таблицы Users показан в листинге 4.2.

create or replace trigger UserTableTrigger

AFTER

INSERT OR DELETE OR UPDATE

ON UserTable FOR EACH ROW

BEGIN

IF(inserting) THEN

Продолжение листинга 4.2

INSERT INTO UserTable@cwcon(USERLOGIN,USERPASSWORD,USERNAME,USERLASTNAME,USERPHONENUMBER,USEREMAIL,USERROLE,USERICON) VALUES(:NEW.USERLOGIN,:NEW.USERPASSWORD,:NEW.USERNAME,:NEW.USERLASTNAME,:NEW.USERPHONENUMBER,:NEW.USEREMAIL,:NEW.USERROLE,:NEW.USERICON);

END IF;

IF(updating) THEN

UPDATE UserTable@cwcon SET USERLOGIN = :NEW.USERLOGIN WHERE USERLOGIN = :OLD.USERLOGIN;

UPDATE UserTable@cwcon SET USERPASSWORD = :NEW.USERPASSWORD WHERE USERLOGIN = :OLD.USERLOGIN;

UPDATE UserTable@cwcon SET USERNAME = :NEW.USERNAME WHERE USERLOGIN = :OLD.USERLOGIN;

UPDATE UserTable@cwcon SET USERLASTNAME = :NEW.USERLASTNAME WHERE USERLOGIN = :OLD.USERLOGIN;

UPDATE UserTable@cwcon SET USERPHONENUMBER = :NEW.USERPHONENUMBER WHERE USERLOGIN = :OLD.USERLOGIN;

UPDATE UserTable@cwcon SET USEREMAIL = :NEW.USEREMAIL WHERE USERLOGIN = :OLD.USERLOGIN;

UPDATE UserTable@cwcon SET USERROLE = :NEW.USERROLE WHERE USERLOGIN = :OLD.USERLOGIN;

UPDATE UserTable@cwcon SET USERICON = :NEW.USERICON WHERE USERLOGIN = :OLD.USERLOGIN;

END IF;

IF(deleting) THEN

DELETE UserTable@cwcon WHERE USERLOGIN = :OLD.USERLOGIN;

END IF;

END;

Листинг 4.1 – Триггер для таблицы UserTable

После произведения вышеуказанных манипуляций была достигнута репликация данных между двумя серверами.

* 1. Мультимедийные типы данных

Для работы с данными большого объема 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 данном курсовом проекте технология мультимедийных типов данных отражена в необходимости хранить изображения, являющиеся иконками пользователей и команд. Речь идет о таблицах UserTable и TeamTable. Для хранения в поле изображений в СУБД применяется тип BLOB. Тип BLOB представляет из себя массив двоичных данных, поэтому процесс сводился к разбиению мультимедийных файлов на массив битов при загрузке их в базу данных, и к сборке изображения из массива битов на обратном пути.

OpenFileDialog openFileDialog = new OpenFileDialog()

{

Filter = "Image Files|\*.jpg;\*.png;"

};

if (openFileDialog.ShowDialog() == true)

{

imageName = openFileDialog.FileName;

image = PictureToByte(imageName);

UserProfile.userIcon = image;

}

openFileDialog = null;

Листинг 4.2 – Загрузка изображения и разбиение его на массив битов

Рассмотрим сначала процесс преобразования изображения в тип данных, понятный Oracle. Так как в хранимых процедурах нельзя передавать данные объемом более 32 Кб, процесс создания администратором нового альбома происходит в два этапа. Сначала вызывается хранимая процедура create\_album, после чего происходит добавление обложки к появившейся в базе данных строке.

Процесс выглядит следующим образом: по нажатию на кнопку для добавления обложки открывается стандартное диалоговое окно Windows, с помощью которого пользователь волен выбрать нужное изображение с расширением .JPG или .PNG. Последующие действия можно увидеть на рисунке 3.1. Для удобства администратора загруженное изображение дополнительно отображается в окне, чтобы можно было убедиться в правильности выбора.

После превращения изображения в массив битов дело остается за малым, а именно: используя драйвер Managed Data Access мы отправляем UPDATE-запрос для вставки изображения в существующую строку, где с помощью подготовленного заявления передаем этот массив для вставки в поле UserIcon. Это можно увидеть в листинге 4.3.

con.Open();

OracleCommand cmd2 = con.CreateCommand();

OracleTransaction txn;

txn = con.BeginTransaction(IsolationLevel.ReadCommitted);

cmd2.Transaction = txn;

cmd2.CommandText = "UPDATE DBNoto.UserTable " +

"SET " +

"UserIcon = :ImageFront " +

"WHERE UserID = UPPER('" + UserProfile.userId + "')";

cmd2.Parameters.Add(":ImageFront", OracleDbType.Blob);

cmd2.Parameters[":ImageFront"].Value = image;

cmd2.ExecuteNonQuery();

txn.Commit();

con.Close();

Листинг 4.3 – Обновление изображения в базе данных

На этом этапе изображение успешно загружено в базу данных. Также в своем приложении я обращаюсь к этому изображению для отображения иконок пользователей.

Процесс получения значения поля UserIcon изображен в листинге 4.4. Он заключается в проcтейшем SELECT-запросе к таблице и чтении результата в качестве массива битов. После этого с помощь. Объекти BitmapImage() мы превращаем полученный массив битов в битовую карту. То есть, непосредственно в изображения.

con.Open();

OracleCommand cmd2 = con.CreateCommand();

cmd2.CommandText = "SELECT UserIcon FROM DBNoto.UserTable WHERE UserID = " + UserProfile.userId; ;

cmd2.CommandType = CommandType.Text;

OracleDataReader reader = cmd2.ExecuteReader();

while (reader.Read())

{

UserProfile.userIconImg = LoadImage(reader.GetValue(0) as byte[]);

}

con.Close();

Листинг 4.4 – Обновление изображения в базе данных

Таким образом была реализована работа с изображениями в базе данных.

* 1. Средства диагностики

Oracle предоставляет очень богатый функционал для разработчика, его отличительной чертой является также ограниченность в бесплатном вспомогательном программном обеспечении, в отличие от MS SQL.

Для диагностики работоспособности использовали EXPLAIN. Применение изображено в листинге 4.5.

EXPLAIN PLAN FOR select \* from UserTable;

SELECT \* FROM TABLE(DBMS\_XPLAN.DISPLAY());

Листинг 4.5 – Применение EXPLAIN

Результат выполнения EXPLAIN для select-запроса к таблице UserTable представлен в листинге 4.6.

Plan hash value: 1904374143

----------------------------------------------------------------------

|Id| Operation | Name | Rows |Bytes|Cost (%CPU)| Time |

----------------------------------------------------------------------

|0 |SELECT STATEMENT | |100000| 73M| 238 (0)| 00:00:01 |

|1 |TABLE ACCESS FULL| USERTABLE |100000| 73M| 238 (0)| 00:00:01 |

----------------------------------------------------------------------

Note

-----

- dynamic statistics used: dynamic sampling (level=2) SELECT \* FROM TABLE(DBMS\_XPLAN.DISPLAY());

Листинг 4.6 – Результат выполнения EXPLAIN

1. Руководство пользователя, тестирование

Разработанное в рамках курсового проекта приложение поддерживает два вида ролей у пользователя: администратор и рядовой пользователь. Роль администратора дополняет функционал обычного пользователя, поэтому будет уместно в рамках руководства осветить возможности аккаунта администратора.

При запуске приложения открывается окно авторизации (рисунок 5.1), где зарегистрированному пользователю следует ввести свой логин и пароль.

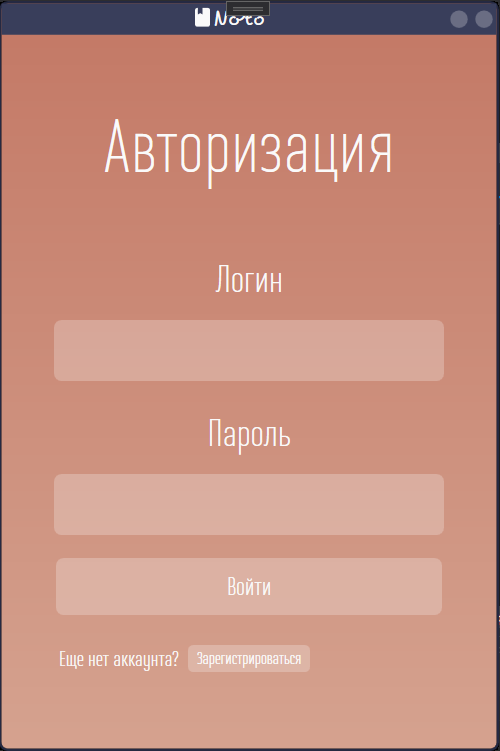


Рисунок 5.1 – Окно авторизации

При неправильном вводе пароля пользователю выводится сообщение о неправильном пароле (рисунок 5.2).

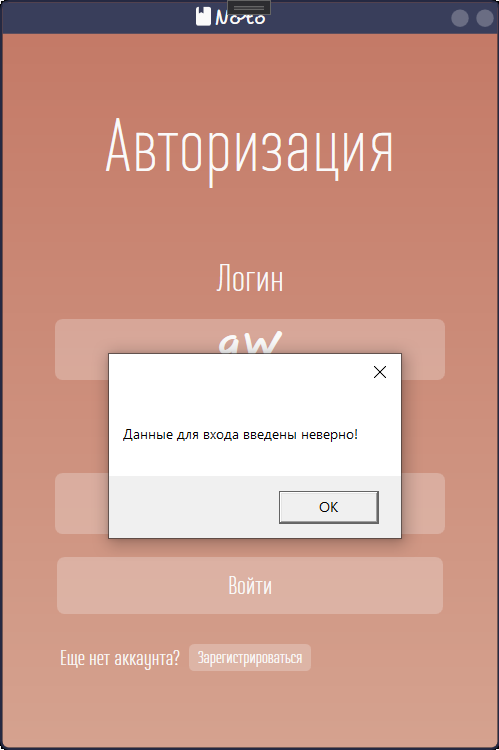


Рисунок 5.2 – Окно авторизации, неверный логин или пароль

При успешной авторизации пользователь будет перенаправлен на главную страницу приложения.

Неавторизованный пользователь не может зайти на главную страницу, пока не авторизуется. Для этого следует нажать на кнопку «Зарегистрироваться» в окне авторизации. Окно авторизации изображено на рисунке 5.3.

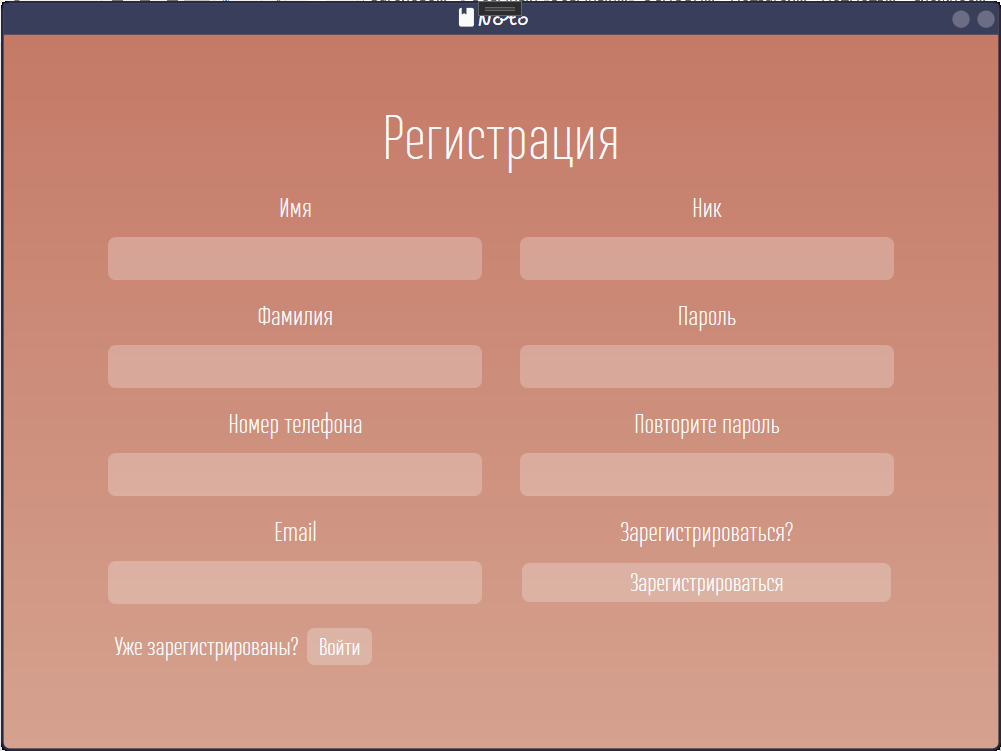


Рисунок 5.3 – Окно регистрации

Допустим, что пользователь успешно зарегистрирован. Далее, пройдя авторизацию под аккаунтом администратора, мы попадем на домашнюю страницу приложения (рисунок 5.4).

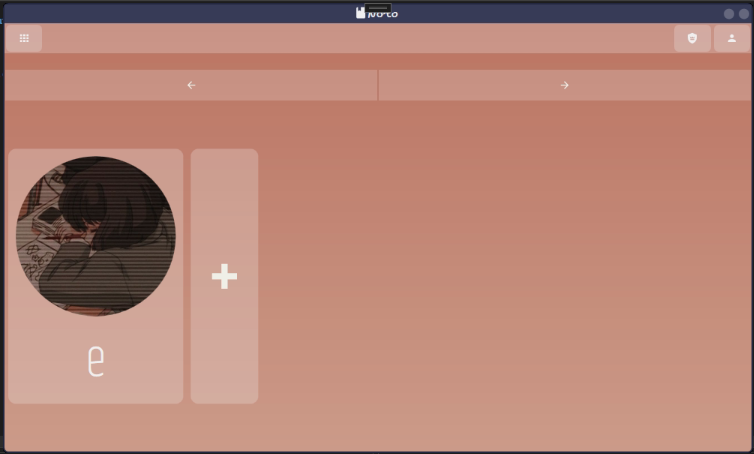


Рисунок 5.4 – Домашняя страница приложения (администратор)

С аккаунта обычного пользователя можно осуществлять создание команд и управление созданными этим пользователем командами, создание заданий и управление ими, добавление пользователей в команду, просмотр заданий по дате дедлайна и графика, отражающего количество заданий по заданному промежутку времени.

Администратор может делать то же, что и пользователь, с расширенным функционалом: администратор может просматривать список всех пользователей и команд, осуществлять их импорт и экспорт в XML, удалять их.

Страница «Администрация» содержит две кнопки, которые реализуют переключение между списками пользователей и команд. Также можно произвести экспорт или импорт отображаемых данных. По нажатию на кнопку «Export to XML» запустится процедура, экспортирующая полный список пользователей или команд в файл. По нажатию на кнопку «Import to XML» запустится процедура, импортирующая полный список пользователей или команд из файла. Данная страница представлена на рисунке 5.5.

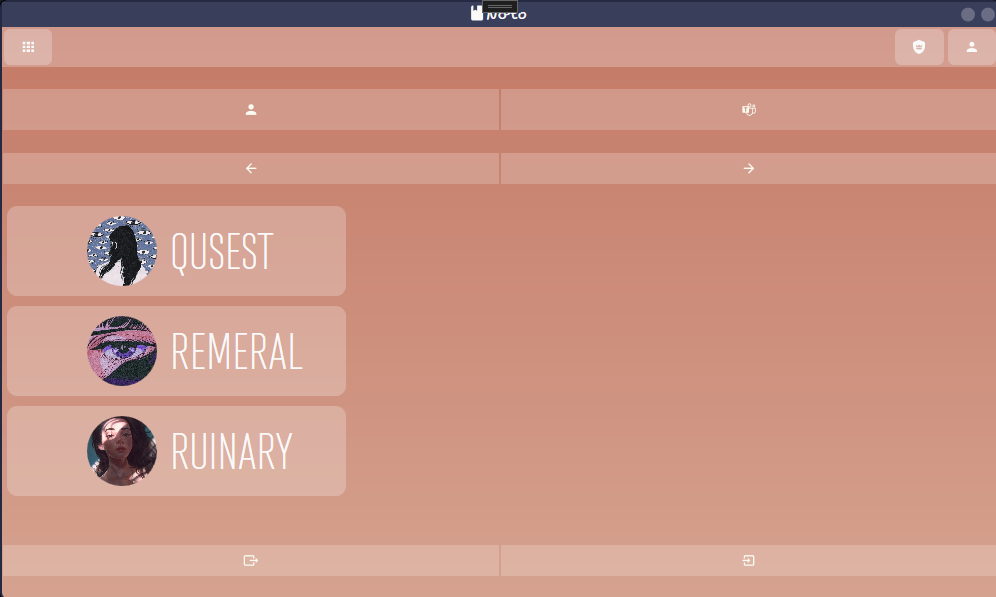


Рисунок 5.5 – Страница «Администрация»

Попав на домашнюю страницу, по умолчанию открывается вкладка «Команды». Если обратить внимание на меню, в нем также имеются вкладки «Профиля» и «Администратора», если вход осуществлен под учетной записью администратора.

Вкладка «Команды» содержит список заданий, который представлен плитками с информацией о данном задании. Любой пользователь может нажать на кнопку «Настройки» для определенного задания и изменить информацию. Это окно можно увидеть на рисунке 5.6.

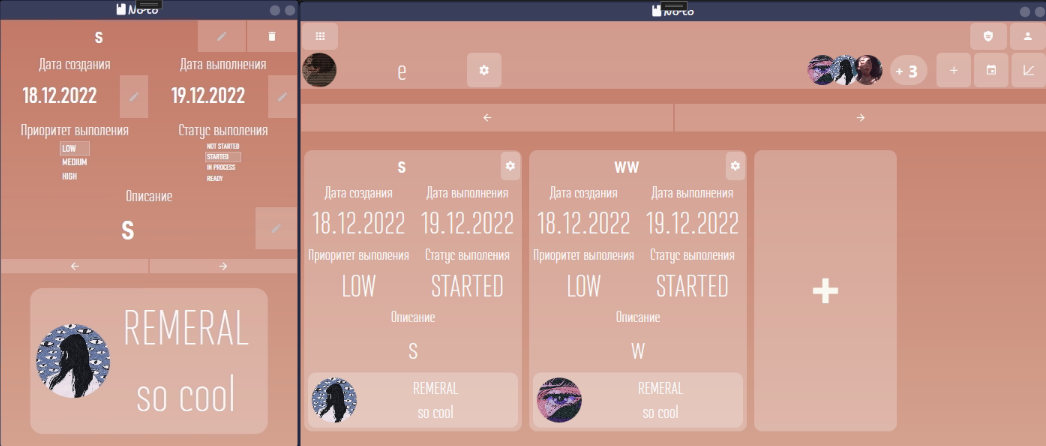


Рисунок 5.6 – Список заданий и настройки определенного задания

Пользователь, который является создателем данной команды, может открыть окно настроек для данной команды. Где можно изменить название команды и удалить пользователей из команды. Это окно представлено на рисунке 5.7.

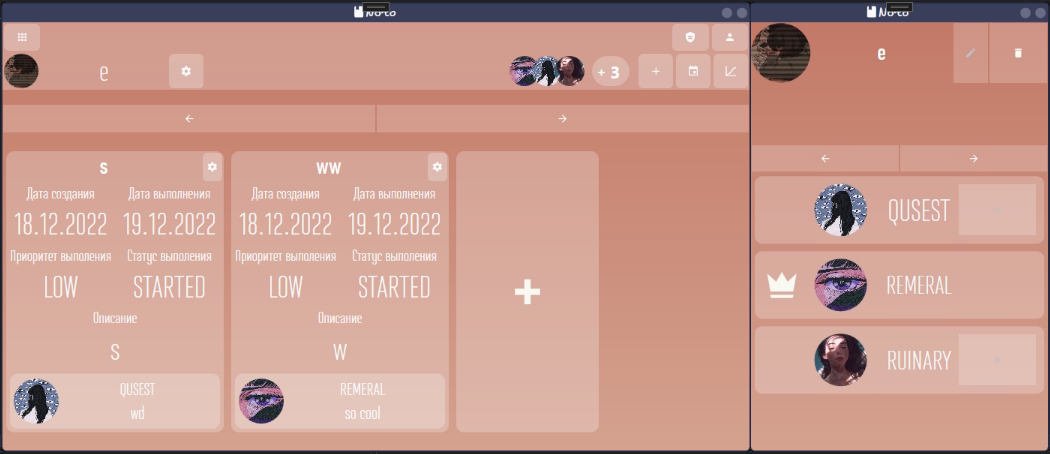


Рисунок 5.7 – Окна «Настройки команды» и «Задания команды» для владельца команды

Любой пользователь может нажать на кнопку «Календарь» и попадет на страницу календарем и произвести выборку заданий по заданной дате. Это окно можно увидеть на рисунке 5.8.

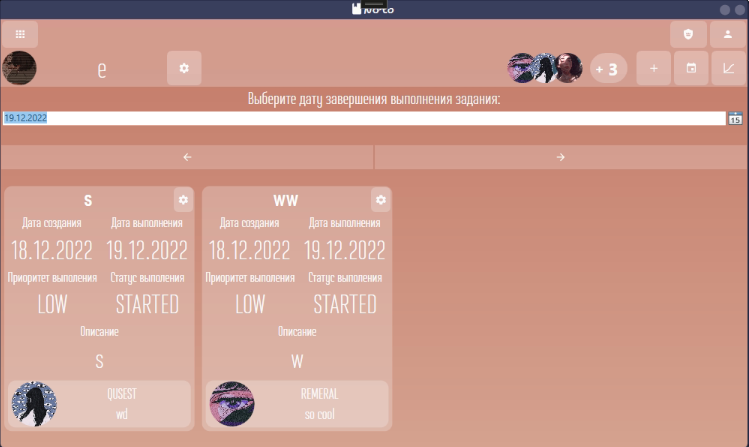


Рисунок 5.8 – Страница «Календарь»

Страница «График» содержит календарь, на котором можно выбрать диапазон дат и отобразится график, отображающий отношение количества заданий по определенной дате. Это окно можно увидеть на рисунке 5.9.



Рисунок 5.9 – Страница «График»

Любой пользователь может удалить свой аккаунт по желанию. При этом надо все данные, связанные с этим пользователем будут удалены.

На странице «Профиль» можно увидеть информацию о пользователе. Если открыта страница пользователя, от которого осуществлен вход в приложение, то есть возможность изменить данные этого пользователя. Эта страница представлена на рисунке 5.10.

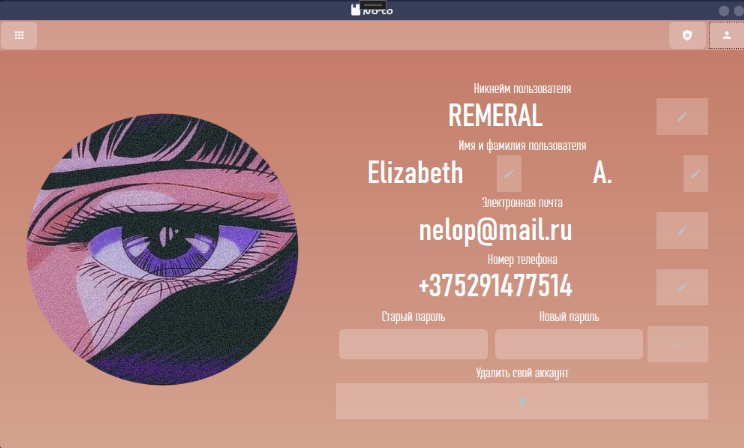


Рисунок 5.10 – Страница «Профиль»

Страница «Профиль» будет иметь вид, который можно увидеть на рисунке 5.11, если открыть профиль другого пользователя, а не того, от которого произведен вход.

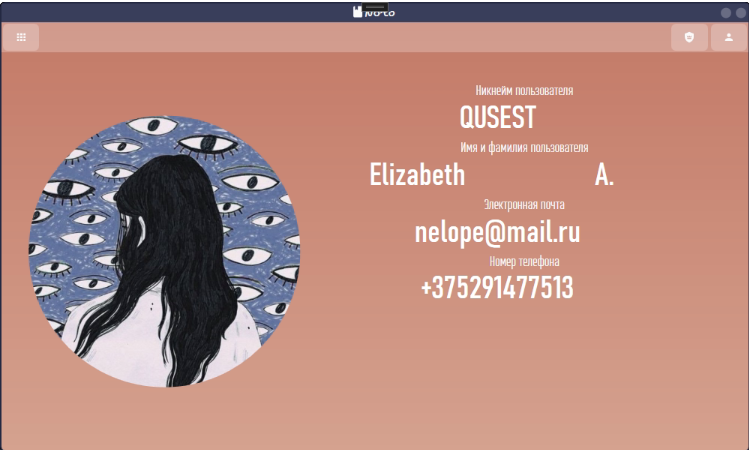


Рисунок 5.11 – Страница «Профиль»

В проекте был протестирован весь базовый функционал пользователя и администратора приложения. Все компоненты работают исправно.

Заключение

Результатом выполнения курсового проекта стало .NET приложение с использованием технологии WPF для операционной системы Windows. В ходе курсовой работы была разработана база данных в Oracle 21XE на тему «Управление проектами и задачами малых групп». Приложение является средством взаимодействия клиента с базой данных.

Были реализованы основные требования, указанные в листе задания вместе со следующими пунктами:

* регистрация и авторизация пользователей с учётом выданных им ролей;
* управление задачами команд;
* управление командами;
* добавление комментариев к задачам;
* просмотр графиков по задачам команды;
* просмотр задачам по определенному дню;
* изменение данных пользователя;
* пароли при регистрации шифруются;
* заполнение таблицы на 100 000 записей.

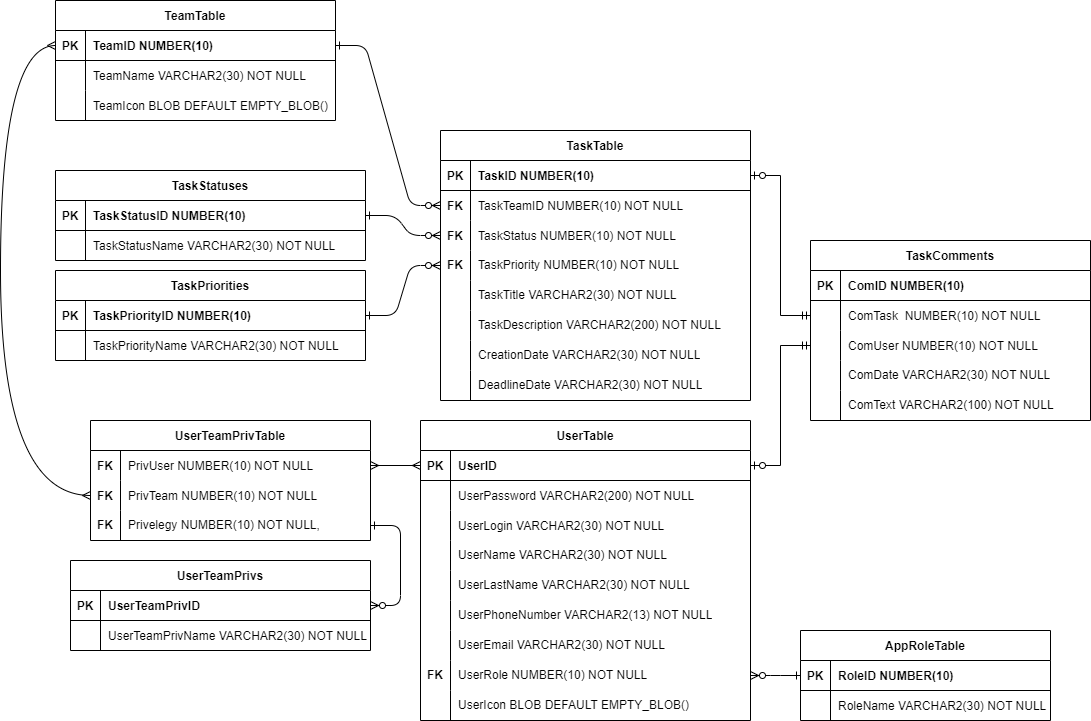
Протестировав все компоненты приложения, можно прийти к заключению, что все основные требования выполнены и приложение работает исправно.

Существует ряд приложений, имеющих функционал и дизайн более продуманней и можно сделать вывод, что в приложение еще можно разработать еще больше функций.

Список литературы

1. Блинова Е.А. Курс лекций по базам данных / Е.А. Блинова
2. Пацей, Н.В. Курс лекций по языку программирования C# / Н.В. Пацей. – Минск: БГТУ, 2016. – 175 с.
3. Oracle-dba.ru [Электронный ресурс] / Режим доступа: https://oracle-dba.ru – Дата доступа: 15.10.2022.
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: 16.10.2022.
5. DBMS\_XMLDOM [Электронный ресурс] / Режим доступа: https://docs.oracle.com/cd/B1930601/appdev.102/b14258/dxmldom.htm#i1076719 – Дата доступа: 18.10.2022.

Приложение А



Приложение Б

