



Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет
имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ ИНФОРМАТИКА И СИСТЕМЫ УПРАВЛЕНИЯ

КАФЕДРА ТЕОРЕТИЧЕСКАЯ ИНФОРМАТИКА И КОМПЬЮТЕРНЫЕ ТЕХНОЛОГИИ

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА К КУРСОВОЙ РАБОТЕ ПО КУРСУ *Базы данных*

НА ТЕМУ:

База данных настольных игр

Студент ИУ9-62Б
(Группа)

В. Е. Пичугин
(Подпись, дата) (И.О.Фамилия)

Руководитель курсовой работы

И. Э. Вишняков
(Подпись, дата) (И.О.Фамилия)

Москва 2020 г.

Содержание

Введение	3
1. Обзор предметной области	4
1.1. Формирование требований к логике приложения	4
1.2. Разработка схемы базы данных	4
2. Проектирование веб-приложения	6
3. Реализация веб-приложения	8
3.1. Реализация базы данных	8
3.2. Реализация API-сервера	14
3.3. Реализация клиентской части приложения	19
3.4. Обзор пользовательского интерфейса	22
4. Тестирование приложения	29
Заключение	32
Список использованных источников	33

Введение

Одним из способов интересного и полезного проведения свободного времени являются настольные игры. В настоящее время их существует большое множество, поэтому для поиска новых хороших игр хочется иметь приложение, в котором можно было бы просматривать информацию о любой настольной игре и узнавать мнение других пользователей о ней.

Таким образом, в рамках выполнения данной курсовой работы требуется разработать веб-приложение для обмена информацией о настольных играх, в котором обычный пользователь должен обладать возможностями просмотра списка настольных игр, просмотра расширенной информации о настольной игре и обзорах, связанных с ней, поиска настольных игр по названию.

Также пользователь должен иметь возможность зарегистрировать аккаунт в приложении, что откроет ему ряд таких дополнительных функций, как создание обзора к игре, добавление игры в список любимых, который можно посмотреть в личном кабинете, повышение/понижение рейтинга у конкретного обзора.

1. Обзор предметной области

1.1. Формирование требований к логике приложения

В приложении выделяются две главных сущности – настольная игра и пользователь, каждой из которых нужно предъявить ряд определенных требований.

Объект настольной игры должен содержать подробное ее описание, включающее название игры, год публикации, описание игрового процесса, минимальное и максимальное количество игроков, время игры, минимальное возрастное ограничение, рейтинг, а также информацию об изображении обложки игры и связанных с ней механиках и категориях.

Объект пользователя должен содержать следующую информацию о нем: имя, карму, дату регистрации, пароль, расширенное описание. Пользователь должен иметь возможность писать рецензии к играм, выставляя игре свой рейтинговый балл от 1 до 10 и описывая свое мнение о ней. Соответственно каждая написанная рецензия должна влиять на общий рейтинг игры. Другие пользователи должны иметь возможность увеличить или уменьшить рейтинг определенной рецензии, что окажет влияние на карму создателя рецензии. Также зарегистрированный пользователь должен иметь возможность формировать и просматривать список своих любимых игр и список своих рецензий в личном кабинете.

1.2. Разработка схемы базы данных

В качестве концептуальной модели данных приложения была выбрана модель “сущность-связь” (англ. “Entity-Relationship model”) в нотации Дж. Мартина [1].

В соответствии со сформированными требованиями к приложению была разработана следующая ER-модель (см. Рисунок 1).

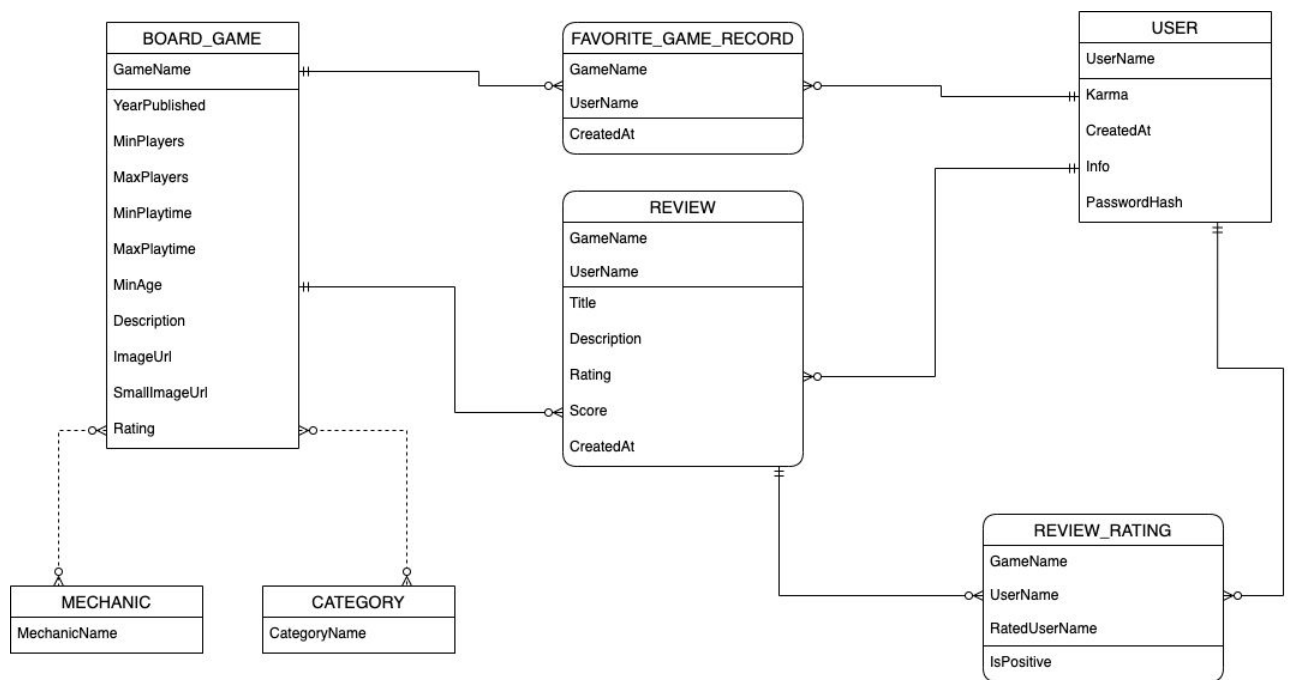


Рисунок 1. ER-модель данных приложения.

2. Проектирование веб-приложения

Приложение основано на классической для веб-приложений клиент-серверной архитектуре, которая подразумевает выделение в проекте двух основных модулей – клиентского и серверного, осуществляющих сетевые запросы к данным и их обработку с последующим ответом соответственно [2].

Ключевым элементом приложения является база данных, хранящая информацию о всех сущностях приложения: настольных играх, пользователях и рецензиях. Для ее наполнения данными о настольных играх был использован ресурс Board Game Atlas API [3], который предоставляет публичный программный интерфейс приложения, позволяющий в формате JSON [4] получить информацию обо всех известных настольных играх.

Одним из важных элементов приложения является механизм аутентификации и авторизации пользователей, основанный на классическом подходе с сессиями и файлами пользовательского контекста (англ. “cookie”) [5]: после того как пользователь авторизуется в системе с помощью имени и пароля, сервер проставит в ответе файл пользовательского контекста с идентификатором авторизованной сессии, а сама сессия с идентификатором пользователя будет храниться в резидентном хранилище. При следующем запросе к серверу у клиентского запроса будет определен пользовательский контекст, на основании лежащего там идентификатора сессии сервер получит идентификатор пользователя из резидентного хранилища, и запрос будет авторизован. Если же пользователь захочет выйти из системы, то необходимо будет лишь удалить сессию из резидентного хранилища на сервере.

Связующим звеном между данными и пользовательским интерфейсом является API-сервер, предоставляющий клиентской части приложения REST API [6] для доступа к данным в формате JSON.

Клиентская часть приложения спроектирована как одностраничное приложение (single page application), использующее единственный HTML-документ как оболочку для всех веб-страниц и организующее взаимодействие с пользователем через динамически подгружаемые данные в формате JSON [7].

Схематично архитектура веб-приложения отображена на Рисунке 2.

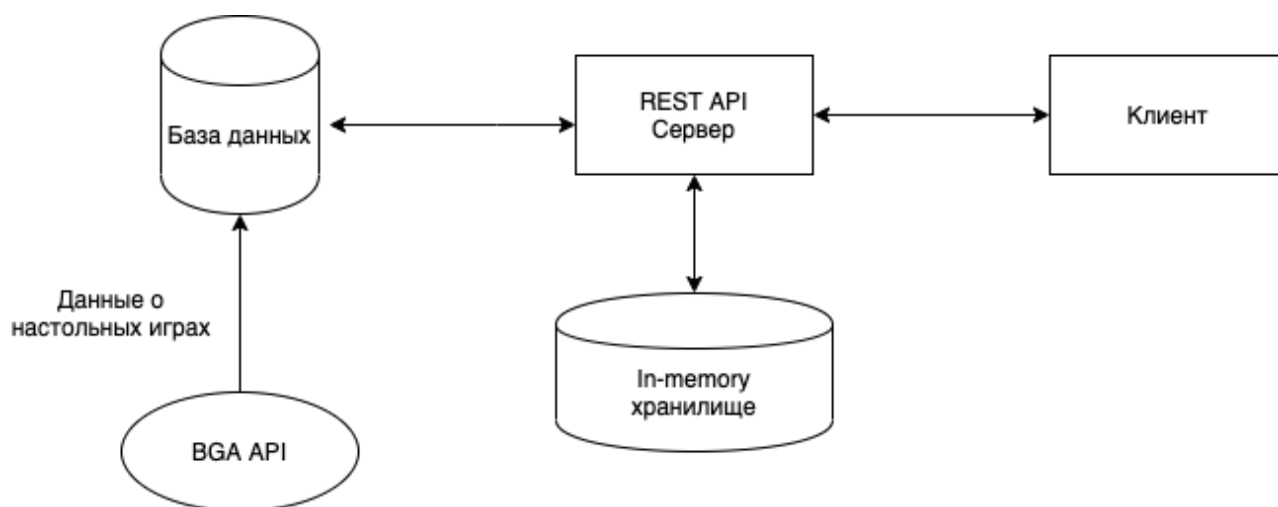


Рисунок 2. Архитектура веб-приложения.

3. Реализация веб-приложения

3.1. Реализация базы данных

В качестве СУБД приложения был выбран PostgreSQL — это объектно-реляционная система управления базами данных с открытым исходным кодом, реализованная в архитектуре клиент-сервер [8].

При реализации приложения использовался PostgreSQL версии 12. Сервер базы данных был запущен локально с использованием приложения Postgres.app [9]. Вся конфигурационная информация о сервере базы данных была помещена в .env файл, и в дальнейшем все части приложения получали конфигурационную информацию именно из этого файла.

Полученная ранее ER-модель данных была преобразована в реляционную модель PostgreSQL (см. Рисунок 3).

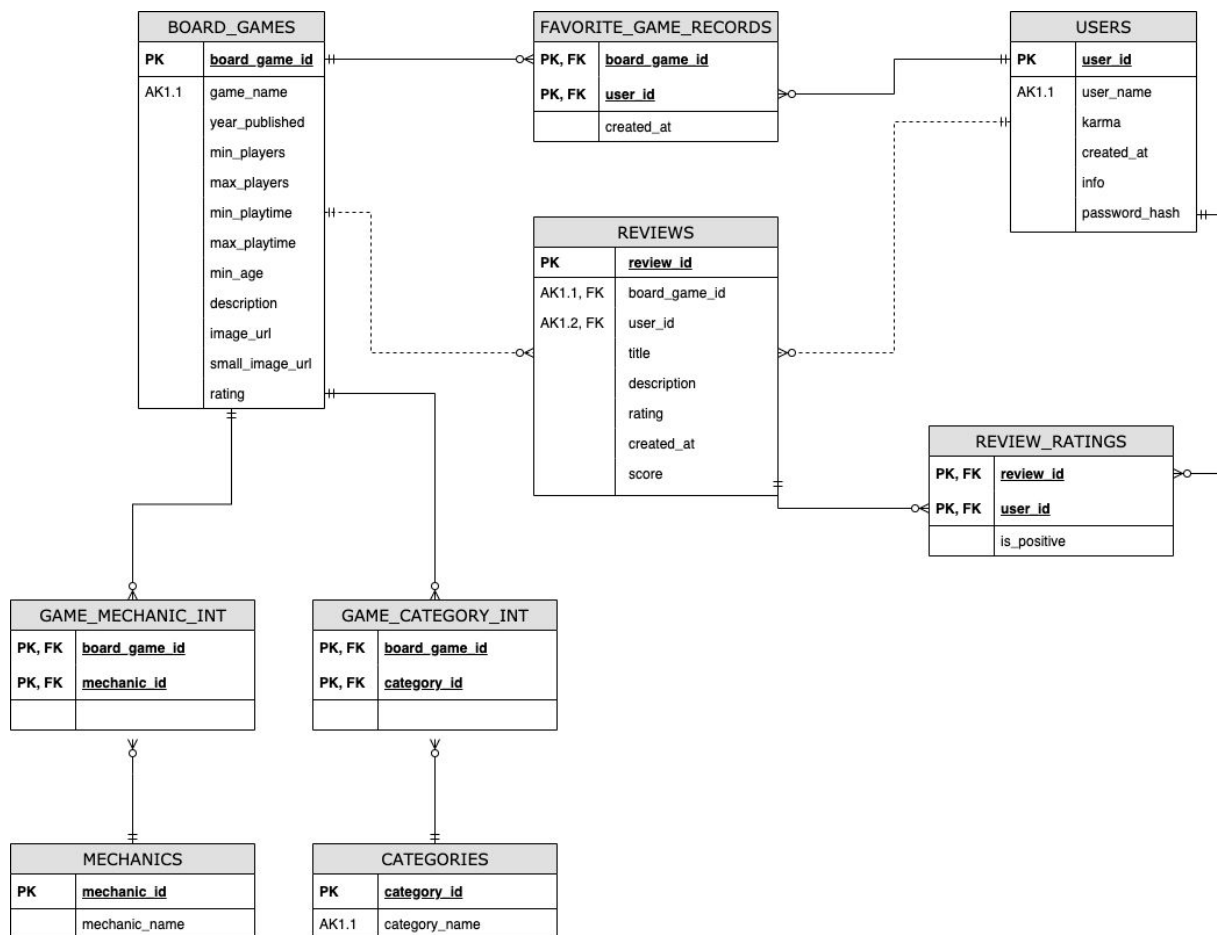


Рисунок 3. Реляционная модель данных приложения.

Свойства столбцов, включающие в себя типы данных, наличие неопределенного значения, значения по умолчанию и ограничения на значения, представлены в Таблице 1.

Таблица 1. Свойства столбцов.

Имя таблицы	Имя столбца	Тип данных	Неопределенное значение	Значение по умолчанию	Ограничения на значения
BOARD_GAMES	board_game_id	varchar	NOT NULL	-	PK
BOARD_GAMES	game_name	varchar	NOT NULL	-	AK, UNIQUE
BOARD_GAMES	year_published	smallint	NULL	-	year_published > 0
BOARD_GAMES	min_players	smallint	NULL	-	min_players >= 0
BOARD_GAMES	max_players	smallint	NULL	-	max_players >= min_players
BOARD_GAMES	min_playtime	smallint	NULL	-	min_playtime > 0
BOARD_GAMES	max_playtime	smallint	NULL	-	max_playtime >= min_playtime
BOARD_GAMES	min_age	smallint	NULL	-	min_age >= 0
BOARD_GAMES	description	text	NOT NULL	-	-
BOARD_GAMES	image_url	varchar	NOT NULL	-	-
BOARD_GAMES	small_image_url	varchar	NOT NULL	-	-

GAMES					
BOARD_GAMES	rating	real	NOT NULL	0	rating >= 0 and rating <= 10
MECHANICS	mechanic_id	varchar	NOT NULL	-	PK
MECHANICS	mechanic_name	varchar	NOT NULL	-	-
CATEGORIES	category_id	varchar	NOT NULL	-	PK
CATEGORIES	category_name	varchar	NOT NULL	-	AK, UNIQUE
GAME_MECHANIC_INT	board_game_id	varchar	NOT NULL	-	PK1.1, FK
GAME_MECHANIC_INT	mechanic_id	varchar	NOT NULL	-	PK1.2, FK
GAME_CATEGORY_INT	board_game_id	varchar	NOT NULL	-	PK1.1, FK
GAME_CATEGORY_INT	category_id	varchar	NOT NULL	-	PK1.2, FK
USERS	user_id	uuid	NOT NULL	-	PK
USERS	user_name	varchar	NOT NULL	-	AK, UNIQUE
USERS	karma	integer	NOT NULL	0	-
USERS	created_at	timestampz	NOT NULL	-	-
USERS	info	text	NULL	-	-
USERS	password_hash	varchar	NOT NULL	-	-
REVIEWS	review_id	uuid	NOT NULL	-	PK

REVIEWS	board_game_id	varchar	NOT NULL	-	AK1.1, FK
REVIEWS	user_id	uuid	NOT NULL	-	AK1.2, FK
REVIEWS	title	varchar	NOT NULL	-	-
REVIEWS	description	text	NOT NULL	-	-
REVIEWS	rating	integer	NOT NULL	-	rating >= 1 and rating <= 10
REVIEWS	created_at	timestampz	NOT NULL	-	-
REVIEWS	score	integer	NOT NULL	0	-
REVIEW_RATINGS	review_id	uuid	NOT NULL	-	PK1.1, FK
REVIEW_RATINGS	user_id	uuid	NOT NULL	-	PK1.2, FK
REVIEW_RATINGS	is_positive	boolean	NOT NULL	-	-
FAVORITE_GAME_RECORDS	board_game_id	varchar	NOT NULL	-	PK1.1, FK
FAVORITE_GAME_RECORDS	user_id	uuid	NOT NULL	-	PK1.2, FK
FAVORITE_GAME_RECORDS	created_at	timestampz	NOT NULL	-	-

Все связи между таблицами построенной реляционной модели имеют тип М-О, т.е родительская сущность является обязательной, дочерняя сущность является необязательной. Поэтому таблица ограничений минимальной кардинальности для всех связей имеет следующий вид (см. Таблица 2).

Таблица 2. Ограничения минимальной кардинальности.

Операция	Действие над родительской таблицей	Действие над дочерней таблицей
Вставка	Без ограничений	Подбор родительской записи
Обновление ключа	Запрет	Запрет
Удаление	Каскадное удаление	Без ограничений

Кроме того был также реализован следующий набор sql, shell и npm скриптов для удобства разработки:

1. `init-db` – уничтожает все таблицы, а затем заново их воссоздает в пустом виде;
2. `clear-games` – удаляет данные из таблиц `board_games`, `mechanics`, `categories`;
3. `populate-db` – наполняет таблицы `board_games`, `mechanics`, `categories` данными из Board Game Atlas API;
4. `refresh-games` – запускает последовательно `clear-games` и `populate-db`;
5. `clear-users` – удаляет данные из таблицы `users`;
6. `select-all` – осуществляет выборку данных из всех таблиц.

Логика самого важного скрипта `populate-db` и в целом всего API-сервера написана на TypeScript – статически типизированном надмножестве языка JavaScript, компилирующемся в последний [10].

Также разработка серверной части велась на платформе Node.js – программной платформе, основанной на компиляторе V8 (транслирующем JavaScript в машинный код), превращающей JavaScript из узкоспециализированного языка в язык общего назначения [11].

В качестве клиента для взаимодействия с сервером PostgreSQL использовалась библиотека `node-postgres`, предоставляющая асинхронный API для взаимодействия с СУБД и преобразующая данные запросов в нативные

типы JavaScript (объекты, массивы и т.д.) [12]. Среди полезных возможностей данной библиотеки также стоит отметить возможность совершать параметризованные запросы без риска sql-инъекций, пул переиспользуемых клиентских соединений, поддержку транзакций из typescript-кода (Postgresql по умолчанию использует уровень изоляции транзакций `READ_COMMITTED`, который защищает от проблем грязного чтения и потерянного обновления, что вполне достаточно для логики данного веб-приложения).

На Рисунке 4 представлена структура вспомогательного скрипта для наполнения базы данных информацией о настольных играх.



Рисунок 4. Структура исходного кода скрипта populate-db.

Составляющие ее элементы характеризуются следующим образом. В файле `tsconfig.json` определены настройки компилятора TypeScript, включающие пути до компилируемых `ts`-файлов, директорию для скомпилированных `js`-файлов, настройки компиляции (были сделаны максимально строгими).

Файл `config.ts` определяет функцию, возвращающую конфигурацию запроса к API. Токен для запроса к Board Game Atlas API берется из `.env` файла с помощью библиотеки `dotenv`.

В `model.ts` определены все необходимые типы данных.

В файле `api.ts` определена асинхронная функция, осуществляющая запросы к API с помощью библиотеки `axios` и преобразующая полученные данные в объекты модели.

Наконец, в файле `populateDB.ts` сосредоточена основная логика заполнения таблиц `board_games`, `mechanics`, `categories`, `game_mechanic_int`, `game_category_int` на основе объектов, полученных из API, с помощью `node-postgres`.

3.2. Реализация API-сервера

Логика API-сервера написана на TypeScript с использованием платформы Node.js и библиотеки Express – минималистичной и гибкой библиотеки для серверных приложений Node.js [13].

Упрощенно архитектуру API-сервера можно описать следующим образом: HTTP-запрос маршрутизируется в нужный сервис, там HTTP-обработчик вызывает нужный метод сервиса, осуществляется взаимодействие с СУБД и отправляется HTTP-ответ с данными в формате JSON, либо HTTP-ответ с возникшей ошибкой. Схематично это изображено на Рисунке 5.

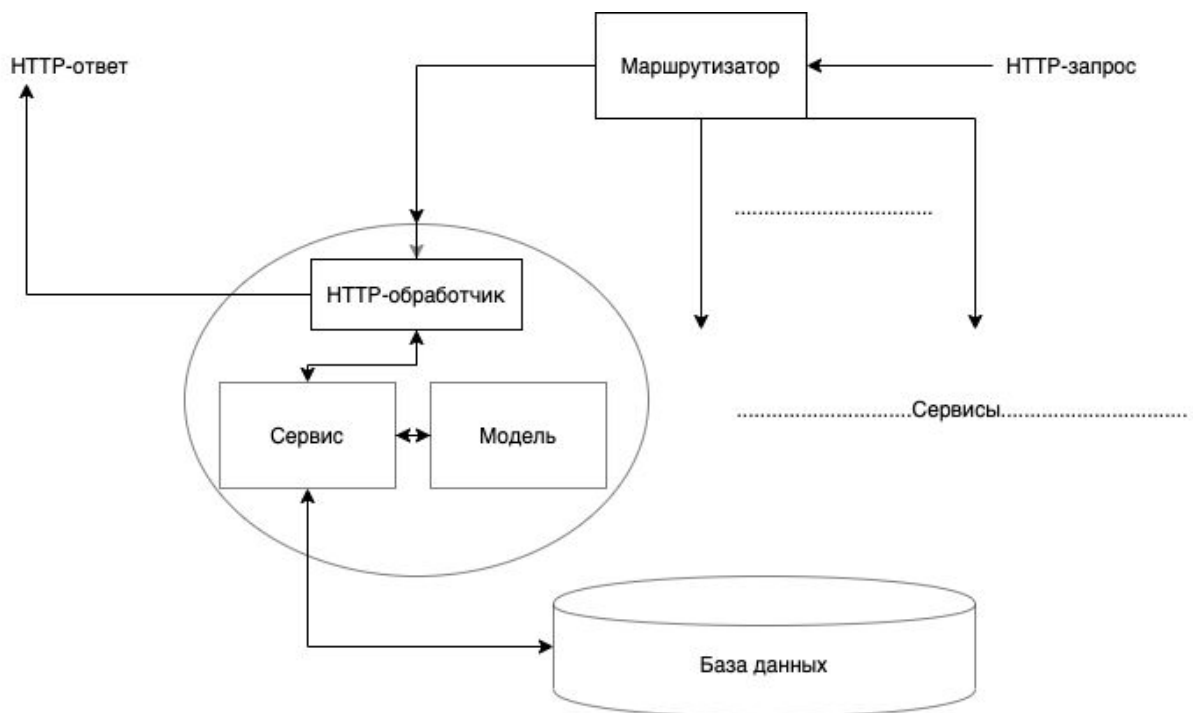


Рисунок 5. Упрощенная архитектура API-сервера.

В действительности структура сервера несколько сложнее (см. Рисунок 6).

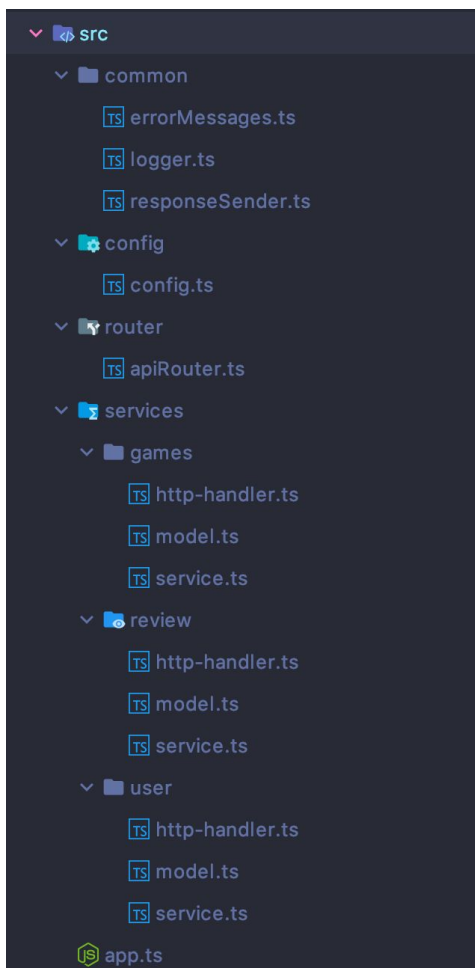


Рисунок 6. Структура API-сервера.

Точка запуска сервера расположена в файле `app.ts`. Здесь устанавливается главный обработчик по пути `/api/`, а также обработчик для всех нераспознанных маршрутов, возвращающий ошибку с кодом 404. Также здесь происходит установка всех промежуточных обработчиков запросов: `cors` – обработчик, обеспечивающий CORS (Cross-Origin Resource Sharing), механизм, использующий дополнительные HTTP-заголовки, чтобы дать возможность агенту пользователя получать разрешения на доступ к выбранным ресурсам с сервера на источнике (домене), отличном от того, что сайт использует в данный момент [14], `bodyparser` – обработчик, осуществляющий синтаксический разбор тела POST-запроса, `session` – обработчик, необходимый для осуществления аутентификации пользователей с помощью файлов пользовательского

контекста и сессий. При его инициализации также происходит инициализация резидентного хранилища для сессий, в качестве которого был выбран Redis – резидентная система управления базами данных класса NoSQL с открытым исходным кодом, работающая со структурами данных типа “ключ — значение” [15].

Файл `config.ts` содержит класс, реализующий шаблон проектирования Singleton (экземпляр данного класса – единственный) и инкапсулирующий конфигурационную информацию приложения, считанную из `.env`, такую как: хост и порт для запуска сервера, токен для подписи файла пользовательского контекста, `redis`-клиент и набор соединений с `Postgres`;

В директории `common` содержатся общие для всех сервисов сущности: `errorMessages` – набор строковых констант, представляющих сообщения об ошибках, `Logger` – класс, инкапсулирующий логику логирования в стандартные потоки вывода и ошибок, `responseSender.ts` – файл, в котором содержатся сущности, представляющие тело HTTP-ответа API-сервера, и функции, инкапсулирующие логику успешного ответа и ответа с ошибкой.

Главный маршрутизатор приложения расположен в файле `apiRouter.ts`. Он определяет маршруты для всех `http`-обработчиков.

В директории `services` расположены все API-сервисы, каждый из которых состоит из трех частей: `http-handler.ts` – устанавливает локальные маршруты и обрабатывает входящие HTTP-запросы, `model.ts` – содержит типы данных, необходимые для работы сервиса, `service.ts` – содержит бизнес-логику приложения, активно взаимодействует с СУБД.

Всего в API-сервере имеется три сервиса: `games` – сервис, содержащий API-методы для получения информации о настольных играх, механиках и категориях. Стоит отметить, что возвращаемая информация о настольной игре расширена данными о связанных с ней механиках, категориях и обзорах, а также количеством “лайков”.

Сервис user содержит API-методы для авторизации пользователей, получения данных о пользователе, действий с пользовательским списком любимых игр. Стоит отметить, что возвращаемые данные о пользователе обогащены информацией о его любимых играх, написанных им обзорах, и поставленных им рейтинговых отметках. При регистрации пользователя в базу данных не записывается пароль в его исходном виде, вместо этого используется метод хеширования пароля с использованием криптографической соли [16], затем в базе данных сохраняется именно этот хеш.

Сервис review содержит API-методы для создания обзора, его получения и обновления его рейтинга. Бизнес-логика, сосредоточенная в данном сервисе несколько сложнее, чем в остальных сервисах, здесь уже явно задействован механизм транзакций. Рассмотрим, например, метод обновления рейтинга у обзора (см. Листинг 1). Сначала происходит проверка существования рецензии и анализ возможных ситуаций по обновлению рейтинга (см. Таблицу 3).

Таблица 3. Возможные ситуации при обновлении рейтинга рецензии.

Рейтинг в запросе	Рейтинг, уже поставленный пользователем на данный момент		
	+	-	NULL
+	ERROR	NULL	+
-	NULL	ERROR	-

Если проверки успешны, то из пула соединений берется одно. Далее расположен блок try-catch-finally, необходимый для корректного освобождения соединения, отката транзакции в случае неудачи. В рамках транзакции происходит попытка вставки/удаления рейтинговой записи, обновления рейтинга обзора и кармы соответствующего пользователя.

Листинг 1. Функция обновления рейтинга у обзора.

```

1. export async function updateReviewRating(
2.     reviewId: string,
```

```

3.         userId: string,
4.         isPositive: boolean
5. ): Promise<RatingResponse> {
6.     const review = await getReviewById(reviewId);
7.
8.     if (!review) {
9.         throw new Error(ErrorMessage.REVIEW_DOESNT_EXIST);
10.    }
11.
12.    const rating = await getReviewRating(reviewId, userId);
13.
14.    if (
15.        rating &&
16.        ((rating.is_positive && isPositive) || (!rating.is_positive && !isPositive))
17.    ) {
18.        throw new Error(ErrorMessage.INCORRECT_RATING);
19.    }
20.
21.    const client = await ServerConfig.get().pool.connect();
22.
23.    try {
24.        await client.query('BEGIN');
25.        const is_positive = rating ? null : isPositive;
26.
27.        if (!rating) {
28.            await insertReviewRating(client, reviewId, userId, isPositive);
29.        } else {
30.            await deleteReviewRating(client, reviewId, userId);
31.        }
32.
33.        const new_score = await updateReviewScore(client, reviewId, isPositive);
34.        await updateUserKarma(client, review.user_id, isPositive);
35.        await client.query('COMMIT');
36.
37.        return { is_positive, new_score };
38.    } catch (e) {
39.        await client.query('ROLLBACK');
40.        throw e;
41.    } finally {
42.        client.release();
43.    }
44. }

```

3.3. Реализация клиентской части приложения

Клиентская часть веб-приложения написана на JavaScript с помощью Vue.js – библиотеки для создания пользовательских интерфейсов [17]. Для создания проекта использовался официальный инструмент Vue CLI, который помог настроить следующие инструменты разработки:

1. Webpack – статический сборщик модулей для современных JavaScript приложений. В процессе обработки приложения он строит граф зависимостей, затем собирает их все в один пакет, а также способствует минификации, позволяет транспилировать код CSS-препроцессоров, однофайловых vue-компонентов, предоставляет сервер для разработки [18];
2. Babel – транспилятор JavaScript кода в более старые обратно совместимые версии JavaScript для охвата более широкого класса браузеров [19];
3. ESLint – инструмент для статического анализа JavaScript/TypeScript кода [20].

Для осуществления маршрутизации на стороне клиента без перезагрузки страницы использовался vue-router – официальное дополнение к библиотеке Vue.js, а для стилизации пользовательского интерфейса использовалась библиотека Vuetify, созданная для фреймворка Vue.js и соответствующая принципам Material Design [21].

Основным элементом пользовательского интерфейса является переиспользуемый vue-компонент, инкапсулирующий HTML-шаблон, JavaScript код и css/scss-стили, необходимые для его функционирования. С точки зрения паттерна MVVM [22] vue-компонент представляет собой модель представления, т.к. он реактивно связан с представлением (своим HTML-шаблоном) и получает данные с помощью сервисов.

Сервисы являются JavaScript-модулями, содержащими логику взаимодействия с API-сервером с помощью библиотеки axios и хранящими полученные данные. Во vue-компоненты они передаются посредством механизма внедрения зависимостей [23]: родителем всех vue-компонентов является компонент App.vue, при его создании в его свойства передаются все инстанции сервисов, которые он затем передает во все дочерние компоненты по запросу. Сервисы являются единственным источником по отношению к данным приложения. Они позволяют компонентам подписаться на связанный с данными поток событий, все изменения данных происходят через сервисы, которые реагируют на них путем передачи в поток нового события, которое получают все подписчики и изменяют пользовательский интерфейс. Достигается этот эффект во многом за счет самой библиотеки Vue.js, которая по своей сути является реактивной, но также большую роль в управлении состоянием приложения играет библиотека RxJS [24].

Упрощенное схематическое представление клиентской архитектуры приложения представлено на Рисунке 7.

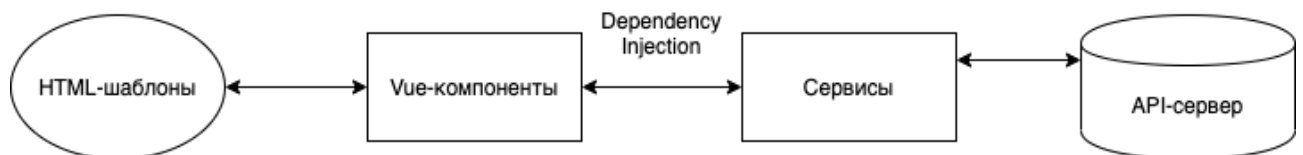


Рисунок 7. Упрощенная frontend-архитектура.

В действительности структура клиентской части приложения немного сложнее (см. Рисунок 8).

В директории public содержится иконка приложения и единственный HTML-файл, содержимое которого будет динамически меняться из JavaScript-кода.

Директория assets содержит png-иконку для навигационной панели приложения.

В директории `components` расположены `vue`-компоненты приложения, о которых чуть позже будет рассказано подробнее.



Рисунок 8. Структура клиентской части приложения.

В папке `helpers` находятся два вспомогательных JavaScript-модуля: `apiError.js` – содержит функцию, которая помогает получить читабельное сообщение об ошибке от API-сервера, `constants.js` – здесь содержатся строковые константы приложения.

Директория `model` содержит JavaScript-классы, объекты которых представляют собой сущности приложения, которые нужно отобразить.

В `plugins` определяются два важных для приложения Vue-плагина: `router.js` – инициализирует клиентскую маршрутизацию, `vuetify.js` – инициализирует `Vuetify`, указывая цветовую тему приложения.

В `services` расположены сервисы приложения. Через сервис `errorService.js` осуществляется показ сообщений об ошибках. Каждое действие, которое может привести к ошибке, во Vue-компоненте оборачивается в блок `try-catch`, где в случае `catch`, сообщение об ошибке передается в `errorService`, который в свою очередь передает сообщение об ошибке в поток событий, на который подписан компонент `Error.vue`. Получив очередное сообщение об ошибке он инициирует его показ пользователю. Таким образом, в приложении реализована реактивная обработка ошибок.

В сервисе `gameService.js` содержится информация о настольных играх, он предоставляет методы Vue-компонентам для взаимодействия с API-сервером. В этом сервисе также реализована логика поиска игр по названию, базирующаяся на вышеописанном реактивном подходе с использованием `RxJS`.

Сервис `userService.js` содержит информацию о текущем авторизованном пользователе, а также предоставляет Vue-компонентам методы для взаимодействия с API-сервером.

Функция, создающая и возвращающая инициализированные объекты всех сервисов, содержится в файле `serviceBuilder.js`.

Родительским компонентом для всех Vue-компонентов, предоставляющим им экземпляры всех сервисов, является `App.vue`. Также он содержит такие глобальные элементы пользовательского интерфейса, как `navigation drawer` и `app-bar`. Именно здесь расположен компонент `Error.vue` и компонент `router-view`, внутри которого находятся компоненты, соответствующие текущему маршруту.

В файле `main.js` происходит инициализация корневого Vue-компонента, с передачей ему необходимых дополнений и объектов сервисов.

3.4. Обзор пользовательского интерфейса

Теперь стоит подробнее рассмотреть все имеющиеся Vue-компоненты (см. Рисунок 9).

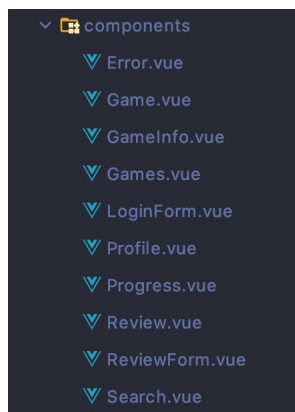


Рисунок 9. Vue-компоненты приложения.

При открытии сайта отображается компонент Games.vue (см. Рисунок 10), который содержит список настольных игр, полученных из базы данных API-сервера.

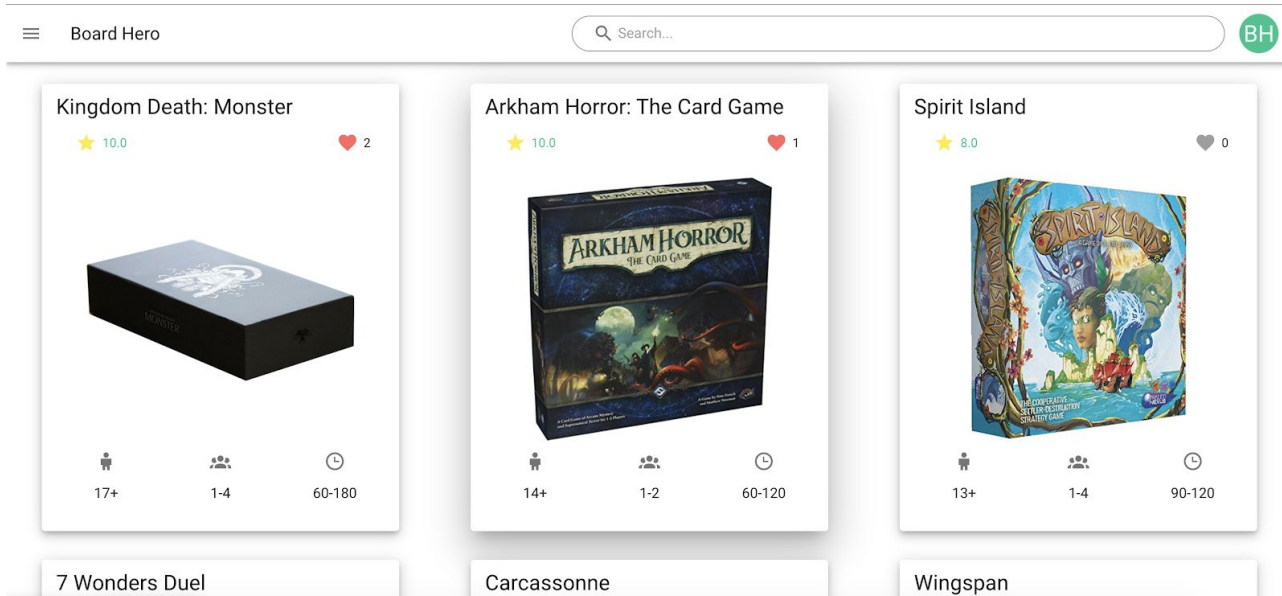


Рисунок 10. Отображение списка настольных игр.

Интерфейс проектировался гибким и отзывчивым (англ. responsive), например, при открытии боковой навигационной панели список игр будет адаптироваться (см. Рисунок 11).

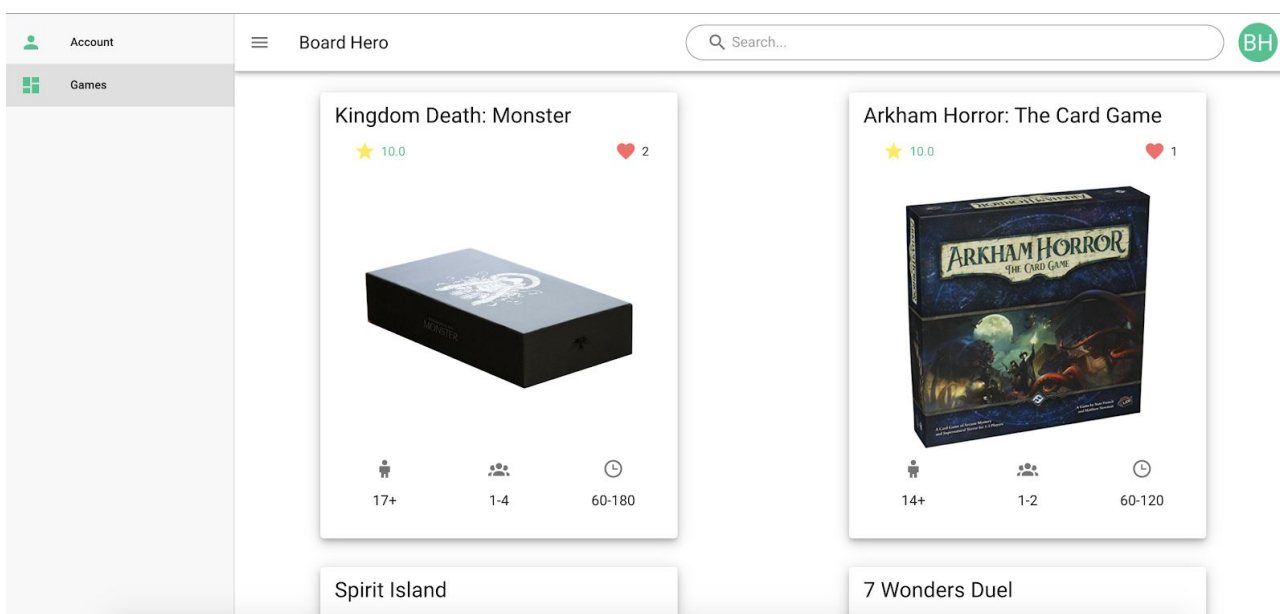


Рисунок 11. Навигационная панель приложения.

Каждый элемент списка представлен компонентом `Game.vue` (см. Рисунок 12). Зарегистрированный пользователь может взаимодействовать с двумя кнопками в верхней части этого компонента. Правая позволяет добавлять (или удалять) игру в список любимых игр, а с помощью левой можно попасть на компонент `ReviewForm.vue` (см. Рисунок 13), если на данный момент пользователь еще не написал на нее обзор.

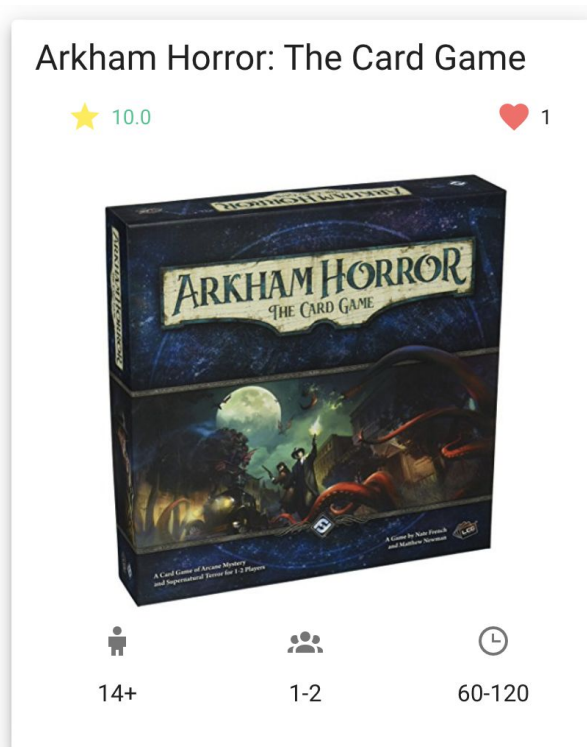


Рисунок 12. Компонент настольной игры.

The screenshot shows the 'Board Hero' header with a search bar and a 'BH' logo. Below the header, there are ten green stars for rating. A form with a red border contains a 'Title' field with a red error message 'Title is required' and a 'Description' text area with a green border and a green 'G' icon in the bottom right corner. Below the form is a green 'SEND REVIEW' button.

Рисунок 13. Форма создания обзора к игре.

Внутри верхней навигационной панели расположен компонент `Search.vue`, с помощью которого можно осуществлять поиск настольных игр по названию (см. Рисунок 14).

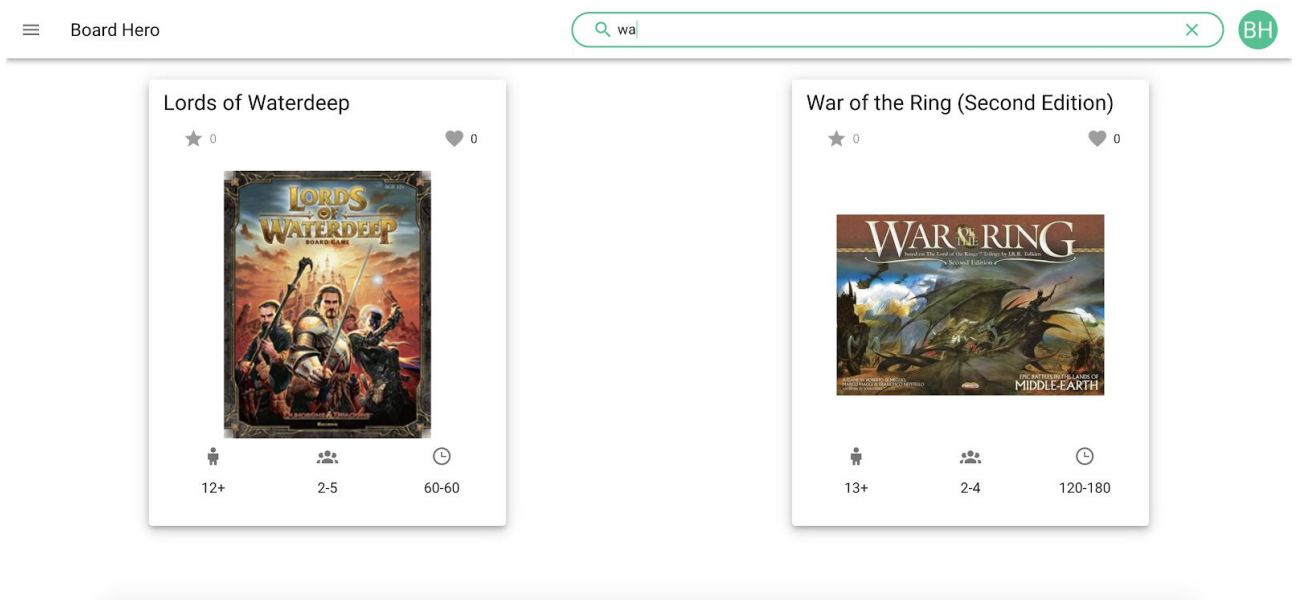


Рисунок 14. Поиск настольных игр по названию.

Нажав на изображение игры, расположенное на игровой карточке, можно переместиться на компонент `GameInfo.vue` (см. Рисунок 15), который содержит полное описание настольной игры, включающее в себя связанные игровые механики и категории, а также список обзоров данной игры (см. Рисунок 16).


Зарегистрированные пользователи имеют возможность повышать/понижать рейтинг конкретного обзора, влияя тем самым на карму автора данной рецензии.

Из боковой навигационной панели можно попасть не только на список настольных игр, но и в свой личный кабинет (см. Рисунок 17), в котором можно увидеть информацию о пользователе, а также списки его рецензий и любимых игр, между которыми можно переключаться, используя элемент пользовательского интерфейса tab.

Board Hero

Search...

BH



Arkham Horror: The Card Game (2016)

★ 10.0

♥ 1

14+

1-2

60-120

Deck Building

Hand Management

Cooperative Play

Adventure

Card Game

Fantasy

Horror

"Something evil stirs in Arkham, and only you can stop it. Blurring the traditional lines between roleplaying and card game experiences, Arkham horror: the card game is the living card game of love craft | an mystery, monsters, and madness! you and your friend (or up to three friends with two core sets) become characters within the quiet new England town of Arkham. You have your talents, sure, but you also have your flaws. Perhaps you've dabbled a little too much in the writings of the Necronomicon, and its words continue to haunt you. Perhaps you feel compelled to cover up any

Рисунок 15. Расширенное описание настольной игры.

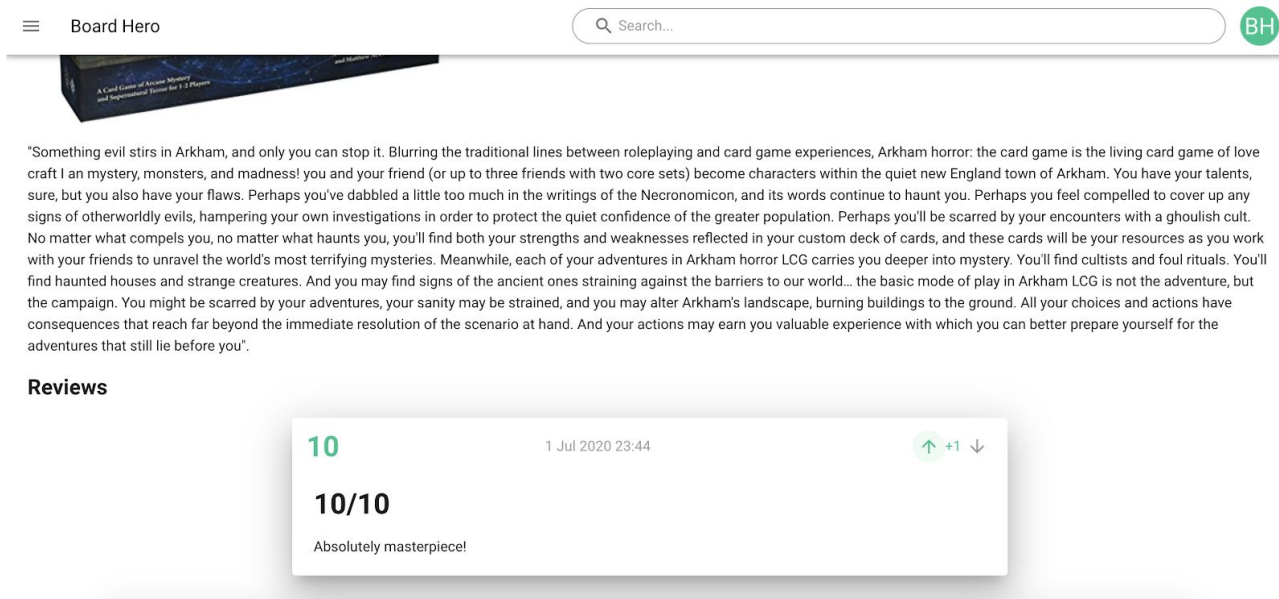


Рисунок 16. Список обзоров к игре.

Если же пользователь не авторизован, то при попытке войти в профиль, он будет перенаправлен на форму логина, представленную компонентом LoginForm (см. Рисунок 18). Обе формы приложения валидируются перед отправкой запроса на API-сервер.

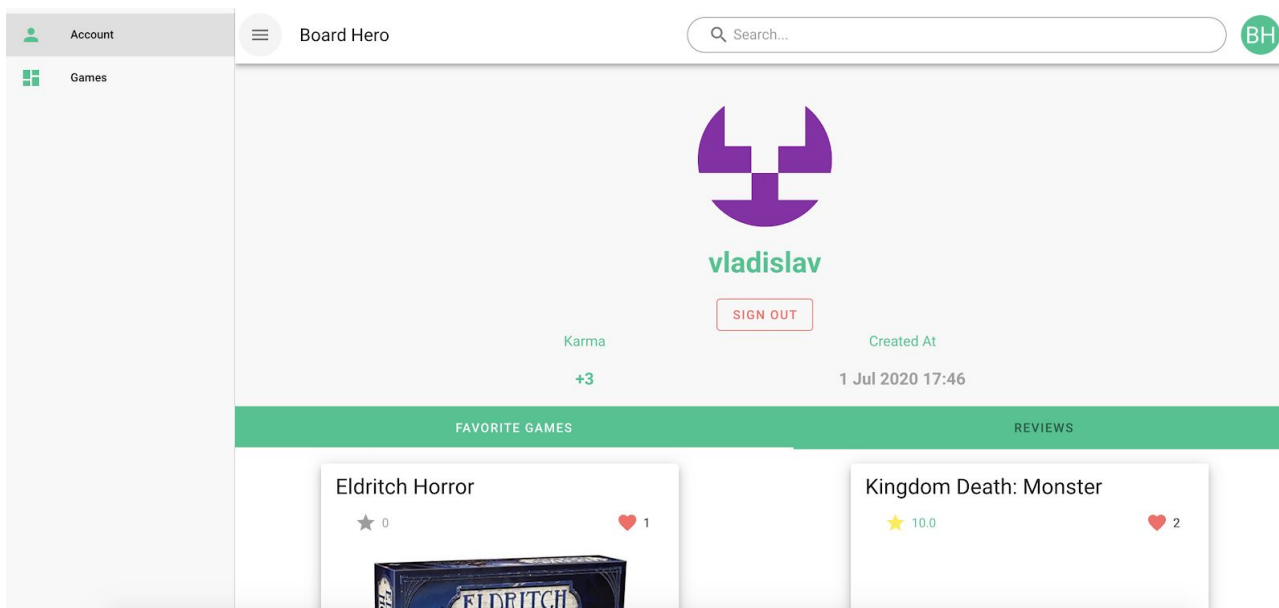


Рисунок 17. Профиль пользователя.

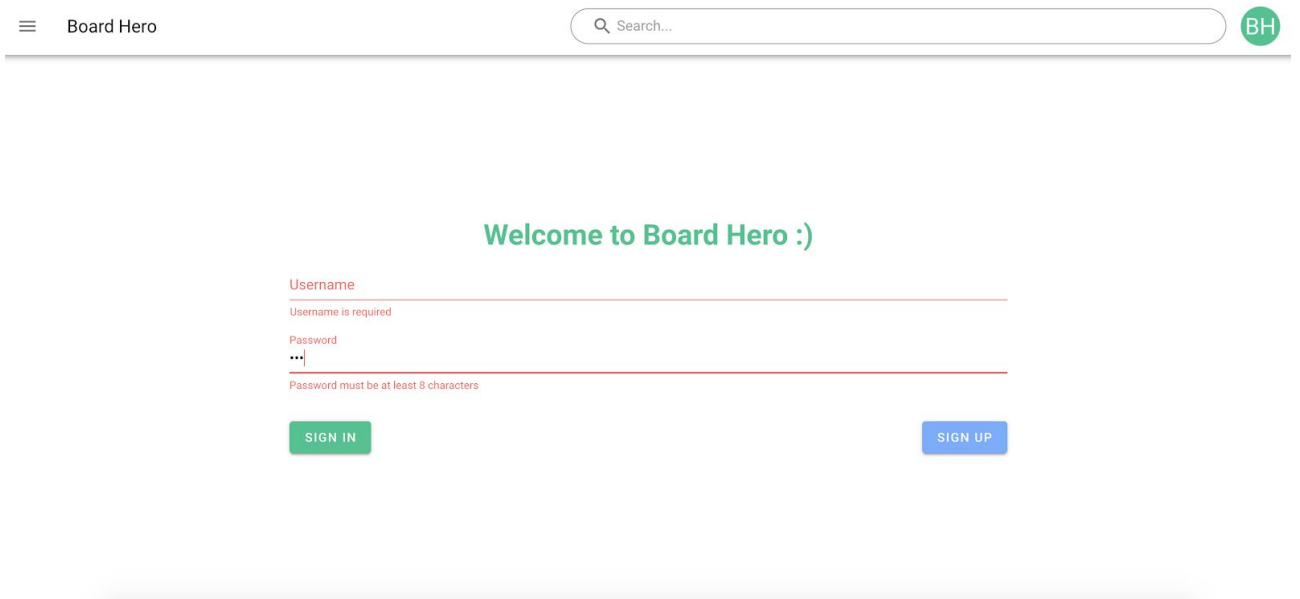


Рисунок 18. Форма логина.

В случае возникновения ошибки в приложении, пользователь получит об этом уведомление в виде элемента пользовательского интерфейса `snackbar` (см. Рисунок 19).

Также во многих компонентах приложения используется компонент `Progress.vue`, представляющий собой спиннер для отображения состояния загрузки данных.

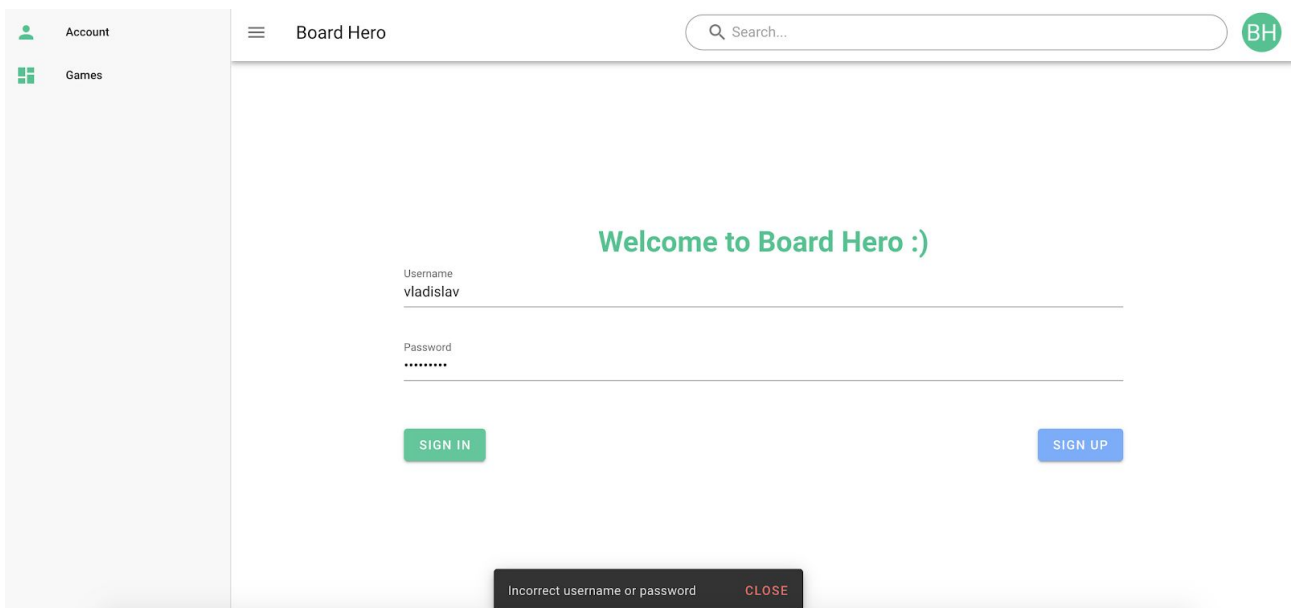


Рисунок 19. Отображение сообщения об ошибке.

4. Тестирование приложения

В процессе создания приложения проводились постоянные проверки корректности его работы.

На начальном этапе реализации базы данных тестировались написанные `sql` и `shell` скрипты, которые в дальнейшем хорошо послужили для автоматизации рабочих процессов. Особую роль играл скрипт наполнения базы данными, полученными из стороннего API. Для уменьшения количества ошибок в процессе разработки было принято решение писать серверный код на TypeScript и дополнительно использовать линтер.

В процессе создания API-сервера использовался дополнительный инструмент `nodemon` [25], отслеживающий изменения в рабочей директории и инициирующий пересборку и перезапуск сервера. В целях тестирования также был продуман механизм логирования для получения конкретной реакции сервера на каждый запрос и подробной информации о возникшей ошибке, включая описание стека вызовов.

Тестирование написанных методов API производилось с помощью программы Postman [26], предоставляющей удобный пользовательский интерфейс для осуществления запросов к серверу (см. Рисунок 20).

На первом этапе реализации API-сервера были проведены следующие тестовые запросы (см. Таблицу 4).

Таблица 4. Тестовые запросы на первом этапе разработки API.

Маршрут запроса	Ожидаемый результат
<code>/api/games</code>	список настольных игр, обогащенный информацией о механиках и категориях
<code>/api/games/mechanics</code>	список игровых механик
<code>/api/games/categories</code>	список игровых категорий
прочие произвольные маршруты	404 Not Found

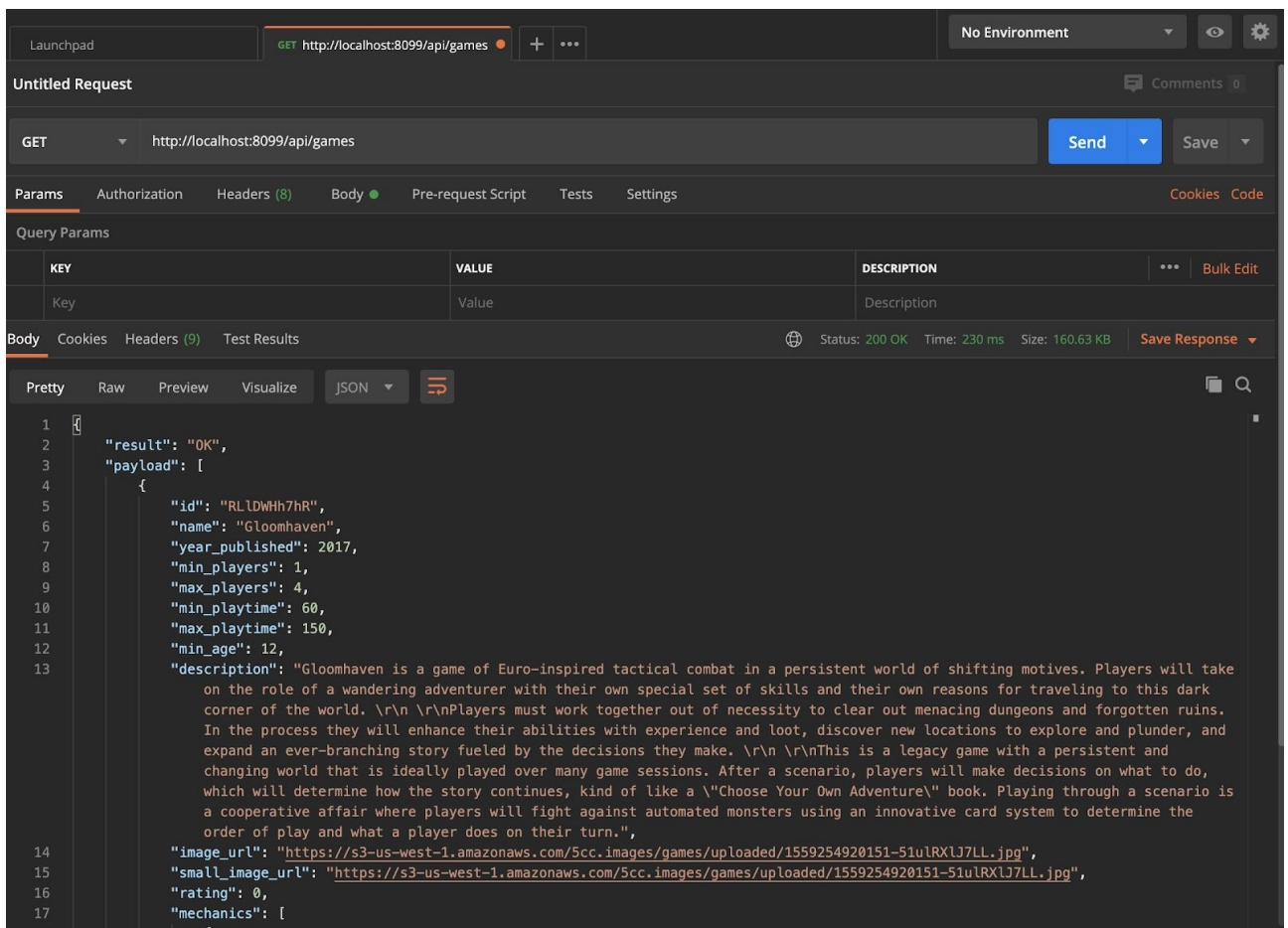


Рисунок 20. Тестирование API с помощью Postman.

Следующим важным шагом было тестирование процесса авторизации и аутентификации, который состоял из следующих этапов:

1. Клиент делает запрос на получение профиля без предоставленного пользовательского контекста. Сервер отвечает ему кодом 401 UNAUTHORIZED;
2. Клиент осуществляет запрос на регистрацию/авторизацию с помощью имени и пароля. Если все прошло успешно, то сервер должен проставить в ответе клиенту контекст;
3. Клиент повторно совершает запрос на получение профиля, но уже с предоставленным контекстом. Сервер авторизует запрос;
4. Клиент делает запрос на выход из профиля. Сервер уничтожает пользовательскую сессию.

Также необходимо было протестировать корректное масштабирование хранения сессий с помощью Redis.

Также при тестировании необходимо было предусмотреть множество крайних случаев для каждого метода. Например, случаи некорректного тела POST-запроса, неверный пароль или несуществующий пользователь в случае запроса на вход в аккаунт, возможные ситуации при обновлении рейтинга рецензии и т.д.

В процессе разработки клиентской части приложения тестировались по большей части те же случаи, что и при разработке API-сервера, но уже с позиции пользовательского интерфейса. Особое внимание уделялось тестированию главных функций приложения:

1. Отображение списка настольных игр;
2. Просмотр информации о конкретной игре и обзорах на нее;
3. Поиск игр по названию;
4. Создание аккаунта + механизм авторизации;
5. Просмотр профиля пользователя, включая рецензии и любимые игры;
6. Процесс написания обзора к игре;
7. Добавление/удаление любимой игры;
8. Повышение/понижение рейтинга обзора + механизм изменения кармы.

Также была настроена информативная для пользователя обработка ошибок.

В результате тестирования установлено, что приложение удовлетворяет всем требованиям, сформулированным при проектировании.

Заключение

В ходе выполнения курсовой работы было реализовано веб-приложение для обмена информацией о настольных играх, которое включило в себя:

1. Базу данных настольных игр;
2. API-сервер;
3. Клиентское приложение.

Среди дальнейших планов по улучшению приложения стоит выделить следующие:

1. Развернуть приложение в облаке, например используя платформу Heroku [27];
2. Сгенерировать документацию к REST API, например используя Swagger [28];
3. Добавить в приложение прочие полезные функции, такие как возможность создания обсуждений под настольными играми, возможность просмотра профилей других пользователей, расширение механизма кармы и т.д.

Список использованных источников

1. Питер Пин-Шен Чен. Модель «сущность-связь» — шаг к единому представлению о данных. Пер. М. Р. Когаловского.
2. Дж. Куроуз, К. Росс. Компьютерные сети: нисходящий подход. — М.: Издательство «Э», 2016
3. Board Game Atlas API.
URL: <https://www.boardgameatlas.com/api/docs> (дата обращения 02.05.2020)
4. Introducing JSON.
URL: <https://www.json.org/json-ru.html> (дата обращения 02.05.2020)
5. Cookie HTTP. URL:
<https://developer.mozilla.org/ru/docs/Web/HTTP/%D0%9A%D1%83%D0%BA%D0%B8> (дата обращения 02.05.2020)
6. Chapter 5 of Roy Fielding’s dissertation “Representational State Transfer (REST)”.
URL: https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm (дата обращения 02.05.2020)
7. М. Миковски, Дж. Пауэлл. Разработка одностраничных веб-приложений = Single Page Web Applications: JavaScript End-to-end. — ДМК Пресс, 2014. — 512 с. — ISBN 978-5-457-83457-6.
8. Что такое PostgreSQL?
URL: <https://postgrespro.ru/docs/postgresql/12/intro-what-is> (дата обращения 02.05.2020)
9. Postgres.app. URL: <https://postgresapp.com/> (дата обращения 02.05.2020)
10. TypeScript. URL: <https://www.typescriptlang.org/index.html> (дата обращения 02.05.2020)
11. Node.js. URL: <https://nodejs.org/en/> (дата обращения 02.05.2020)

12. Node-postgres. URL: <https://node-postgres.com/> (дата обращения 02.05.2020)
13. Express.js. URL: <https://expressjs.com/> (дата обращения 02.05.2020)
14. Cross-Origin Resource Sharing (CORS). URL: <https://developer.mozilla.org/ru/docs/Web/HTTP/CORS> (дата обращения 02.05.2020)
15. Redis. URL: <https://redis.io/> (дата обращения 02.05.2020)
16. Storing password the right way.
URL: <https://www.bigomega.dev/passwords-in-node> (дата обращения 02.05.2020)
17. Vue.js. URL: <https://ru.vuejs.org/v2/guide/> (дата обращения 02.05.2020)
18. Webpack. URL: <https://webpack.js.org/concepts/> (дата обращения 02.05.2020)
19. Babel. URL: <https://babeljs.io/docs/en/> (дата обращения 02.05.2020)
20. ESLint. URL: <https://eslint.org/docs/user-guide/getting-started> (дата обращения 02.05.2020)
21. Vuetify.js. URL: <https://vuetifyjs.com/ru/> (дата обращения 02.05.2020)
22. MVVM.
URL: [https://docs.microsoft.com/en-us/previous-versions/msp-n-p/hh848246\(v=pandp.10\)?redirectedfrom=MSDN](https://docs.microsoft.com/en-us/previous-versions/msp-n-p/hh848246(v=pandp.10)?redirectedfrom=MSDN) (дата обращения 02.05.2020)
23. Р. Мартин. Clean Code: A Handbook of Agile Software Craftsmanship. — Pearson Education, 2008. — P. 157. — ISBN 978-0-13-608325-2.
24. RxJS. URL: <https://rxjs-dev.firebaseapp.com/> (дата обращения 02.05.2020)
25. Nodemon. URL: <https://www.npmjs.com/package/nodemon> (дата обращения 02.05.2020)
26. Postman. URL: <https://www.postman.com/> (дата обращения 02.05.2020)
27. Heroku. URL: <https://www.heroku.com/> (дата обращения 02.05.2020)
28. Swagger. URL: <https://swagger.io/> (дата обращения 02.05.2020)