Министерство образования и науки РФ

Московский Авиационный институт
(Национальный исследовательский университет)

Институт №8
«Информационные технологии и прикладная математика»

Кафедра 813
«Компьютерная математика»

Курсовой проект по дисциплине «Базы данных»

Тема: «Веб-приложение для тестирования»

Студент: Васильев Дмитрий Олегович

Группа: М8О-310Б-18

Преподаватель: Романенков Александр Михайлович

Дата: 9 ноября 2020 г.

Оценка: _____

Подпись преподавателя: _____

Подпись студента: _____

Москва 2020

Содержание

1 Введение

1.1 Формальные требования

1. Необходимо выбрать предметную область для создания базы данных. Выбранная предметная область должна быть уникальной для всего потока, а не только в рамках учебной группы.

2. Необходимо описать таблицы и их назначение. Выполнить проектирование логической структуры базы данных. Описать схему базы данных. Все реальные таблицы должны иметь 3 нормальную форму или выше. База данных должна иметь минимум 5 таблиц.

3. Необходимо разработать два клиентских приложения для доступа к базе данных. Данные приложения должны быть написаны на двух разных языках программирования и иметь разный интерфейс (например, классическое оконное приложение и web-приложение). Выбор языков программирования произволен.

4. Необходимо организовать различные роли пользователей и права доступа к данным. Далее, необходимо реализовать возможность создания архивных копий и восстановления данных из клиентского приложения.

5. При разработке базы данных следует организовать логику обработки данных не на стороне клиента, а, например, на стороне сервера, базы данных, клиентские приложения служат только для представления данных и тривиальной обработки данных.

6. Ваша база данных должна иметь представления, триггеры и хранимые процедуры, причем все эти объекты должны быть осмысленны, а их использование оправдано.

7. При показе вашего проекта необходимо уметь демонстрировать таблицы, представления, триггеры и хранимые процедуры базы данных, внешние ключи, ограничения целостности и др. В клиентских приложениях уметь демонстрировать подключение к базе данных, основные режимы работы с данными (просмотр, редактирование, обновление ...)

8. Необходимо реализовать корректную обработку различного рода ошибок, которые могут возникать при работе с базой данных.

1.2 Клиентские приложения

Один клиент будет SPA веб-приложение. Другой десктоп на Electron. Оба общаются с сервером посредством REST API и HTTP протокола.

1.2.1 Тестирующая система

Данное приложение даёт возможность пользователю:

- создавать, редактировать, удалять тесты.
- рассылать приглашения на прохождения тестов.
- смотреть статистику.

1.2.2 Контольная тестирующая система

Данное приложение нужно для прохождения тестов в очном режиме. В нём можно только проходить тесты.

1.3 Предметная область

Область применения данного приложения универсальна. Можно использовать тестирование на сотрудниках, школьниках, студентах и так далее.

1.4 Стэк технологий

- *JavaScript* [8] — мультипарадигменный язык программирования. Поддерживает объектно-ориентированный, императивный и функциональный стили. Является реализацией стандарта *ECMAScript* [2].
- *React* [13] — *JavaScript* [8] библиотека для создания пользовательских интерфейсов.
- *Redux* [15] — контейнер состояния для *JavaScript* [8] приложения.
- *React Router* [14] — набор навигационных компонентов.
- *SCSS* [16] — препроцессор, который расширяет *CSS* [1].
- *CSS* [1] — формальный язык описания внешнего вида документа, написанного с использованием языка разметки.

- *HTML* [7] — гипертекстовый язык разметки.
- *Node.js* [11] — среда выполнения JavaScript, созданная на основе движка Chrome V8 JavaScript.
- *Express* [4] — минимальный и гибкий *Node.js* [11] фреймворк для создания веб-приложений.
- *PostgreSQL* [12] — объектно-реляционная база данных с открытым исходным кодом.
- *Sequelize* [17] — *Node.js* [11] ORM на основе обещаний для Postgres *PostgreSQL* [12].

1.5 Инструменты

- *Git* [5] — система контроля версий.
- Postman — платформа совместной разработки API
- IDEs: — интегрированная среда разработки.
  - WebStorm
  - DataGrip
- Линтеры — программы, которые следят за качеством кода.
  - *ESLint* [3] — проверяет качество кода на *JavaScript* [8].
  - *Stylelint* [18] — проверяет качество кода на *SCSS* [16], *CSS* [1].
- Тестирующие фреймворки:
  - *Jest* [9] — среда тестирования *JavaScript* [8] с упором на простоту.
- *Webpack* [19] — сборщик статических модулей для современных *JavaScript* [8] приложений.

2 Инфраструктура проекта

2.1 Архитекрутра

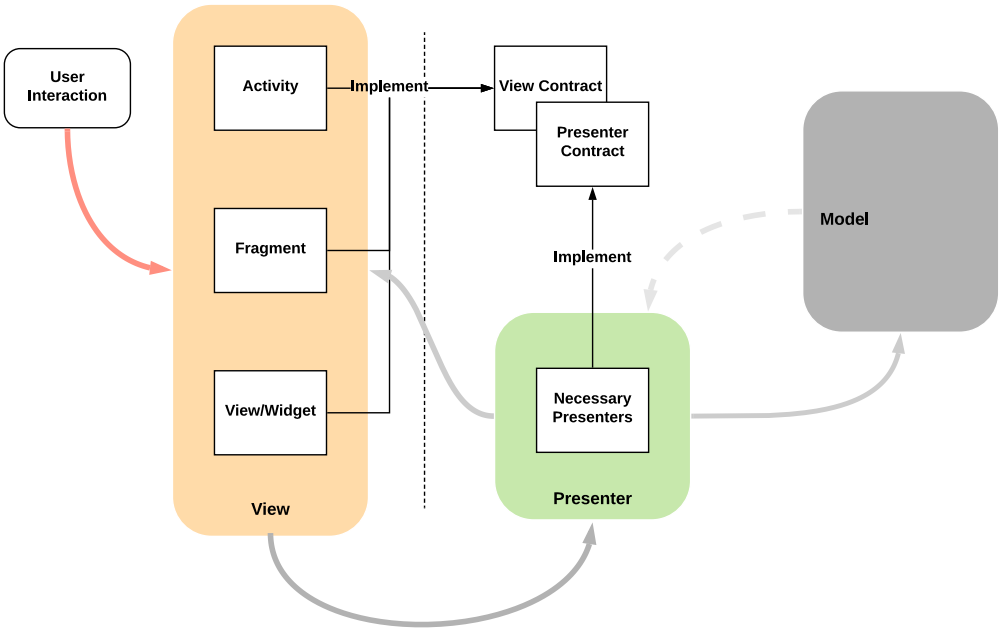

Рис. 1: Визуализация архитектуры MVP

За основу берётся архитектурный паттерн MVP. Он предполагает разделение данных приложения, пользовательского интерфейса и управляющей логики на три отдельных компонента: Модель, Представление и Презентер – таким образом, что модификация каждого компонента может осуществляться независимо.

- **Модель** — предоставляет собой объектную модель некой предметной области, включает в себя данные и методы работы с этими данными, реагирует на запросы из презентера, возвращая данные и/или изменяя своё состояние. При этом модель не содержит в себе информации о способах визуализации данных или форматах их представления, а также не взаимодействует с пользователем напрямую.

- **Представление** — отвечает за отображение информации. Одни и те же данные могут представляться различными способами и в различных форматах. Например, коллекцию объектов при помощи разных представлений можно представить на уровне пользовательского интерфейса как в табличном виде, так и списком.

- **Презентер** — обеспечивает связь между пользователем и системой, использует модель и представление для реализации необходимой реакции на действия

пользователя. Как правило, на уровне презентера осуществляется фильтрация полученных данных и авторизация — проверяются права пользователя на выполнение действий или получение информации.

2.1.1 Связь логических компонетов и применяемых технологих

- **Model** — *Sequelize* [17] для сервера и *Redux* [15] для клиента.
- **View** — *React* [13]
- **Presenter** — *Express* [4]

2.2 Сущности

Всегда перед проектированием проекта описывают сущности и их атрибуты. На основе данной информации будет строиться база данных. В моём случае они следующие:

Сущность	**Атрибуты**
Пользователь	• ID пользователя • Логин • Пароль • Email • Дата создания • Дата последнего изменения
Тест	• ID теста • Название • Описание • Теги • Содержание • Дата создания • Дата последнего изменения

Тег	• ID тега • Название
Попытка	• ID попытки • ID пользователя • ID теста • Результат • Ответы пользователя • Дата прохождения

Таблица 1: Описание сущностей и атрибутов

2.3 Сборка и запуск

Все процессы отвечающие за сборку и запуск приложения я разделил на подзадачи. Каждая такая подзадача является npm скриптом. Они все описываются в файле `package.json`. Также среди данных скриптов можно выделить две группы – Development и Production.

2.3.1 Development

Скрипты из данной группы отвечают за то, чтобы приложение можно было запускать в режиме разработки, а именно:

1. сборка клинтской части не занимало слишком много времени
2. клинет пересобирался при изменении какого-либо файла
3. сервер перезапускался при изменения кода серверверной части
4. в браузере были доступны source map

2.3.2 Production

Скрипты из данной группы отвечают за то, чтобы приложение можно было запускать в режиме с максимальными оптимизациями, а именно:

1. минификация статический файлов
2. оптимизация работы библиотек
3. сборка серверной части

2.4 Деплоинг

Приложение разворачивается в системе *Heroku* [6]. Там же работает СУБД.

2.5 Организация работы с *Git* [5]

2.5.1 Git Workflow

Для организации работы с системой контроля версий в проекте используется подход Git Workflow. Он нужен для согласованного и продуктивного выполнения работы.

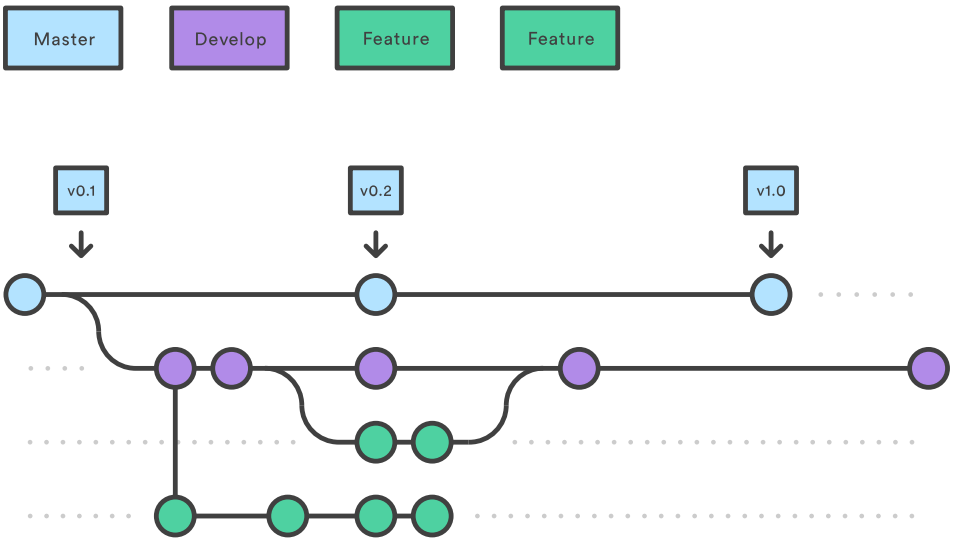

Рис. 2: Пример использования подхода Git Workflow

2.5.2 Git hooks

Чтобы в репозитории хранился код, который проходит проверки линтеров и тестовых фреймворков, нужно использовать Git Hooks. Они позволяют обработать события pre-commit, pre-push, post-commit и так далее.

Есть удобный пакет в npm – husky. Он позволяет определить в `package.json` обработку событий. В моём проекте нужно, чтобы на событие pre-commit выполняли проверки линтеры, а потом при успешном результате исполнялись unit-тесты.

```
{
    "hooks": {
        "pre-commit": "yarn es-lint && yarn style-lint && yarn test",
        "pre-push": "./venv/bin/pytest tests"
    }
}
```

Листинг 1: Настройки для Git Hooks

3 Описание проекта

3.1 Разработка дизайна

Так как я не дизайнер, то мне нужно оперировать концептами и эскизами интерфейса. Чтобы не тратить время верстку стандартный компонентов, я буду использовать Bootstrap. Там оформлены основные компоненты. Также есть встроенная сетка, которая позволяет делать адаптивный дизайн.

3.2 Архитектура решения

В данном решении есть два клиента (1-ый веб-приложение, а 2-ой десктоп), один веб-сервер (он обслуживает оба клиента) и СУБД *PostgreSQL* [12].

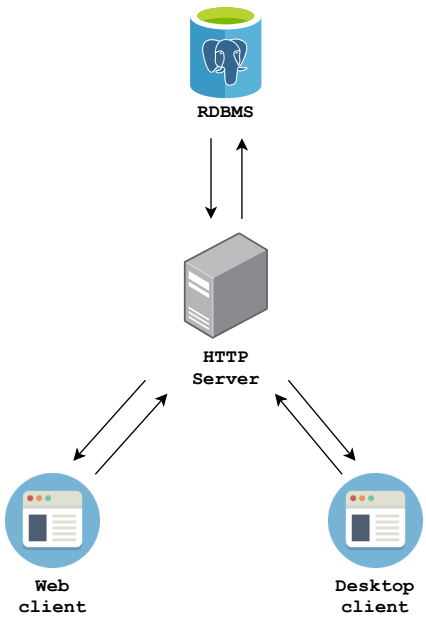

Рис. 3: Схема архитектуры решения

Так как десктоп приложение работает на Electron, то необходимо на сервере обработать отправку нужного шаблона. Для этого есть проверка заголовка `User-agent` в главном роуторе:

```
import { Router } from "express";

const router = new Router();

router.get("*", (req, res) => {
    const isElectronApp = req.header('user-agent').includes('Electron');

    res.render(isElectronApp ? "contest" : "index");
});

export default router;
```

Листинг 2: Исходный код главного роутора

3.3 Авторизация через JSON Web Token (JWT)

3.3.1 Авторизация

Авторизация — это процесс предоставления определённому лицу или группе лиц прав на выполнение определённых действий. Также сюда входит проверка данных, прав при попытке выполнения этих действий.

3.3.2 Аутентификация

Аутентификация — процедура проверки подлинности данный.

3.3.3 JSON Web Token (JWT)

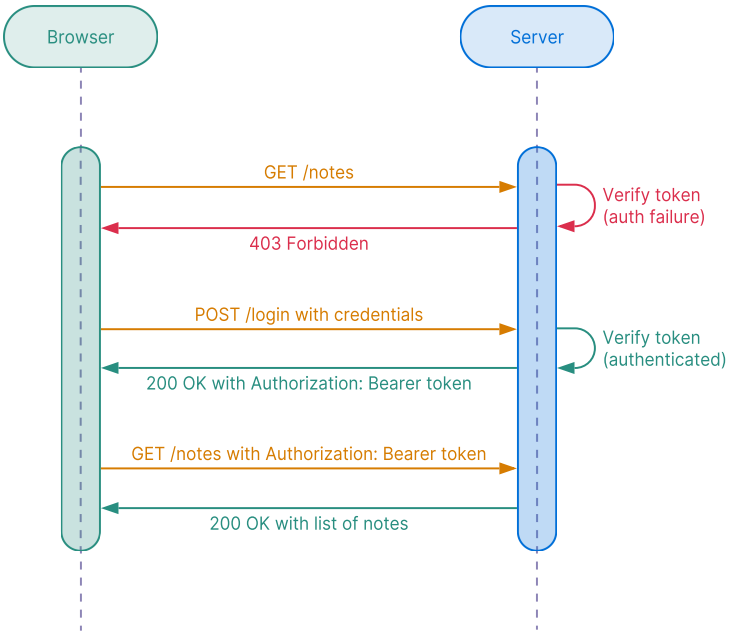

Рис. 4: Демонстрация работы JWT

JSON Web Token (JWT) — это открытый стандарт (RFC 7519), который определяет способ для безопасной передачи информации между сторонами с помощью JSON объектов. Эту информацию можно проверить, потому что она имеет цифровую подпись.

Вот несколько сценариев, в которых полезен JWT:

- **Авторизация** — это наиболее распространенный сценарий использования JWT. После того, как пользователь вошел в систему, каждый последующий запрос будет включать JWT, позволяя пользователю получать доступ к маршрутам, службам и ресурсам, разрешенным с помощью этого токена.
- **Обмен информацией** — JWT хороший способ безопасной передачи информации между сторонами. Поскольку JWT могут быть подписаны, например, с использованием пар открытого и закрытого ключей, вы можете быть уверены, что отправители являются теми, кем они себя называют. Кроме того, поскольку подпись рассчитывается с использованием **заголовка** и **полезных данных**, вы также можете убедиться, что содержимое не было изменено.

JWT состоит из следующих частей:

- **Заголовок** — содержит информацию о том, как должна вычисляться подпись. Обычно состоит из двух частей: типа токена, которым является JWT, и используемого алгоритма подписи, такого как HMAC SHA256 или RSA.
- **Полезные данные** — это данные, которые хранятся внутри JWT. Они также называют JWT-claims (заявки). Список доступных полей для JWT доступен на Wiki.
- **Подпись** — используется для проверки того, что сообщение не было изменено в процессе. В компактной форме JWT является сторой, которая состоит из трех частей, разделенных точками. Псевдокод вычисления подписи:

```
SECRET_KEY = 'some string';
unsignedToken = encodeBase64Url(header) + '.' +
 ↪  encodeBase64Url(payload)
signature = SHA256(unsignedToken, SECRET_KEY);

// собираем всё вместе
jwt = encodeBase64Url(header) + '.' + encodeBase64Url(payload) + '.'
 ↪  + encodeBase64Url(signature);
```

3.3.4 Реализация

Авторизация проходит следущим образом:

1. Пользователь делает `POST /api/signup` запрос на регистрацию. Если всё нормально, то в базе данных создаётся запись с данными пользователя.

2. Пользователь делает `POST /api/signin` запрос на аутентификацию. Если данные верные, то высылается JWT вместе с состоянием пользователя (логин, почта). Когда ответ с сервера получен, то JWT сохраняется в `localStorage` (долговременное хранилище), а состояние передается в глобальное *Redux* [15] хранилище.

3. После того, как состояние глобального хранилища обновилось, приложение обновляет интерфейс.

Если перезагрузить веб-страницу, то *Redux* [15] хранилище обнуляется. Поэтому нам нужно сделать следующее: при запуске приложения проверять на валидность JWT, который лежит в `localStorage`. Это делается через `POST /api/init` запрос. Если токен валидный, то переавторизовываем пользователя. Иначе перенаправляем на главную страницу.

Далее этот токет будет использоваться для доступа к защищёнными ресурсам. Для проверки доступа к ресурсу была реализована middleware, которая исполняется перед основным обработчиком запроса. Если токен валидный, то запрос идёт дальше с извлечёнными данными из JWT, иначе сервер отвечает со статусом `403 Forbidden`.

```
import * as jwt from "jsonwebtoken";
import {FORBIDDEN} from "http-status-codes";

export default async (req, res, next) => {
    const token = req.headers["authorization"];

    let tokenObj = null;
    try {
        tokenObj = await jwt.verify(token, process.env.JWT_SECRET);
    } catch (err) {
        next({
            status: FORBIDDEN,
            errors: [{
                message: "something went wrong"
            }]
        });
    }

    req.userId = tokenObj.sub;

    next();
};
```

Листинг 3: Исходный код middleware checkToken.js

3.3.5 Ограничение доступа к страницам

Для ограничения доступа к определенным страницам я сделал high-order компоненты: `PrivateRoute`, `NotIsLoggedInRoute`.

- `PrivateRoute` – нужен для ограничения доступа к страницам, где нужна авторизация.

```jsx
import * as React from "react";
import PropTypes from "prop-types";
import { Route, Redirect } from "react-router-dom";

const PrivateRoute = ({ component: Component, isLoggedIn, ...rest }) => {
    return (
        <Route { ...rest } render={() => (
            !isLoggedIn ? (
                <Redirect to="/login" />
            ) : (
                <Component />
            )
        )}
        />
    );
};

PrivateRoute.propTypes = {
    component: PropTypes.elementType.isRequired,
    isLoggedIn: PropTypes.bool.isRequired
};

export default PrivateRoute;
```

Листинг 4: Исходный код PrivateRoute.jsx

- `NotIsLoggedInRoute` – нужен для ограничения доступа к страницам, где не нужна авторизация.

```jsx
import * as React from "react";
import PropType from "prop-types";
import { Route, Redirect } from "react-router-dom";

const NotIsLoggedInRoute = ({ component: Component, isLoggedIn, ...rest
  ↪ }) => {
    return (
        <Route { ...rest } render={() => (
            !isLoggedIn ? (
                <Component />
            ) : (
                <Redirect to="/" />
            )
        )} />
    );
};

NotIsLoggedInRoute.propTypes = {
    isLoggedIn: PropType.bool.isRequired,
    component: PropType.elementType.isRequired,
};

export default NotIsLoggedInRoute;
```

Листинг 5: Исходный код NotIsLoggedInRoute.jsx

Оба компонента являются обёрткой над `Route` из библиотеки `react-router-dom`.

3.4 API сервера

- Пользователь:
  - `POST /api/profile/delete` – удаление пользователя.
  - `POST /api/profile/update-password` – обновление пароля пользователя.
  - `POST /api/signup` – регистрация пользователя.
  - `POST /api/signin` – аутентификация пользователя.
- Тест:
  - `POST /api/test/create` – создание теста.
  - `PUT /api/test/update` – обновление теста.
  - `POST /api/test/update` – получение теста для редактирования.

- `GET /api/test/pass` – получение теста для прохождения.
- `GET /api/test/result` – получение результа попытки.
- `POST /api/test/check` – проверка ответов теста.
- `GET /api/test/profile` – получение собственных тестов.
- `GET /api/test/all` – получение всех тестов.
- `DELETE /api/test/delete` – удаление теста.

- Попытка:
  - `GET /api/attempt/test` – получение статистики теста.
  - `GET /api/attempt/profile` – получение попыток пользователя.

3.4.1 Обработка ошибок

Если во время обработки запроса появилась ошибка, то она идёт в middleware `errorHandler`. Там формируется ответ с кодом ошибки и с сообщениями.

```
1  import { INTERNAL_SERVER_ERROR } from "http-status-codes";
2
3  // eslint-disable-next-line no-unused-vars
4  export default function(err, req, res, next) {
5      let { status, errors } = err;
6
7      if (!status) {
8          status = INTERNAL_SERVER_ERROR;
9      }
10
11     res.status(status).json({ errors });
12 }
```

Листинг 6: Исходный код middleware errorHandler.js

3.5 Схема базы данных

3.5.1 Связи

- (users) $1 : N$ (tests)
- (tests) $M : N$ (tags)
- (attempts) $1 : 1$ (tests)

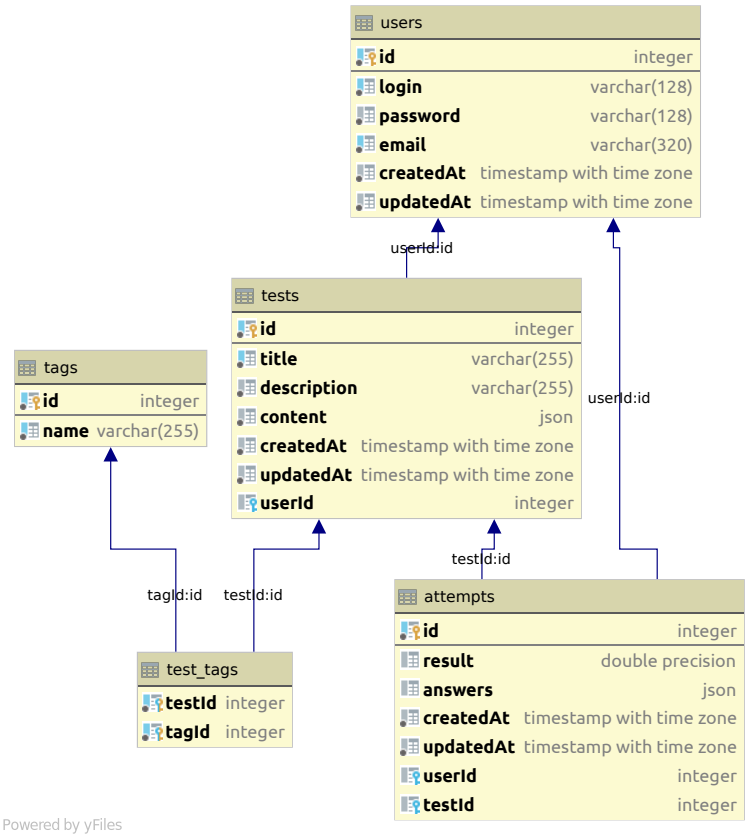

Рис. 5: Схема базы данных

- (attempts) 1 : 1 (users)

Примечание:

- $1 : N$ – один ко многим
- $M : N$ – многие ко многим
- $1 : 1$ – один к одному

3.5.2 Хуки (Триггеры)

Для некоторых таблиц заданы хуки, которые позволяют обработать события связанные с вставкой, обновлением, удалением и т.д.

- `User`:
  - `beforeCreate` – шифрует пароль перед вставкой в таблицу.
  - `beforeUpdate` – шифрует пароль перед обновлением.
- `Attempt`:

  - `afterUpdate` – удаляет все попытки у теста, если он был изменён.

- `TestTag`:

  - `afterDelete` – удаляет теги, которое не используются.

3.5.3 Процедуры (методы модели)

Для модели `User` есть две процедуры:

- `comparePasswords(password: string)` – инстанс метод, который стравнивает пароль текущего пользователя с `password`.
- `hashPassword(password: string)` – статический метод, который шифрует `password`.

3.5.4 Каскадное удаление

Каскадное удаление в мире реляционных баз данных позволяет удалять связанные данные из зависимой таблицы, при удалении данных из основной таблицы. У меня оно использую во всех таблицах.

4 Заключение

Благодаря данному курсовому проекту, я поверхностно освоил разработку SPA приложений с простой JWT авторизации. Также были получены навыки для работы с СУБД *PostgreSQL* [12].

4.1 Недостатки

Подводя итоги, мне бы хотелось перечислить вещи, на которые я буду обращать внимание при разработке следующих проектов:

- Использование CI/CD.
- Использование методологий в вёрстке. Например, Block Element Modificator (BEM). Её разработали внутри компании Яндекс. У них есть свой стек технологий под данную методологию, который облегчает разработку клиентской части.
- Использование NoSQL баз данных вместе с реалиционными. Хранить JSON в таблице плохо, поэтому для этой задачи подходить *MongoDB* [10].
- Разделение клиентского кода на чанки.
- Микросервисная архитектура.
- Использование TypeScript или Flow. Во время разработки я отказался от TypeScript из-за того, что надо очень много времени тратить на type-hinting.

A Визуализации структуры проекта

```
 /db-course-project-app
--- .gitignore
--- package.json
--- yarn-error.log
--- .babelrc
--- jest.config.js
--- requirements.txt
--- .eslintignore
--- yarn.lock
--- .env
--- README.md
--- LICENSE
--- /util
------ nodemon.json
--- /tests
------ test_smoke.py
--- /src
------ main.js
------ setupTests.js
------ .stylelintrc
------ config.js
------ index.js
------ webpack.config.js
------ .eslintrc.js
------ /client
--------- contest.jsx
--------- main.jsx
--------- /containers
------------ /TestEditorQuestionList
--------------- index.js
--------------- TestEditorQuestionList.jsx
------------ /NotIsLoggedInRoute
```

```
--------------- index.js
--------------- NotIsLoggedInRoute.jsx
------------ /TestEditorTagList
--------------- TestEditorTagList.jsx
--------------- index.js
------------ /Test
--------------- index.js
--------------- Test.jsx
------------ /PrivateRoute
--------------- PrivateRoute.jsx
--------------- index.js
--------- /reducers
------------ auth.js
------------ testPassing.js
------------ index.js
------------ testEditor.js
--------- /pages
------------ /SignUp
--------------- style.scss
--------------- index.js
--------------- SignUp.jsx
------------ /Test
--------------- style.scss
--------------- index.js
--------------- Test.jsx
------------ /TestResult
--------------- style.scss
--------------- index.js
--------------- TestResult.jsx
------------ /Login
--------------- style.scss
--------------- Login.jsx
--------------- index.js
--------- /tests
```

```
------------ smoke.test.js
------------ /services
--------------- auth.test.js
--------- /store
------------ store.js
------------ index.js
--------- /helpers
------------ header.js
------------ question.js
------------ loader.js
------------ token.js
--------- /apps
------------ /main
--------------- App.jsx
--------------- /pages
------------------ /ProfileSettings
---------------------- style.scss
---------------------- ProfileSettings.jsx
---------------------- index.js
---------------------- /components
------------------------ /DeleteProfileForm
-------------------------- DeleteProfileForm.jsx
-------------------------- index.js
------------------------ /UpdatePasswordForm
-------------------------- index.js
-------------------------- UpdatePasswordForm.jsx
------------------ /ProfileTests
---------------------- style.scss
---------------------- ProfileTests.jsx
---------------------- index.js
------------------ /TestEditor
---------------------- style.scss
---------------------- index.js
---------------------- TestEditor.jsx
```

```
------------------ /Profile
---------------------- style.scss
---------------------- Profile.jsx
---------------------- index.js
------------------ /ProfileAttempts
---------------------- style.scss
---------------------- ProfileAttempts.jsx
---------------------- index.js
------------------ /Home
---------------------- style.scss
---------------------- index.js
---------------------- Home.jsx
------------------ /TestStatistic
---------------------- TestStatistic.jsx
---------------------- index.js
--------------- /components
------------------ /Header
---------------------- style.scss
---------------------- Header.jsx
---------------------- index.js
------------------ /AnswerEditList
---------------------- style.scss
---------------------- index.js
---------------------- AnswerEditList.jsx
------------------ /QuestionEditItem
---------------------- QuestionEditItem.jsx
---------------------- index.js
------------------ /AnswerEditItem
---------------------- style.scss
---------------------- AnswerEditItem.jsx
---------------------- index.js
------------------ /TableStatistic
---------------------- index.js
---------------------- TableStatistic.jsx
```

```
------------------ /QuestionEditList
---------------------- style.scss
---------------------- index.js
---------------------- QuestionEditList.jsx
------------ /contest
--------------- App.jsx
--------------- /pages
------------------ /AllTests
---------------------- style.scss
---------------------- AllTests.jsx
---------------------- index.js
------------------ /Home
---------------------- style.scss
---------------------- index.js
---------------------- Home.jsx
--------------- /components
------------------ /Header
---------------------- style.scss
---------------------- Header.jsx
---------------------- index.js
--------- /components
------------ /TagList
--------------- style.scss
--------------- index.js
--------------- TagList.jsx
------------ /Tag
--------------- style.scss
--------------- index.js
--------------- Tag.jsx
------------ /ResultStatus
--------------- style.scss
--------------- ResultStatus.jsx
--------------- index.js
------------ /TestCard
```

```
--------------- style.scss
--------------- index.js
--------------- TestCard.jsx
------------ /Question
--------------- style.scss
--------------- index.js
--------------- Question.jsx
------------ /ListTestCards
--------------- style.scss
--------------- index.js
--------------- ListTestCards.jsx
------------ /LogOutModal
--------------- LogOutModal.jsx
--------------- index.js
------------ /AnswerList
--------------- style.scss
--------------- index.js
--------------- AnswerList.jsx
------------ /Answer
--------------- style.scss
--------------- Answer.jsx
--------------- index.js
------------ /Footer
--------------- style.scss
--------------- Footer.jsx
--------------- index.js
------------ /ErrorFormAlert
--------------- ErrorFormAlert.jsx
--------------- index.js
------------ /HttpErrorInfo
--------------- style.scss
--------------- HttpErrorInfo.jsx
--------------- index.js
--------- /actions
```

```
------------ auth.js
------------ testPassing.js
------------ testEditor.js
--------- /history
------------ index.js
--------- /services
------------ testResult.js
------------ attempt.js
------------ auth.js
------------ testPassing.js
------------ editProfileSettings.js
------------ editTest.js
--------- /hoc
------------ /NotIsLoggedInRoute
--------------- index.js
--------------- NotIsLoggedInRoute.jsx
------------ /PrivateRoute
--------------- PrivateRoute.jsx
--------------- index.js
------ /routes
--------- main.js
--------- attempt.js
--------- auth.js
--------- profileModify.js
--------- testEditor.js
------ /tests
--------- smoke.test.js
------ /controllers
--------- attempt.js
--------- auth.js
--------- profileModify.js
--------- testEditor.js
------ /helpers
--------- FormListErrors.js
```

```
------ /templates
--------- contest.handlebars
--------- index.handlebars
--------- /layouts
------------ main.handlebars
--------- /partials
------------ favicon.handlebars
------------ loader.handlebars
------------ meta.handlebars
------ /middlewares
--------- checkToken.js
--------- errorHandler.js
------ /models
--------- TestTag.js
--------- index.js
--------- /Tag
------------ constraints.js
------------ index.js
------------ Tag.js
--------- /Test
------------ constraints.js
------------ config.js
------------ index.js
------------ Test.js
--------- /User
------------ constraints.js
------------ User.js
------------ index.js
--------- /Attempt
------------ Attempt.js
------------ index.js
--- /db
------ init_db.sql
```

Список использованных источников

[1] *CSS*. URL: https://ru.wikipedia.org/?oldid=107701928.

[2] *ECMAScript*. URL: https://ru.wikipedia.org/?oldid=108101383.

[3] *ESLint*. URL: https://eslint.org.

[4] *Express*. URL: https://expressjs.com.

[5] *Git*. URL: https://git-scm.com.

[6] *Heroku*. URL: https://heroku.com.

[7] *HTML*. URL: https://developer.mozilla.org/en-US/docs/Web/HTML.

[8] *JavaScript*. URL: https://ru.wikipedia.org/?oldid=107293496.

[9] *Jest*. URL: https://jestjs.io.

[10] *MongoDB*. URL: https://www.mongodb.com/.

[11] *Node.js*. URL: https://nodejs.org.

[12] *PostgreSQL*. URL: https://www.postgresql.org.

[13] *React*. URL: https://reactjs.org.

[14] *React Router*. URL: https://reactrouter.com.

[15] *Redux*. URL: https://redux.js.org.

[16] *SCSS*. URL: https://sass-lang.com.

[17] *Sequelize*. URL: https://sequelize.org.

[18] *Stylelint*. URL: https://stylelint.io.

[19] *Webpack*. URL: https://webpack.js.org.