

МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ
ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет Информационных Технологий
Кафедра Информационных систем и технологий
Специальность 1-40 01 01 «Программное обеспечение информационных технологий»
Специализация 1-40 01 01 10 «Программное обеспечение информационных технологий
(программирование интернет-приложений)»

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
к курсовому проекту на тему:**

Web-приложение для банка

Выполнил студент Макаров Алексей Игоревич
(Ф.И.О.)

Руководитель проекта ст.преп. Дубовик М.В.
(учен. степень, звание, должность, подпись, Ф.И.О.)

И.о. заведующего кафедрой ст.преп. Блинова Е.А.
(учен. степень, звание, должность, подпись, Ф.И.О.)

Курсовой проект защищен с оценкой _____

Содержание

Введение	3
1 Постановка задачи	3
1.1 Обзор аналогичных решений	3
1.2 Спецификация требований	5
2 Проектирование web-приложения	6
2.1 Проектирование вариантов использования	6
2.2 Структура web-приложения	6
2.3 Проектирование базы данных	7
2.4 Проектирование сервера web-приложения	11
3 Разработка web-приложения	13
3.1 Разработка бэкэнда	13
3.2 Разработка фронтэнда	16
4 Тестирование web-приложения.....	18
5 Руководство пользователя	22
Список используемых источников	29
Приложение А.....	30

Введение

В настоящее время веб-приложения становятся необходимостью для большого бизнеса при привлечении клиентов. В данном курсовом проекте будет рассмотрено создание веб приложения для банка на платформе Node.js с использованием языка программирования JavaScript. Данное приложение предназначено для обеспечения широкого спектра функциональности, необходимой для эффективного управления банковскими операциями и взаимодействия с клиентами.

Функциональные требования к веб-приложению включают в себя:

- Регистрацию и авторизацию пользователей с поддержкой ролей администратора и обычного пользователя.
- Возможность изменения информации о клиентах и напоминание о платежах по кредитам.
- Возможность открытия вкладов и поддержание связи между клиентом и сотрудником банка через чат.
- Предоставление калькулятора кредита и возможность подбора кредитов по критериям.
- Реализацию автоматического списания средств на сохраненные платежи.

Программное средство выполнено с применением асинхронного программирования для взаимодействия с базой данных и реализованным для различных платформ. Веб-приложение имеет асинхронный пользовательский интерфейс, при этом отображение, бизнес-логика и хранилище данных максимально независимы друг от друга для обеспечения возможности расширения функциональности.

В рамках проекта необходимо разработана диаграмма вариантов использования на основе UML, логическая схему базы данных и структурная схема приложения. Управление программой должно быть интуитивно понятным и удобным для пользователя.

1 Постановка задачи

1.1 Обзор аналогичных решений

В качестве первого аналогичного решения была рассмотрена платформа `ibank.belapb.by`, на которой размещены услуги интернет банкинга банка «Белагропром» Пример страницы представлен на рисунке 1.1.

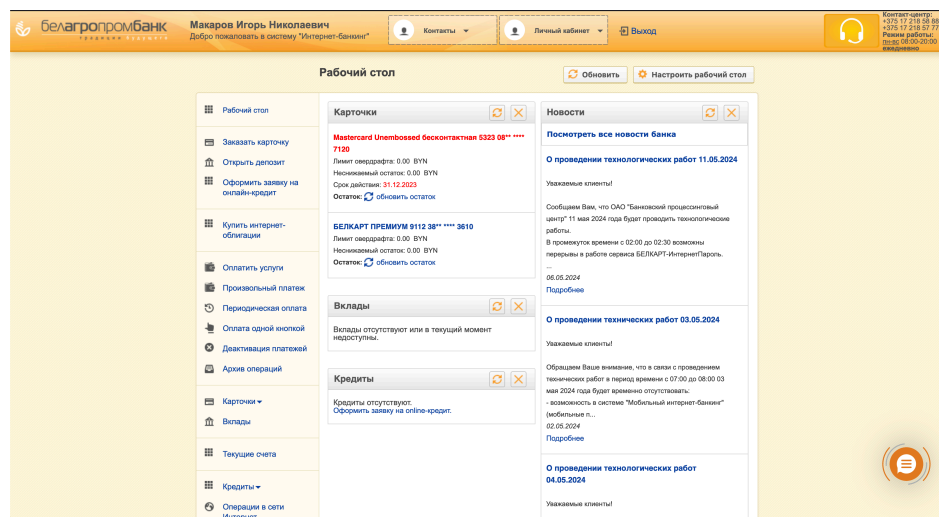


Рисунок 1.1 – Страница ibank.belapb.by

Пользователи могут открывать счета, брать кредиты, открывать банковские карты, а также оплачивать услуги и совершать переводы.

Преимуществом платформы являются простой процесс оформления счетов, возможность просмотра остатков по кредитным и дебетовым счетам, банковским картам. К недостаткам относится недостаточная возможность персонализации главной страницы, невозможность сохранить частые платежи на рабочем столе, отсутствие подсказок по навигации.

В качестве другого аналогичного решения была рассмотрена платформа «Интернет банкинг» от Беларусбанк, предлагающая более широкий спектр услуг. Платформа предоставляет возможность открытия вкладов, кредитов, покупки ценных бумаг, а также широкий спектр возможностей оплаты услуг, таких как оплата связи, переводы на карты других банков, заказ банковских карточек онлайн. Пример страницы платформы представлен на рисунке 1.2.

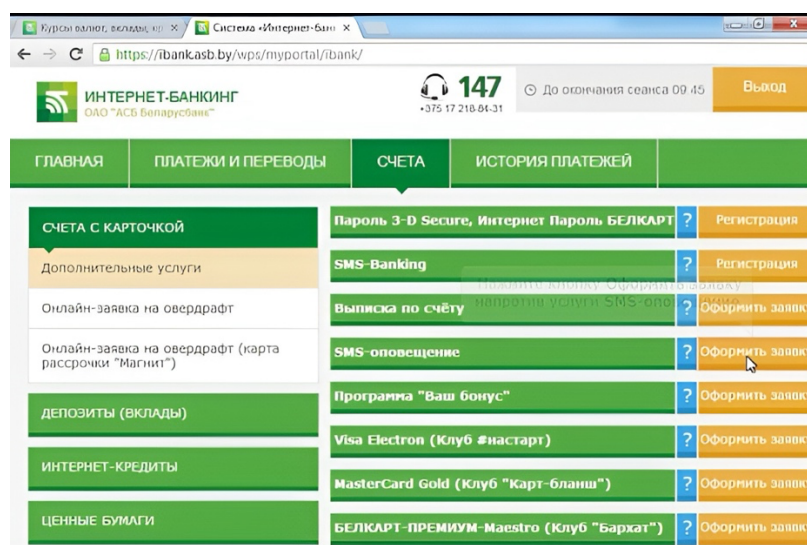


Рисунок 1.2 – Страница Интернет банкинг

К преимуществам платформы относится возможность сохранения частых

платежей в избранное, возможность запланировать и автоматизировать платежи, онлайн техподдержка, адаптивный дизайн.

1.2 Спецификация требований

На основе рассмотренных аналогичных решений были сформированы следующие требования к программному продукту:

- обеспечивать возможность регистрации и авторизации;
- поддерживать роли администратора и пользователя;
- позволять изменять информацию о клиентах;
- напоминать о платежах по кредитам;
- предоставлять возможность открытия вкладов;
- предоставлять возможность поддерживать связь между клиентом и работником банка при помощи чата;
- предоставлять калькулятор кредита по выбранным условиям;
- предоставлять возможность подбора кредитов по критериям;
- позволять устанавливать автоматическое списание средств на сохраненные платежи;

Неавторизованному пользователю доступна только страница авторизации. Авторизованным пользователям с ролью клиент доступны взятие кредитов, открытие вкладов, изменение информации о себе, создание запланированных платежей, открытие чата с техподдержкой.

Авторизованным пользователям с ролью работника доступны функции блокировки счета пользователя, общения с пользователем в чате техподдержки, создание типов кредитов, создание изменение и удаление типов вкладов, создание изменение и удаление типов кредитов. Создание и удаление новых типов счетов. Работнику не доступна возможность перевода денег со счета пользователя на другой счет, функция удаления пользователя, а также функция открытия вклада или взятия кредита.

Авторизованному пользователю с ролью администратор доступны все функции роли работник, а также удаление, изменение пользовательских данных, удаление счетов, вкладов, кредитов, блокировка счетов, общение с пользователями в чате технической поддержки, отправка уведомлений пользователям, регистрация новых пользователей.

Администратору не доступно взятие кредитов и открытие вкладов.

2 Проектирование web-приложения

2.1 Проектирование вариантов использования

Согласно сформулированным требованиям была создана диаграмма вариантов использования, представленная на рисунке 2.1.

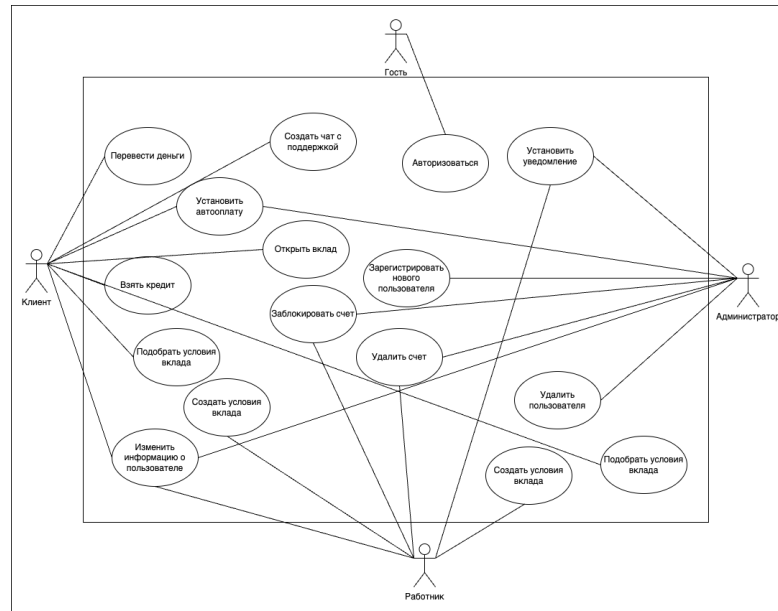


Рисунок 2.1 – Диаграмма вариантов использования

На данной диаграмме Гость – это пользователь который еще не прошел авторизацию. После регистрации ему выдается соответствующая роль в зависимости от которой пользователь получает набор доступных ему действий.

Клиенту доступны возможности подбора кредитов и вкладов по заданным им параметрам, открытие вклада или кредита, перевод денег с одного счета на другой, изменение личной информации, создание чата с технической поддержкой, установление уведомлений и запланированных платежей.

Работнику банка доступны создание и удаление условий кредитов, создание и удаление условий вкладов, присоединение к чату с технической поддержкой, изменение информации о пользователе, блокировка счетов.

Администратору доступны все функции выше описанные у работника, к ним добавлены удаление пользователей, удаление счетов, установка уведомлений для всех пользователей, участие в чате технической поддержкой, регистрация новых пользователей.

2.2 Структура web-приложения

Обобщённая структура web-приложения представлена на рисунке 2.2.

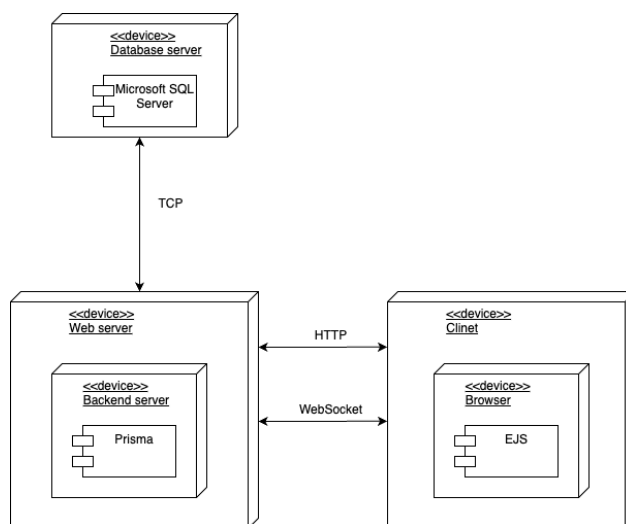


Рисунок 2.2 – Диаграмма развёртывания web-приложения

Согласно данной схеме, клиент и сервер находятся на разных устройствах. Клиент использует браузер для отправки запросов к web-серверу. Клиент и сервер могут обменяться сообщениями по протоколам HTTPS и WebSocket.

Web-сервер обрабатывает запросы при помощи сервера. В случае необходимости сервер отправляет запросы к базе данных, находящейся на отдельном устройстве в контейнере Docker и находящейся под управлением СУБД Microsoft SQL Server.

Для выполнения запросов к базе данных используется ORM Prisma.

2.3 Проектирование базы данных

Согласно схеме вариантов использования была создана база данных. Её логическая схема представлена на рисунке 2.3. Описание моделей представлено в приложении А.

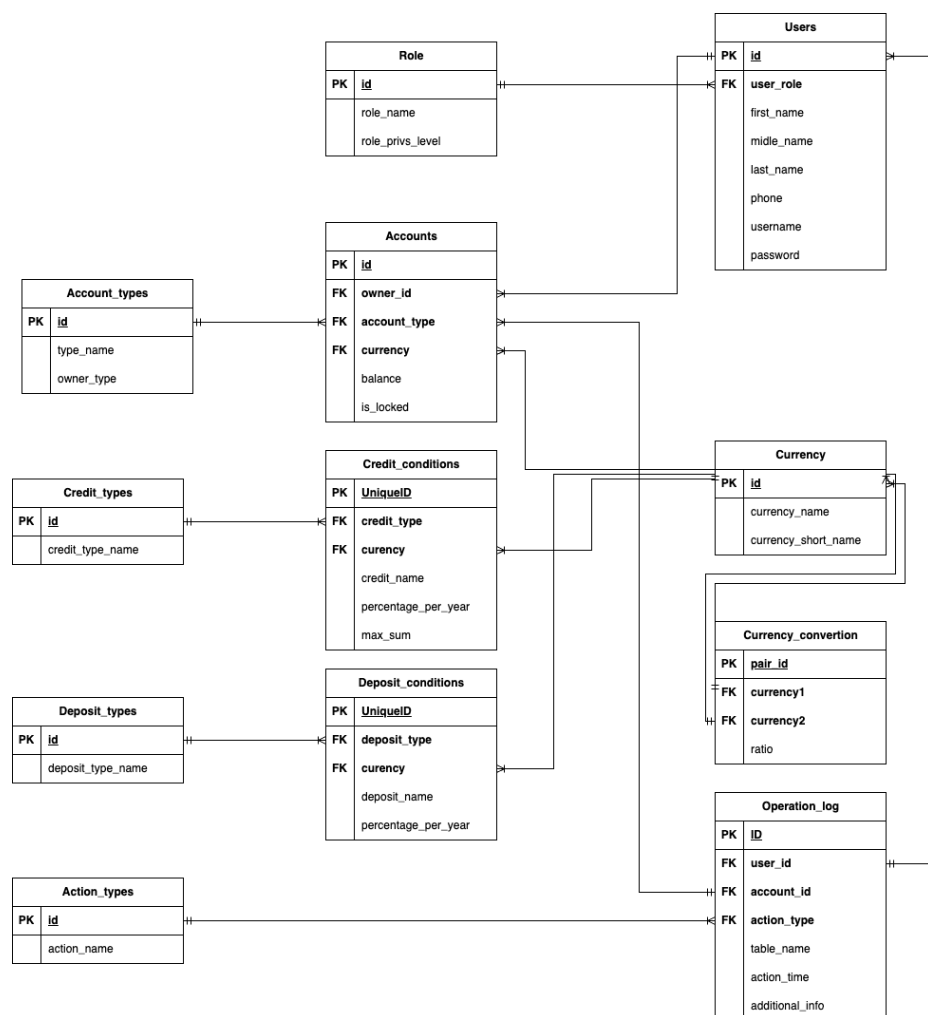


Рисунок 2.3 – Логическая схема базы данных

База данных содержит двенадцать таблиц, хранящих информацию о пользователях, счетах, кредитах, вкладах, операциях. Типы данных были выбраны согласно документации [1].

Таблица ROLE хранит информацию о ролях пользователей. Описание её столбцов представлено в таблице 2.1.

Таблица 2.1 – Описание таблицы ROLE

Название столбца	Тип данных	Описание
id	int	Идентификатор роли
role_name	NVARCHAR	Название роли
role_privs_level	INT	Уровень привилегий роли

Таблица USERS хранит информацию о пользователях. Описание её столбцов представлено в таблице 2.2.

Таблица 2.2 – Описание таблицы USERS

Название столбца	Тип данных	Описание
id	int	Идентификатор пользователя

first_name	NVARCHAR	Имя пользователя
midle_name	NVARCHAR	Отчество пользователя
last_name	NVARCHAR	Фамилия пользователя
phone	NCHAR	Телефон пользователя
user_role	int	Идентификатор роли пользователя
username	VARCHAR	Имя пользователя
passwd	VARCHAR	Пароль пользователя

Таблица ACCOUNT_TYPES хранит информацию о типах счетов которые может создать пользователь. Описание её столбцов представлено в таблице 2.3.

Таблица 2.3 – Описание таблицы ACCOUNT_TYPES

Название столбца	Тип данных	Описание
id	int	Идентификатор типа аккаунта
type_name	NVARCHAR	Название типа аккаунта
owner_type	int	Идентификатор типа владельца аккаунта

Таблица CURRENCY хранит информацию о валютах существующих в базе данных. Описание её столбцов представлено в таблице 2.4.

Таблица 2.4 – Описание таблицы CURRENCY

Название столбца	Тип данных	Описание
id	int	Идентификатор валюты
curency_name	NVARCHAR	Название валюты
currency_short_name	NVARCHAR	Краткое название валюты

Таблица ACCOUNTS хранит информацию счетах пользователей. Описание её столбцов представлено в таблице 2.5.

Таблица 2.5 – Описание таблицы ACCOUNTS

Название столбца	Тип данных	Описание
id	int	Идентификатор аккаунта
owner_id	int	Идентификатор владельца аккаунта
account_type	int	Идентификатор типа аккаунта
balance	money	Баланс аккаунта
currency	int	Идентификатор валюты аккаунта
is_locked	BIT	Заблокирован ли аккаунт (1 - да, 0 - нет)

Таблица CREDIT_TYPES хранит информацию типах кредитов доступных пользователю. Описание её столбцов представлено в таблице 2.6.

Таблица 2.6 – Описание таблицы CREDIT_TYPES

Название столбца	Тип данных	Описание
id	int	Идентификатор типа кредита

credit_type_name	NVARCHAR	Название типа кредита
------------------	----------	-----------------------

Таблица CREDIT_CONDITIONS хранит информацию об условиях кредитов доступных пользователей. Описание её столбцов представлено в таблице 2.7.

Таблица 2.7 – Описание таблицы CREDIT_CONDITIONS

Название столбца	Тип данных	Описание
id	int	Идентификатор условия кредита
credit_name	NVARCHAR	Название кредита
credit_type	int	Идентификатор типа кредита
percentage_per_year	FLOAT	Процентная ставка в год
max_sum	money	Максимальная сумма кредита
currency	int	Идентификатор валюты

Таблица DEPOSIT_TYPES хранит информацию о типах доступных депозитов. Описание её столбцов представлено в таблице 2.8

Таблица 2.8 – Описание таблицы DEPOSIT_TYPES

Название столбца	Тип данных	Описание
id	int	Идентификатор типа депозита
deposit_type_name	NVARCHAR	Название типа депозита

Таблица DEPOSIT_CONDITIONS хранит информацию о доступных для пользователя депозитах. Описание её столбцов представлено в таблице 2.9.

Таблица 2.9 – Описание таблицы DEPOSIT_CONDITIONS

Название столбца	Тип данных	Описание
id	int	Идентификатор условия депозита
deposit_condition_name	NVARCHAR	Название условия депозита
deposit_type	int	Идентификатор типа депозита
percentage_per_year	FLOAT	Процентная ставка в год
currency	int	Идентификатор валюты

Описание столбцов таблицы ACTION_TYPES представлено в таблице 2.10.

Таблица 2.10 – Описание таблицы ACTION_TYPES

Название столбца	Тип данных	Описание
id	int	Идентификатор типа действия
action_name	NVARCHAR	Название действия

Данная таблица хранит информацию о типах операций совершаемых. В базе данных.

Таблица ACTIONS хранит информацию обо всех операциях в базе данных. Таблице 2.11. Описание столбцов таблицы ACTIONS приведены в таблице 2.11.

Таблица 2.11 – Описание столбцов таблицы ACTIONS

Название столбца	Тип данных	Описание
id	INT	Идентификатор записи в журнале операций
user_id	INT	Идентификатор пользователя
account_id	INT	Идентификатор аккаунта
table_name	NVARCHAR	Название таблицы, измененной операцией
action_time	DATETIME	Время выполнения операции
additional_info	NVARCHAR	Дополнительная информация об операции
action_type	INT	Тип действия

Таблица CURRENCY_CONVERSION хранит информацию о курсах валют. Описание столбцов таблицы приведены в таблице 2.12.

Таблица 2.12 – Описание столбцов таблицы CURRENCY_CONVERSION

Название столбца	Тип данных	Описание
pair_id	int	Идентификатор конверсии валют
currncy1	int	Идентификатор первой валюты
currency2	int	Идентификатор второй валюты
ratio	FLOAT	Курс конверсии

2.4 Проектирование сервера web-приложения

Для обработки запросов применяется пять роутеров, каждый из которых обрабатывает запросы к определённым адресам. Так, authRouter обрабатывает все запросы, связанные с авторизацией, такие как авторизация, регистрация, завершение сессии. Все запросы обязательно должны начинаться с /account. Более подробно это рассмотрено в таблице 2.13

Таблица 2.13 – Список представляемых обработчиков роутера authRouter

Адрес	Метод	Описание
/register	GET	Отображение страницы регистрации
/register	POST	Обработка регистрации
/login	GET	Отображение страницы входа
/login	POST	Обработка входа
/logout	GET	Выход из системы и удаление токена аутентификации

Роутер `accountRouter` предназначен для обработки запросов связанных со счетами. Все запросы обязательно должны начинаться с `/account` Список предоставляемых обработчиков представлен в таблице 2.14.

Таблица 2.14 – Список предоставляемых обработчиков роутера `accountRouter`

Адрес	Метод	Описание	Адрес
/account-creation	GET	Отображение страницы создания счета	/account-creation
/account-creation	POST	Создание нового счета	/account-creation
/account-deletion/:id	DELETE	Удаление счета	/account-deletion/:id
/transfer-money	GET	Отображение страницы перевода денег	/transfer-money
/transfer	POST	Перевод денег между счетами	/transfer

Для проверки авторизации используется `middleware`, которое проверяет, находится ли запрашиваемый ресурс в списке защищаемых, и в случае, если для доступа к данному ресурсу необходима роль соискателя, работодателя или администратора, производит авторизацию и аутентификацию.

Все запросы, для которых не был зарегистрирован обработчик, сначала обрабатываются `middleware`, обрабатывающим запросы на статические файлы. Если и это `middleware` не вернуло ответ, возвращается страница `ejs`.

3 Разработка web-приложения

3.1 Разработка бэкэнда

Для разработки бэкэнда был использован фреймворк express. Согласно [2], в нём для обработки запросов могут применяться простые обработчики запросов. Обработчики запросов, добавляющие к ответу заголовки CORS и возвращающие html-страницу на любой запрос, представлены в листинге 3.1.

```
const express = require("express");
const authRoutes = require("./routes/authRoutes");
const userRouter = require("./routes/userRoutes"); // Путь к файлу с роутером
const accountRoutes = require("./routes/accountRoutes");
const creditRoutes = require("./routes/creditRoutes");
const depositRoutes = require("./routes/depositRoutes");
const path = require("path");
const app = express();
const cookieParser = require("cookie-parser");
//MIDDLEWARE
app.use(cookieParser());
app.use(express.json());
// ROUTES
app.use("/auth", authRoutes);
app.use("/users", userRouter);
app.use("/account", accountRoutes);
app.use("/credit", creditRoutes);
app.use("/deposit", depositRoutes);
```

Листинг 3.1 – Простые обработчики запросов

Объекты запроса и ответа над которым можно производить различные операции, такие как получение и установка cookie, чтение тела запроса, установка тела ответа при помощи оператора return и так далее.

Для запуска сервера необходимо создать объект приложения, определить обработчики запросов и роутеры, преобразовать объект приложения к слушателю событий Node.js и запустить сервер при помощи метода listen. Код запуска сервера представлен в листинге 3.3. Поддержка HTTPS реализована согласно [3].

```
const PORT = process.env.PORT || 3000;
app.listen(PORT, () => console.log(`Server running on port ${PORT}`));
```

Листинг 3.3 – Код запуска сервера

Для обработки запросов на WebSocket-соединение используется специальный адаптер для Node.js от Crossws. Функция обработки WebSocker-соединений представлена в листинге 3.4.

```
import wsAdapter from "crossws/adapters/node";
const { handleUpgrade } = wsAdapter({
  hooks: {
    async open(peer) {
      console.log("[ws] open", peer);
      vacancyEmitter.on('changed', vacancy => { peer.send(vacancy);
console.log('sending') });
    },

    message(peer, message) {
      console.log("[ws] message", peer, message);
      if (message.text().includes("ping")) {
        peer.send("pong");
      }
    },
    close(peer, event) {console.log("[ws] close", peer, event);
    },
    error(peer, error) {console.log("[ws] error", peer, error);
    },
  },
});
```

Листинг 3.4 – Функция обработки WebSocker-соединений

Файлы, требующиеся для работы фронтэнда, такие как таблицы стилей и скрипты, запрашиваются браузером и должны обрабатываться отдельно. Для этого предусмотрено использование движка ejs. Код подключения данного шаблонизатора представлен в листинге 3.5.

```
app.set("view engine", "ejs");
app.engine("ejs", require("ejs").__express);
app.set("views", path.join(__dirname, "views"));
```

Листинг 3.5 – Функция обработки запросов на статические файлы

Для получения данных из базы данных использовалась ORM Prisma. Применялся подход database-first, при котором в первую очередь разрабатывались база данных, а уже потом модели и на основе моделей обработчики запросов. Были определены модели для пользователя, счета, ролей пользователей, кредитов, вкладов. Объявление модели представлено в листинге 3.6.

```
Npx prisma init
Npx prisma db pull
Npx prisma generate
```

Листинг 3.6 – Команды инициализации клента для работы с базой данных

```

async function getUserRoleFromToken(token) {
  try {
    // Верификация токена
    const decodedToken = jwt.verify(token, "secret_key");
    const username = decodedToken.username;

    // Поиск пользователя в базе данных
    const user = await prisma.users.findUnique({
      where: {
        username: username,
      },
      include: {
        role: true, // Включаем связанную сущность "role"
      },
    });

    // Проверка наличия пользователя в базе данных
    if (!user) {
      return null; // Пользователь не найден
    }

    // Получение роли пользователя
    const userRole = user.role.id;

    return userRole; // Возвращаем роль пользователя
  } catch (error) {
    console.error("Ошибка при получении роли пользователя из токена:",
error);
    return null; // Возвращаем null в случае ошибки
  }
}

```

Листинг 3.7 – Функция проверки авторизации пользователя

```

async function getUserIdFromToken(token) {
  try {
    // Верификация токена
    const decodedToken = jwt.verify(token, "secret_key");
    return decodedToken.id; // Возвращаем id пользователя из токена
  } catch (error) {
    console.error("Ошибка при получении id пользователя из токена:", error);
    return null; // Возвращаем null в случае ошибки
  }
}

```

Листинг 3.8 функция проверки идентификатора пользователя на основе jwt токена

3.2 Разработка фронтэнда

Для разработки сайта использовалась библиотека react в качестве базы, пакет ejs для создания нескольких страниц с разными URI и Фрагмент страницы фронтэнд-приложения представлен в листинге 3.8.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Login</title>
</head>
<body>
  <h1>Login</h1>
  <form id="loginForm" action="/auth/login" method="POST">
    <input type="text" name="username" id="username" placeholder="Username"
required><br>
    <input type="password" name="password" id="password"
placeholder="Password" required><br>
    <button type="submit">Login</button>
  </form>

  <script>
    document.getElementById('loginForm').addEventListener('submit',
function(event) {
      event.preventDefault(); // Предотвращаем стандартное действие формы

      var username = document.getElementById('username').value;
      var password = document.getElementById('password').value;

      var formData = {
        username: username,
        password: password
      };

      // Отправляем данные на сервер
      fetch('/auth/login', {
        method: 'POST',
        headers: {
          'Content-Type': 'application/json'
        },
        body: JSON.stringify(formData)
      })
      .then(response => response.json())
      .then(data => console.log(data))
```



```

        .catch((error) => console.error('Error:', error));
    });
</script>
</body>
</html>

```

Листинг 3.9 – Фрагмент роутера фронтэнд-приложения

За каждую страницу отвечает свой компонент, который запрашивает данные с сервера и отображает их на странице. Для запроса данных с сервера была разработана специальная функция-декоратор, которая запрашивает данные с определённого адреса и в зависимости от возвращённого статуса ответа либо возвращает данные, либо перенаправляет пользователя на главную страницу или на страницу выхода из учётной записи в случае ошибки. Код данной функции представлен в листинге 3.9.

```

export async function fetchForLoader(path) {
    return fetch(path).then(r => {
        if (r.ok) return r.json();
        else throw r.json();
    })
    .catch(async err => {
        err = await err;
        console.log(err);
        if (err.code === 401) {
            location.href = '/signout';
        } else if (err.code === 403) {
            location.href = '/';
        }
    })
    .then(d => {
        return d;
    });
}

```

Листинг 3.10 – Функция запроса данных с сервера для загрузчика

Для отправки данных на сервер была использована функция `fetch`. Её код представлен в листинге 3.10.

```

document.getElementById('loginForm').addEventListener('submit', function(event) {
    event.preventDefault(); // Предотвращаем стандартное действие формы

```

```

var username = document.getElementById('username').value;
var password = document.getElementById('password').value;

var formData = {
  username: username,
  password: password
};

// Отправляем данные на сервер
fetch('/auth/login', {
  method: 'POST',
  headers: {
    'Content-Type': 'application/json'
  },
  body: JSON.stringify(formData)
})
.then(response => response.json())
.then(data => console.log(data))
.catch((error) => console.error('Error:', error));

```

Листинг 3.11 – Функция для отправки данных на сервер

В качестве параметров данная функция принимает параметры строки запроса текущей страницы, общее количество элементов и функцию обратного вызова, которая вызывается при изменении состояния блока `Pagination`. Так, при выборе любого номера страницы будет произведён переход на данную страницу, но с параметром `offset`, равным произведению разности номера страницы и единицы и двадцати. Сервер, получив запрос с параметром `URI offset`, запросит из базы данных строки с заданным смещением.

4 Тестирование web-приложения

Для тестирования web-приложения использовалось ручное тестирование. Обработчики запросов были проверены на возможность неавторизованного и неаутентифицированного доступа. В ответ на неаутентифицированный доступ присылается json файл с сообщением об ошибке. Пример страницы, показывающей сообщение об ошибке о несуществующей странице, представлен на рисунке 4.1.

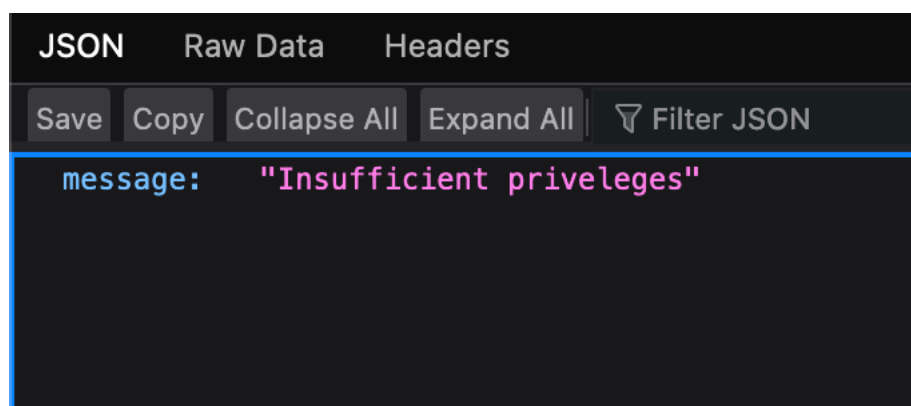
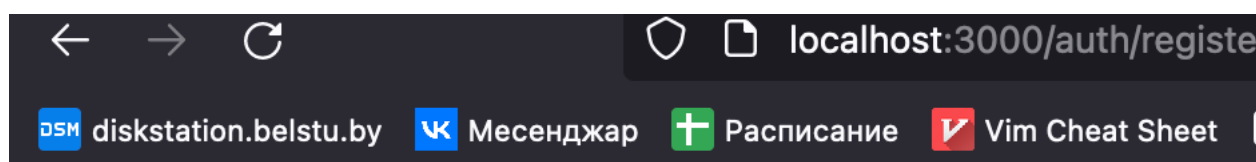


Рисунок 4.1 – Страница сообщения об ошибке не найденной страницы

Также обработчики запросов были проверены на правильность обработки запросов. На рисунке 4.2 представлена страница об ошибке, возникающей при вводе неправильного маршрута.



Cannot GET /auth/registre

Рисунок 4.2 – Страница сообщения об ошибке неверный маршрут

Также обработчики запросов были проверены при помощи ПО Postman. Пример запроса представлен на рисунке 4.3.

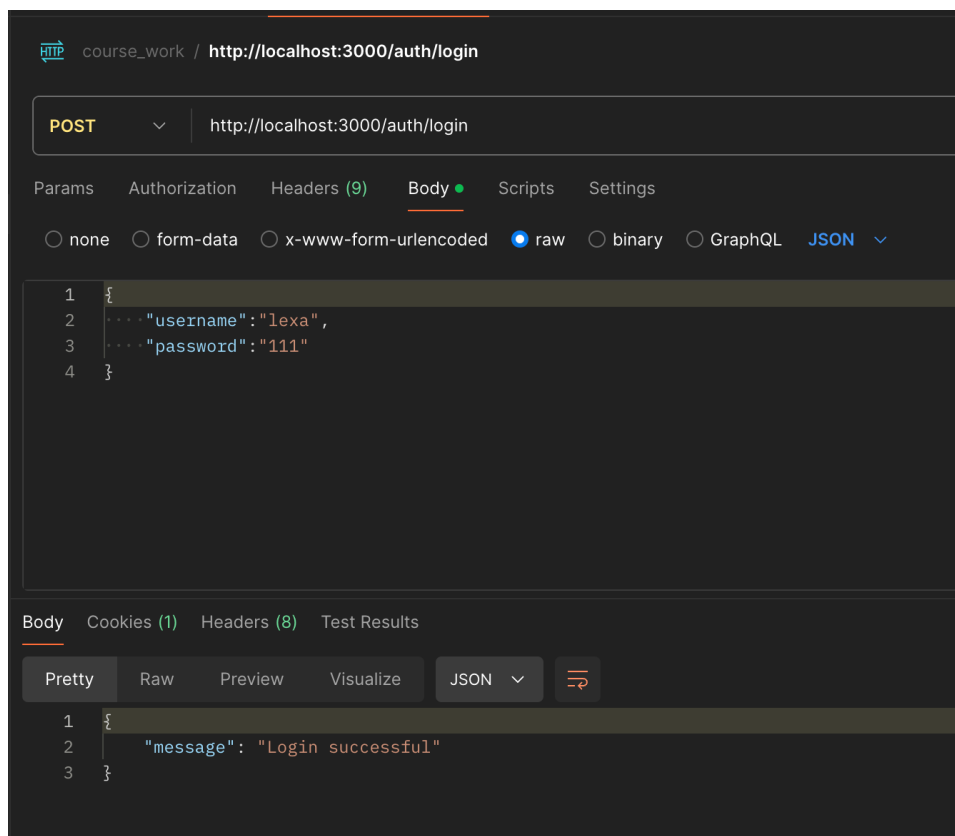


Рисунок 4.3 – Пример результата теста в Postman

В данном тесте был выполнен запрос к обработчику, который предусматривает проверку авторизации. На клиентской стороне были установлены cookie токенов, но не были установлены cookie идентификатора и типа пользователя. Обработчик обнаружил это и вернул сообщение об ошибке.

Схожие сообщения возвращаются в случае, если администратор пытается обратиться к обработчикам запросов, к которым не должен иметь доступа. К таким обработчикам относятся обработчики роутера `creditrouter` и обработчики `depositRouter`. В таких случаях пользователю возвращается ответ с кодом 403 и сообщением о том, что данный обработчик не предназначен для обработки запросов администратора.

Также было применено автоматическое тестирование. Для этого было создано вспомогательное приложение, выполняющее запросы ко всем обработчикам запросов с заданными параметрами при помощи функции `fetch`, вызываемую со всеми методами для каждого адреса. Были проверены все обработчики запросов всеми HTTP-методами с различными параметрами. Фрагмент приложения для тестирования представлен в листинге 4.1.

```
process.env.NODE_TLS_REJECT_UNAUTHORIZED = "0";
```

```
(async () => {
  for (let method of ['GET', 'PUT', 'POST', 'DELETE']) {
    for (let uri of uris) {
      for (let param of params) {
```

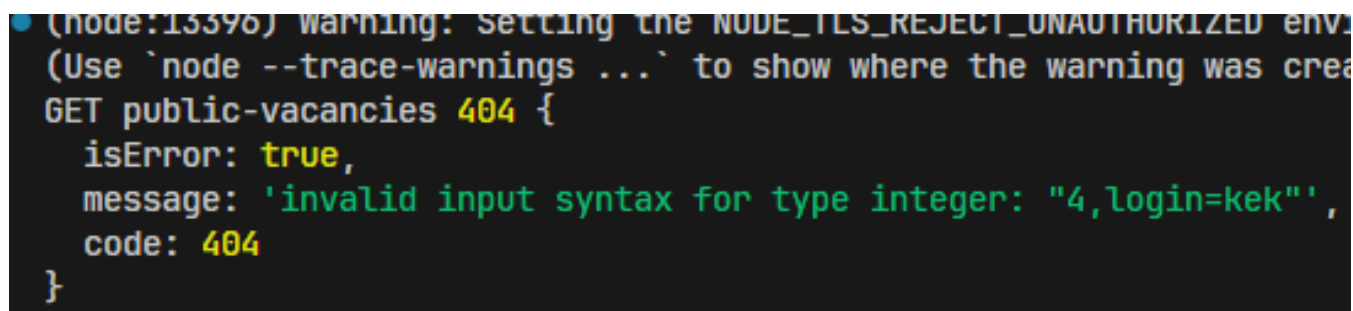
```

const response = await fetch(`https://localhost:4433/${uri}?${params}`, { method
});
const contentType = response.headers.get('content-type');
let responseBody;
if (contentType === 'application/json') {
  responseBody = await response.json();
} else {
  responseBody = await response.text();
}
console.log(method, uri, response.status, responseBody);
}
}
}
})()

```

Листинг 4.1 – Фрагмент приложения для тестирования

Фрагмент вывода данного приложения представлен на рисунке 4.4.



```

(node:13396) warning: Setting the NODE_TLS_REJECT_UNAUTHORIZED env.
(Use `node --trace-warnings ...` to show where the warning was crea
GET public-vacancies 404 {
  isError: true,
  message: 'invalid input syntax for type integer: "4,login=kek"',
  code: 404
}

```

Рисунок 4.4 – Фрагмент вывода приложения для тестирования

В ходе тестирования были выявлены и исправлены несоответствия запрашиваемых клиентом и возвращаемых сервером данных, а также ошибки в исходном коде.

5 Руководство пользователя

При первом открытии сайта пользователь видит страницу вакансий, представленную на рисунке 5.1.

Login

Username:

Password:

Рисунок 5.1 – Страница авторизации

Для проведения доступных операций пользователю необходимо авторизоваться. После авторизации он будет перенаправлен на страницу личного кабинета из которого уже будут доступны остальные действия. Страницы личных кабинетов для ролей администратор, работник, пользователь представлены на рисунках 5.2, 5.3, 5.4 соответственно.

Admin Page

[chat](#)
[create account](#)
[add credit type](#)
[add credit conditions](#)
[add deposit type](#)
[add deposit conditions](#)
[register user](#)
[LOGOUT](#)

Delete Credit Type:

Delete Credit Condition:

Delete Deposit Condition:

Delete Account:

Delete User:

Edit User:

Рисунок 5.2 – личный кабинет администратора.

Далее рассмотрим личный кабинет работника

Welcome to the Worker Dashboard

Here, you can perform various tasks available for workers.

[LOGOUT](#)

Add Deposit Type

Deposit Type Name:

Add Deposit Conditions

[add deposit conditions](#)

Delete Deposit Condition

Deposit Condition ID:

Create Account

> [create account](#)

Edit User

User ID:

Рисунок 5.3 – личный кабинет работника

Наименее загруженным опциями является личный кабинет обыкновенного пользователя или же клиента. Он представлен на рисунке 5.4

User Page

[Transfer Money](#)

[Transfer Money](#)

[Take Deposit](#)

[LOGOUT](#)

Рисунок 5.4 – Личный кабинет пользователя

Теперь рассмотрим каждый из кабинетов более подробно. В личном кабинете администратора имеются функции отправки уведомления, чата с пользователем, создания счетов, создания типов вкладов и кредитов, создания условий вкладов и кредитов, регистрация новых пользователей. Рассмотрим каждую из этих функций в отдельности.

Функция отправки уведомлений реализована с помощью соответствующей кнопки в личном кабинете. Результат работы отражается в кабинете пользователя и представлен на рисунке 5.5

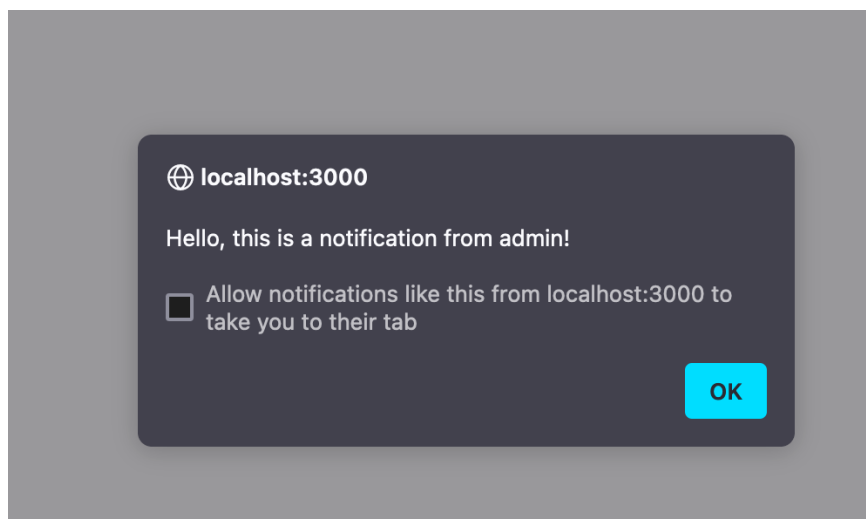


Рисунок 5.5 – уведомление в личном кабинете пользователя

Следующая функция – это чат с пользователем. Вызывается с помощью перехода по соответствующей ссылке в личном кабинете. В чате реализована история сообщений для каждого из участников. Пример страницы чата приведен на рисунке 5.6

Chat as admin

Messages

Messages to/from User

- hello from user

Messages to/from Admin

- hello from admin

Рисунок 5.6 – пример чата с администратором

Администратору доступна также Функция создания счета. Вызывается с помощью перехода по соответствующей ссылке в кабинете. Там необходимо ввести название счета, валюту, id клиента которому он принадлежит и тип счета. Пример страницы приведен ниже.

Account name Personal Account USD Owner id

Рисунок 5.7 – пример страницы создания счета

Администратору также доступны функции создания типа кредита. Страница выглядит аналогично странице создания типа вклада и представлена на рисунке 5.8

Add Credit Type

Credit Type Name:

Рисунок 5.8 – страница создания типа кредита
 Также администратору доступна функция создания условий кредита. Страница выполнения данных действий представлена на ри

Create Credit Conditions

Credit Name:

Credit Type:

Percentage per year:

Max Sum:

Currency:

Paydate:

сунке 5.9

Рисунок 5.9 – создание условий кредита

Здесь необходимо заполнить название условий, тип кредита, процент годовых, максимальную сумму, валюту кредита и дату валютирования.
 Администратору также доступны функции удаления и изменения условий и типов кредитов, информации о пользователях. Данные поля принимают в себя идентификатор изменяемого объекта. Пример представлен на рисунке ниже.

Delete Credit Type:

Delete Credit Condition:

Delete Deposit Condition:

Delete Account:

Delete User:

Edit User:

Рисунок 5.10 – изменение и удаление объектов.

При изменении пользователя после ввода идентификатора происходит перенаправление на страницу изменения личных данных. Эта страница представлена на рисунке 5.11

Edit User

First Name:

Middle Name:

Last Name:

Phone:

Password:

Рисунок 5.11 – редактирование пользователя

Еще одна функция доступная администратору – добавление условий депозитов. Пример страницы для работы данной функции представлен на рисунке 5.12

Create Deposit Conditions

Deposit Condition Name:

Deposit Type:

Percentage per Year:

Currency:

Рисунок 5.12 – добавление условий депозита

Для роли работник доступно большинство тех же функций, что и для администратора. С тем лишь отличием что работник не может посылать уведомления, участвовать в чатах и изменять любую информацию кроме пользовательских данных. Все доступные работнику функции представлены на рисунке 5.12.

Add Deposit Type

Deposit Type Name:

Add Deposit Conditions

[add deposit conditions](#)

Delete Deposit Condition

Deposit Condition ID:

Create Account

> [create account](#)

Edit User

User ID:

Рисунок 5.12 – доступные для работника функции в его личном кабинете.

В личном кабинете у роли пользователь есть следующие функции: чат, перевод средств, открыть вклад, взять кредит. Чат был рассмотрен выше. Открытие вклада и

взятие кредита имеют сходные страницы, поэтому рассмотрена будет только функция взятия кредита. Страница для данной функции представлена на рисунке 5.13.

Credit Conditions

Credit Name	Credit Type Name	Max Sum	Currency	Paydate	Action
Personal Loan Condition 1	Personal Loan	100000	1	05-08	<button>Take</button>
Personal Loan Condition 2	Personal Loan	50000	2	05-07	<button>Take</button>
Business Loan Condition 1	Business Loan	500000	1	05-08	<button>Take</button>
Business Loan Condition 2	Business Loan	200000	2	05-07	<button>Take</button>
Mortgage Condition 1	Mortgage	300000	1	05-08	<button>Take</button>
Mortgage Condition 2	Mortgage	150000	2	05-07	<button>Take</button>

Рисунок 5.12 – страница взятия кредита

Также пользователю доступна функция перевода средств с одного счета на другой. Это представлено на рисунке 5.13.

Transfer Money

From Account:

1

▼

Balance:
 Select an account to view balance

To Account:

Amount:

▲

▼

Transfer

Рисунок 5.13 – пример функции перевода денег между счетами

Таким. Образом в данном разделе было дано описано краткое руководство пользователя для всех доступных ролей в приложении и представлены примеры реализации данных функций.

Заключение

При выполнении курсового проекта было создано приложение банка. Сервер был создан при помощи платформы Node.js, языка программирования JavaScript и фреймворка express. Web-сайт был реализован при помощи библиотеки React. База данных была реализована в СУБД Microsoft SQL Server. Были реализованы все функциональные требования, а именно:

- обеспечивать возможность регистрации и авторизации;
- поддерживать роли администратора и пользователя;
- позволять изменять информацию о клиентах;
- напоминать о платежах по кредитам;
- предоставлять возможность открытия вкладов;
- предоставлять возможность поддерживать связь между клиентом и работником банка при помощи чата;
- предоставлять калькулятор кредита по выбранным условиям;

- предоставлять возможность подбора кредитов по критериям;
- позволять устанавливать автоматическое списание средств на сохраненные платежи;

Также были реализованы WebSocket-сервер и поддержка протокола HTTPS. Приложение было протестировано на наличие ошибок.

Список используемых источников

- 1 PostgreSQL Documentation [Электронный ресурс]. – Электронные данные. – Режим доступа: <https://www.postgresql.org/docs/>.
- 2 h3 – The Web Framework for Modern JavaScript Era [Электронный ресурс]. – Электронные данные. – Режим доступа: <https://h3.unjs.io>.
- 3 Enabling HTTPS on express.js [Электронный ресурс]. – Электронные данные. – Режим доступа: <https://stackoverflow.com/questions/11744975/enabling-https-on-express-js>.
- 4 Material UI components [Электронный ресурс]. – Электронные данные. – Режим доступа: <https://mui.com/material-ui/all-components/>.

Приложение А

Листинг определения моделей для ORM Prisma

```

generator client {
  provider = "prisma-client-js"
}

datasource db {
  provider = "sqlserver"
  url      = env("DATABASE_URL")
}

model account_types {
  id      Int      @id(map: "PK__account__3213E83F2CF43568")
  @default(autoincrement())
  type_name String? @db.NVarChar(200)
  owner_type Int?
  role     role?    @relation(fields: [owner_type], references: [id], onDelete: Cascade,
map: "FK__account_t__owner__3C69FB99")
  accounts accounts[]
}

model accounts {
  id              Int      @id(map:
"PK__accounts__3213E83F9A5EF9A3") @default(autoincrement())
  owner_id        Int?
  account_type    Int?
  balance         Float?   @db.Money
  currency        Int?
  is_locked       Boolean?
  account_types   account_types? @relation(fields: [account_type],
references: [id], onDelete: NoAction, onUpdate: NoAction, map:
"FK__accounts__accoun__45F365D3")
  currency_accounts_currencyTocurrency currency?
  @relation("accounts_currencyTocurrency", fields: [currency], references: [id],
onDelete: NoAction, onUpdate: NoAction, map:
"FK__accounts__curren__46E78A0C")
  users           users?    @relation(fields: [owner_id], references: [id],
onDelete: NoAction, onUpdate: NoAction, map:
"FK__accounts__owner__44FF419A")
  operation_log   operation_log[]
}

model action_types {
  id      Int      @id(map: "PK__action_t__3213E83F663DA6FB")

```

```

@default(autoincrement())
action_name String? @db.NVarChar(200)
}

model credit_conditions {
  id Int @id(map:
"PK__credit_c__3213E83F6D246337") @default(autoincrement())
  credit_name String? @unique(map:
"UQ__credit_c__E2D7E624B7F9B46A") @db.NVarChar(200)
  credit_type Int?
  percentage_per_year Float?
  max_sum Float? @db.Money
  currency Int?
  paydate String? @db.NVarChar(7)
  credit_types credit_types? @relation(fields: [credit_type],
references: [id], onDelete: Cascade, map: "FK__credit_co__credi__4BAC3F29")
  currency_credit_conditions_currencyTocurrency currency?
  @relation("credit_conditions_currencyTocurrency", fields: [currency], references: [id],
onDelete: Cascade, map: "FK__credit_co__curre__4CA06362")
}

model credit_types {
  id Int @id(map: "PK__credit_t__3213E83F9770D8BB")
  @default(autoincrement())
  credit_type_name String? @db.NVarChar(200)
  credit_conditions credit_conditions[]
}

model currency {
  id Int @id(map:
"PK__currency__3213E83FF8B515D0") @default(autoincrement())
  currecy_name String? @db.NVarChar(200)
  currency_short_name String? @db.NVarChar(3)
  accounts_accounts_currencyTocurrency accounts[]
  @relation("accounts_currencyTocurrency")
  credit_conditions_credit_conditions_currencyTocurrency credit_conditions[]
  @relation("credit_conditions_currencyTocurrency")
  currency_conversion_currency_conversion_currency2Tocurrency
currency_conversion[] @relation("currency_conversion_currency2Tocurrency")
  currency_conversion_currency_conversion_currncy1Tocurrency
currency_conversion[] @relation("currency_conversion_currncy1Tocurrency")
  deposit_conditioins_deposit_conditioins_currencyTocurrency deposit_conditioins[]
  @relation("deposit_conditioins_currencyTocurrency")
}

```

```

model currency_conversion {
  pair_id          Int      @id(map:
"PK__currency__97BA35D9017E1F48") @default(autoincrement()))
  currncy1         Int?
  currency2        Int?
  ratio            Float?
  currency_currency_conversion_currency2Tocurrency currency?
  @relation("currency_conversion_currency2Tocurrency", fields: [currency2],
references: [id], onDelete: NoAction, onUpdate: NoAction, map:
"FK__currency__curre__4222D4EF")
  currency_currency_conversion_currncy1Tocurrency currency?
  @relation("currency_conversion_currncy1Tocurrency", fields: [currncy1], references:
[id], onDelete: NoAction, onUpdate: NoAction, map:
"FK__currency__currn__412EB0B6")
}

model deposit_conditioins {
  id              Int      @id(map:
"PK__deposit__3213E83F13DA45E9") @default(autoincrement()))
  deposit_condition_name String? @unique(map:
"UQ__deposit__E5474B9E340FE6D4") @db.NVarChar(200)
  deposit_type    Int?
  percentage_per_year Float?
  currency        Int?
  currency_deposit_conditioins_currencyTocurrency currency?
  @relation("deposit_conditioins_currencyTocurrency", fields: [currency], references:
[id], onDelete: NoAction, onUpdate: NoAction, map:
"FK__deposit_c__curre__52593CB8")
  deposit_types   deposit_types? @relation(fields: [deposit_type],
references: [id], onDelete: Cascade, map: "FK__deposit_c__depos__5165187F")
}

model deposit_types {
  id          Int      @id(map: "PK__deposit__3213E83F9B637F0F")
  @default(autoincrement())
  deposit_type_name String? @db.NVarChar(200)
  deposit_conditioins deposit_conditioins[]
}

model operation_log {
  id          Int      @id(map: "PK__operatio__3213E83F6BDFC5E6")
  @default(autoincrement())
  user_id     Int?
  account_id  Int?
  table_name  String? @db.NVarChar(200)
}

```



```

    action_time DateTime? @default(now(), map:
"DF__operation__actio__59063A47") @db.DateTime
    additional_info String? @db.NVarChar(Max)
    accounts accounts? @relation(fields: [account_id], references: [id], onDelete:
NoAction, map: "FK__operation__accou__5812160E")
    users users? @relation(fields: [user_id], references: [id], onDelete: NoAction,
map: "FK__operation__user__571DF1D5")
}

model role {
    id Int @id(map: "PK__role__3213E83FC3DB53E1")
@default(autoincrement())
    role_name String? @db.NVarChar(200)
    role_privs_level Int?
    account_types account_types[]
    users users[]
}

model users {
    id Int @id(map: "PK__users__3213E83FD0197FA7")
@default(autoincrement())
    first_name String? @db.NVarChar(200)
    midle_name String? @db.NVarChar(200)
    last_name String? @db.NVarChar(200)
    phone String? @db.NChar(15)
    user_role Int?
    username String? @unique(map: "UQ__users__F3DBC57208A65445")
@db.VarChar(200)
    passwd String? @db.VarChar(255)
    accounts accounts[]
    operation_log operation_log[]
    role role? @relation(fields: [user_role], references: [id], onDelete:
Cascade, map: "FK__users__user_role__398D8EEE")
}

```