МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего образования
**«Южно-Уральский государственный университет
(национальный исследовательский университет)»**
Высшая школа электроники и компьютерных наук
Кафедра системного программирования

РАБОТА ПРОВЕРЕНА

Доцент кафедры ИИТиМОИ
ЮУрГГПУ Дмитриева О.А.
“___”_____ 2020 г.

ДОПУСТИТЬ К ЗАЩИТЕ

Заведующий кафедрой,
д.ф.-м.н., профессор
_____ Л.Б. Соколинский
“___”_____ 2020 г.

Разработка веб-приложения «WhaToDo» для навигации по достопримечательностям

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА
ЮУрГУ – 02.03.02.2020.115-124.ВКР

Научный руководитель,
к.ф.-м.н., доцент кафедры СП
_____ И.И. Клебанов

Автор работы,
студент группы КЭ-401
_____ Д.А. Жуков

Ученый секретарь
(нормоконтролер)
_____ И.Д. Володченко
“___”_____ 2020 г.

Челябинск–2020

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего образования
«Южно-Уральский государственный университет
(национальный исследовательский университет)»
Высшая школа электроники и компьютерных наук
Кафедра системного программирования

УТВЕРЖДАЮ

Зав. кафедрой СП

_____ Л.Б. Соколинский

10.02.2020

ЗАДАНИЕ
на выполнение выпускной квалификационной работы бакалавра
студенту группы КЭ-401 Жукову Дмитрий Андреевичу,
обучающемуся по направлению
02.03.02 «Фундаментальная информатика и информационные технологии»

1. Тема работы (утверждена приказом ректора от 24.04.2020 № 627)

Разработка веб-приложения "WhaToDo" для навигации по достопримечательностям

2. Срок сдачи студентом законченной работы: 15.06.2020.

3. Исходные данные к работе:

3.1. Справочник по HTML. [Электронный ресурс] URL: http://htmlbook.ru

3.2. Документация по Sapper. [Электронный ресурс]
URL: https://ru.sapper.svelte.dev/docs

3.3. Справочник по JavaScript [Электронный ресурс]
URL: https://learn.javascript.ru

4. Перечень подлежащих разработке вопросов

4.1. Разработать Frontend с использованием фреймворков Svelte и Sapper

4.2. Разработать Backend с использованием Node.js, JavaScript в формате REST API

4.3. Реализовать авторизацию с JsonWebToken

4.4. Реализовать динамические URL

4.5. Реализовать адаптивная верстку с использованием SCSS

4.6. Добавить возможность определения геолокации пользователя

5. Дата выдачи задания: 10.02.2020.

Научный руководитель,
к.ф.-м.н., доцент кафедры СП И.И. Клебанов

Задание принял к исполнению Д.А. Жуков

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Актуальность темы

В мире активно развиваются технологии, в том числе появляются возможности погружаться в игровой мир с помощью очков виртуальной реальности, различных контроллеров и других приспособлений. Играть становится интереснее, но это отдаляет нас от реальной жизни, которая тоже может быть полна впечатлений.

На одной из станций метро Санкт-Петербурга, один за другим стоят рекламные стенды с надписями «А ты уже посетил все фонтаны Питера?», «А ты уже нашел все статуи львов?». Увидев это, мне пришла идея реализовать сервис, в котором люди смогут находить больше подобных заданий, отмечать их выполнение, сравнивать друг с другом успехи и, конечно, получать впечатления от увиденных мест.

Цель и задачи

Целью работы является реализация сервиса для предложения пользователю мест для прогулок и отслеживания их посещения. Для достижения данной цели должны быть решены следующие задачи:

1) разработать Frontend с использованием фреймворков Svelte и Sapper;
2) разработать Backend с использованием Node.js, JavaScript в формате REST API;
3) реализовать авторизацию с JsonWebToken;
4) реализовать динамические URL;
5) реализовать адаптивная верстку с использованием SCSS;
6) добавить возможность определения геолокации пользователя.

Структура и объем работы

Работа состоит из введения, четырех глав, заключения, списка литературы и одного приложения. Объем работы составляет 42 страницы, список литературы – 15 источников.

В главе «Теоретическая часть» дано описание актуальности темы работы, проведен обзор аналогов, а также выбор средств реализации веб-приложения.

Глава «Проектирование веб-приложения» посвящена определению требований к разрабатываемому мобильному приложению. Определены функциональные и нефункциональные требования, спроектированы база данных и интерфейсы приложения, представлена диаграмма вариантов использования.

В главе «Реализация веб-приложения» рассмотрена архитектура проекта и приведены примеры исходных кодов.

Глава «Тестирование» посвящена результатам тестирования приложения. Представлены результаты функционального тестирования, выполненные в работающем приложении.

В заключении сделаны выводы о проделанной работе и сформулированы перспективы дальнейшей разработки.

1. ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

1.1. Постановка задачи

WhaToDo хоть и имеет развлекательный характер, несет в себе более глубокий смысл. Одной из ключевых задач стоит привлечение людей к исследованию интересных мест России и облегчение поиска этих мест. Однако, на первом этапе следует взять менее большую территорию, а именно – только город Челябинск. Необходимо создать пользователю условия для поиска и отслеживания посещений примечательных мест города и постараться вызвать интерес к этому занятию. Для этого необходимо сделать статистику, удобный интерфейс, гибкую возможность создания своих списков мест для посещений и добавить достаточно информации, чтобы заинтересовать пользователя.

Решать эту задачу стоит созданием веб-приложения, потому что тогда можно быть уверенным в кроссплатформенности и широком охвате аудитории.

1.2. Обзор аналогов

Полных аналогов разрабатываемого веб-приложения не обнаружено. Однако есть несколько проектов, в которых реализованы похожие идеи.

1.2.1. Система достижений в сервисе цифровой дистрибьюции Steam

Система достижений Steam, изображенная на рисунке 1 состоит в том, что пользователи Steam, приобретая некоторые игры из внутреннего магазина, могут выполнять в них определенные задания, тем самым получая достижения.

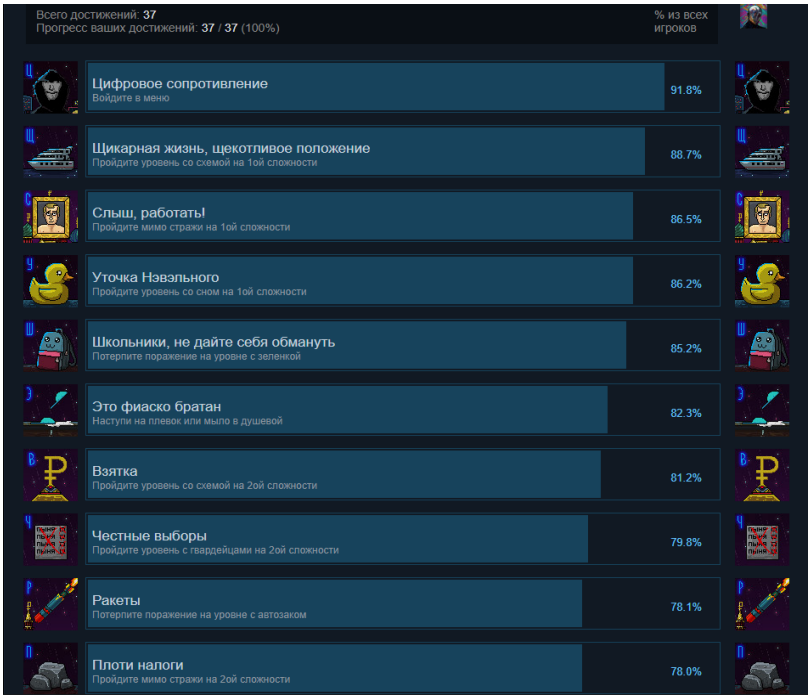

Рис. 1 Система достижений в сервисе цифровой дистрибьюции Steam

Пользователи видят, какие достижения можно получить в конкретной игре, какой процент владельцев этой игры получил данные достижения, и, конечно, какие достижения уже выполнены.

Одной из наиболее явных проблем в системе достижений сервиса Steam является невозможность получить 100% достижений в некоторых случаях. Например, если какая-либо игра выдавала достижения за участие в мероприятиях прошлых лет, получить эти достижения позже становилось невозможно.

1.2.2. Веб-приложение LEVEL UP LIFE

Очень похожим на WhaToDo сервисом является LEVEL UP LIFE, изображенный на рисунке 2. Регистрируя аккаунт на этом сайте, пользователь создает своего виртуального клона, который имеет определенные характеристики и уровень. Сервис предлагает выбирать задания из различных категорий, выполнив которые в реальной жизни, пользователь увеличивает характеристики своего персонажа и получает опыт для повышения уровня.

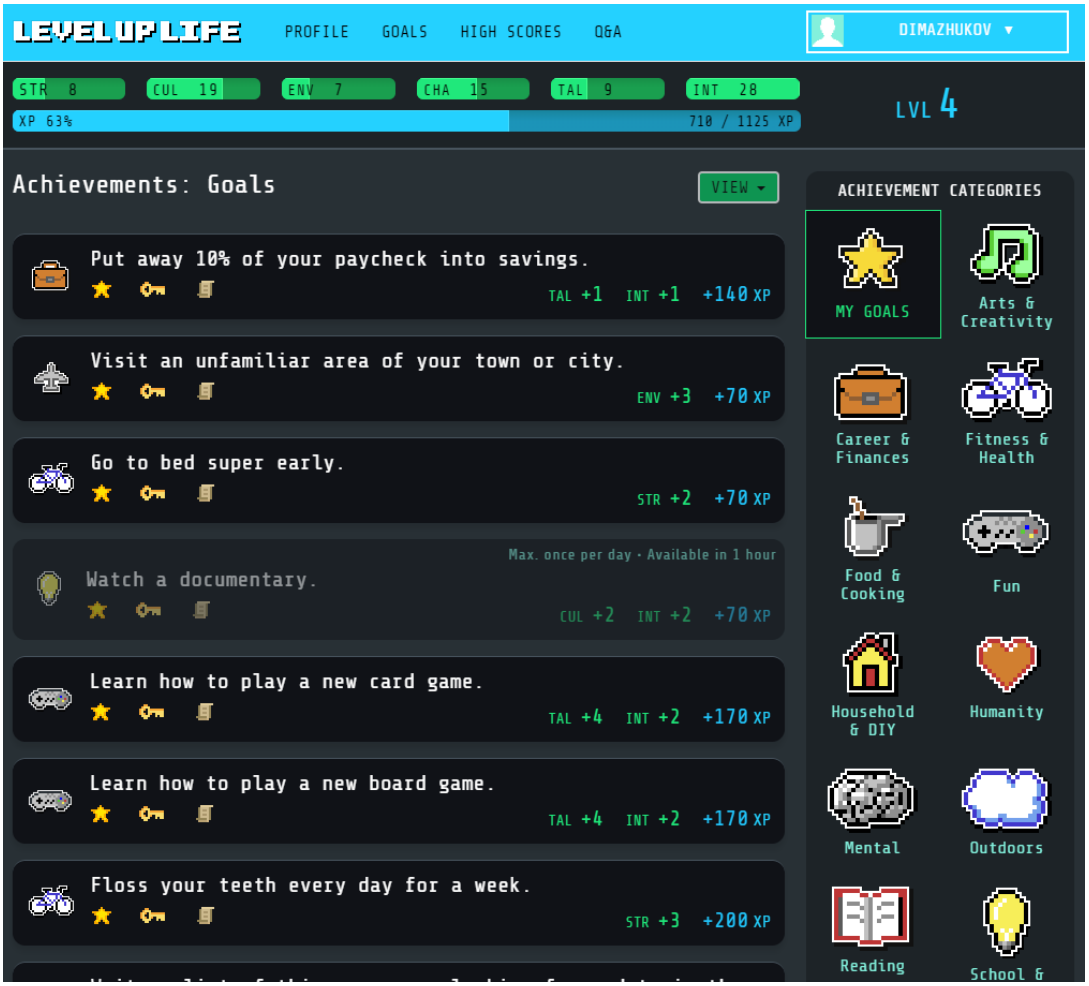

Рис. 2. Веб-приложение LEVEL UP LIFE

Главной проблемой LEVEL UP LIFE является тот факт, что пользователям по сути не нужно что-то делать. Можно зайти в конце дня и просто отметить, что произошло в последнее время, чтобы как-то развить своего виртуального персонажа. Задания вроде «Прочитайте полезную литературу» или «Проведите время на природе/с друзьями», конечно, могут стимулировать пользователя провести день иначе, но с довольно большим выбором ежедневных простых задач, может создаваться впечатление, что твой день прошел активно просто потому, что ты написал кому-то в социальной сети и за пару минут прочитал статью в интернете, так и не встав из-за компьютера.

1.2.3. Приложения PokemonGo, Ingress и подобные

Приложения Ingress и PokemonGO, изображенные на рисунке 3, очень похожи между собой, поэтому нет смысла рассматривать их отдельно.

Рис. 3 Ingress и Pokemon GO

Это приложения для телефонов с операционной системой iOS или Android, дополняющие реальность. Игроки должны ходить по реальному миру, находить игровые предметы с помощью телефона и соревноваться друг с другом во внутриигровом развитии. В Ingress нужно искать и захватывать порталы, представленные различными памятниками в реальном мире, а в PokemonGO нужно искать покемонов и развивать их.

Недостатком этих игр стоит отметить неактуальность игровых достижений в реальной жизни. Если игра закончится, отключат или заблокируют сервера, пользователи фактически останутся ни с чем, ведь полученные впечатления можно будет обсудить только с другими игроками.

Приведенные аналоги значительно отличаются друг от друга, поэтому сложно сравнивать их всех по единым критериям. Наиболее оптимальным решением для подведения итогов будет сформировать список особенностей, которые стоит перенять или наоборот не допустить.

- Пользователю нужно предоставлять статистику, чтобы он мог анализировать сложность достижений и прогресс их выполнения.
- Пользователь всегда должен иметь возможность выполнить все достижения на 100%, но для этого не обязательно отказываться от поощрений за участие в разовых событиях.
- Пользователю нужно предлагать места, которые представляют ценность для людей, незнакомых с сервисом.
- Нужно формировать задания, для выполнения которых нужно приложить хоть какие-то усилия, чтобы их выполнение действительно можно было называть Достижением.

1.3. Средства разработки

В качестве средства разработки веб-приложения была выбрана платформа Node.js.

Node.js - программная платформа, превращающая JavaScript из узкоспециализированного языка в язык общего назначения. В частности, в данном проекте Node.js используется в качестве приложения сервера.

Реализация веб-приложения будет происходить на языках программирования JavaScript (ES6), также с использованием технологий SCSS и фреймворков Svelte/Sapper. Для адаптации к браузеру, будет использоваться сбор-

щик Rollup, преобразующий написанный код так, чтобы он был понятен браузеру, минимизирован и собран в единый файл и множества компонентов.

Для хранения данных об учетных записях пользователей, списка товаров, списка сделок была выбрана СУБД MySQL. Запрос к MySQL будет осуществляться с помощью Node-фреймворка Sequelize.

Для работы с картами был выбран API Yandex Maps. Выбор стоял между Google API и Yandex API, но Google запросили оплату после третьего обращения к их API, а Yandex поставили лимит в 25000 бесплатных обращений в сутки. В ходе более подробного изучения документаций, был сделан вывод, что и этот предел можно не допускать при правильном использовании API.

2. ПРОЕКТИРОВАНИЕ ВЕБ-ПРИЛОЖЕНИЯ

2.1. Анализ требований

Веб-приложение должно отвечать следующим функциональным требованиям.

1. Система должна обеспечивать пользователю возможность выбрать список мест для посещений.

2. Система должна обеспечивать пользователю возможность увидеть список выбранных мест для посещений на карте.

3. Система должна обеспечивать пользователю возможность посмотреть фотографии и описание выбранных для посещения мест.

4. Система должна обеспечивать возможность отмечать посещение заранее выбранных мест.

5. Система должна обеспечивать возможность пользователю видеть, какая часть пользователей выполнила конкретное достижение и какую часть своих достижений он выполнил сам.

Среди нефункциональных требований можно выделить следующие.

1. База данных должна быть реализована с помощью СУБД MySQL.

2. Взаимодействие с картами должно быть реализовано с использованием Yandex Maps API.

3. Корректная работа в следующих браузерах:

- Google Chrome 23 и выше;
- Firefox 21 и выше;
- Opera 15 и выше.

2.2. Варианты использования веб-приложения

С помощью языка графического описания для объектного моделирования UML, была построена диаграмма вариантов использования.

На рисунке 4 представлена диаграмма вариантов использования веб-приложения.

В ходе проектирования был выделен один актер.

Пользователь – пользователь веб-приложения, который зарегистрировался и авторизовался в системе.

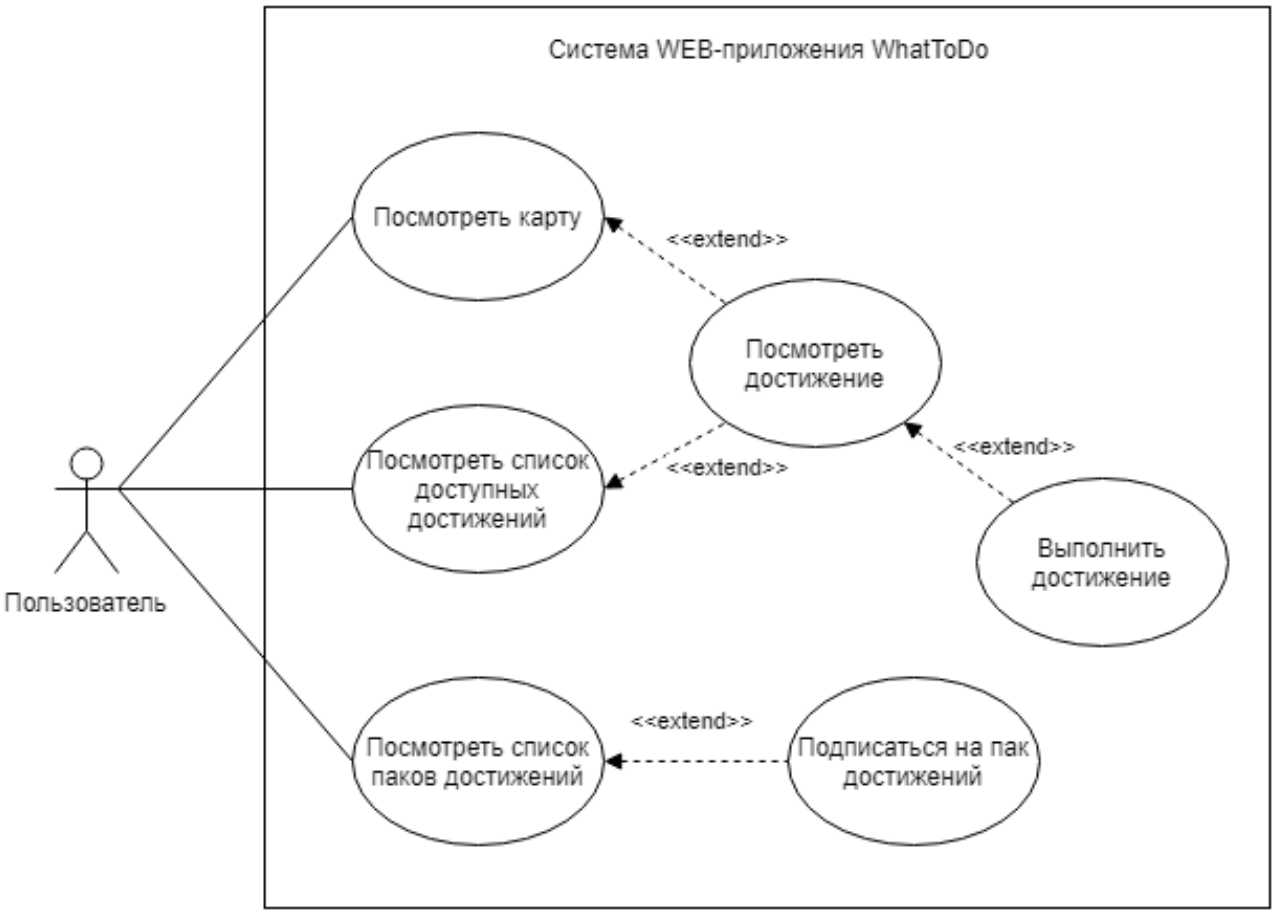

Рис. 4 Диаграмма вариантов использования

Актер может совершать следующие действия.

Посмотреть карту – пользователь может карту с достижениями, которые он может выполнить.

Посмотреть список доступных достижений – пользователь может открыть список достижений, которые он может выполнить.

Посмотреть достижение – пользователь может открыть страницу определенного достижения с фотогалереей.

Выполнить достижение – пользователь может отметить выполнение какого-либо достижения, если оно доступно для выполнения.

Посмотреть список паков достижений – пользователь может открыть список паков (наборов) достижений и увидеть, какие достижения они включают.

Подписаться на пак достижений – пользователь может отметить пак, чтобы получить возможность отслеживать и отмечать выполнение включенных в него достижений.

2.3. Проектирование базы данных

База данных на рисунке 5 необходима для хранения данных пользователей, достижений и записей об их выполнении, наборов достижений и подписках на них, городов, значков и записей об их получении.

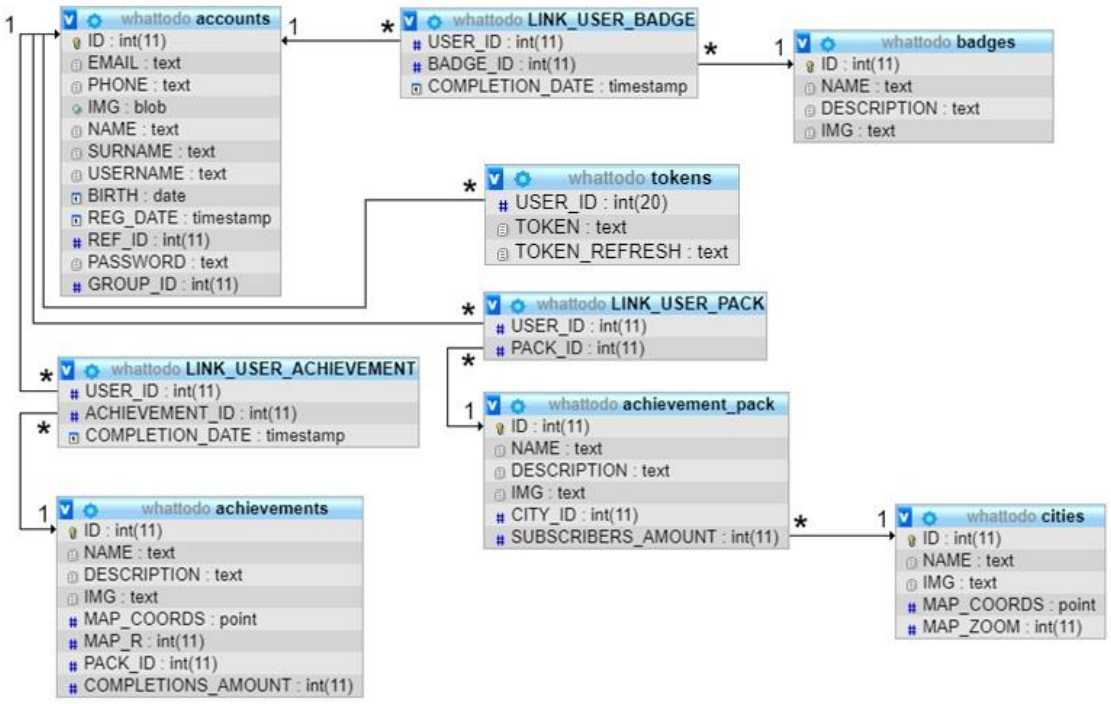

Рис. 5 Схема базы данных

В ходе проектирования были выделены следующие таблицы: accounts, achievements, achievement_pack, badges, cities, LINK_USER_PACK, LINK_USER_BADGE, LINK_USER_ACHIEVEMENT.

accounts – таблица, в которой хранятся данные о пользователях: логин, пароль, дата регистрации, уровень доступа и некоторые личные данные.

achievements – таблица, в которой хранятся данные о достижениях: название, описание, путь к иконке, координаты на карте, радиус области на карте, ID набора, количество выполнивших пользователей.

achievement_pack – таблица, в которой хранятся данные о наборах достижений: название, описание, путь к иконке, ID города, количество подписчиков.

badges – таблица, в которой хранятся данные о значках: название, описание, путь к иконке.

cities – таблица, в которой хранятся данные о городах: название, путь к иконке, координаты на карте, уровень приближения при просмотре на карте.

Впоследствии для реализации связей много-ко-многим, были созданы таблицы **LINK_USER_PACK**, **LINK_USER_BADGE**, **LINK_USER_ACHIEVEMENT**.

LINK_USER_PACK – таблица, в которой хранятся данные о подписках на паки: ID подписавшегося пользователя, ID пака, на который была совершена подписка.

LINK_USER_BADGE – таблица, в которой хранятся данные о получении значков: ID получившего пользователя, ID полученного значка, дата получения.

LINK_USER_ACHIEVEMENT – таблица, в которой хранятся данные о выполнении достижений: ID выполнившего пользователя, ID выполненного достижения, дата выполнения.

2.4. Проектирование интерфейсов

Интерфейс пользователя (*user interface или сокращенно UI*) – это интерфейс, обеспечивающий передачу информации между пользователем-человеком и программно-аппаратными компонентами компьютерной системы.

Экран, который увидит неавторизованный пользователь, когда зайдет на сайт, изображен на рисунке 6. При авторизации, левая часть страницы будет изменена: добавятся пункты меню и информация пользователя: его никнейм и процент выполненных достижений. Кроме того, на карте станут отображены только достижения, на которые пользователь подписан. Выполненные станут другого цвета. Страница сайта примет вид, который указан на рисунке 7.

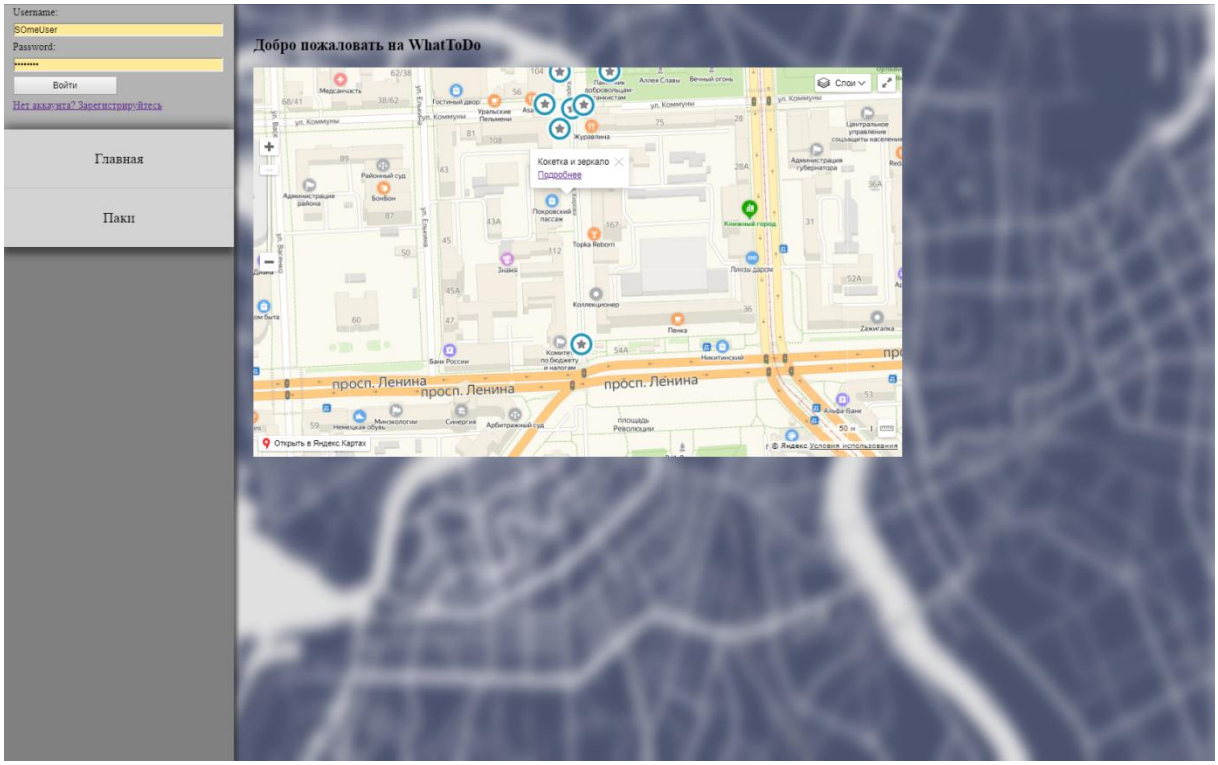

Рис. 6 Главная страница (Неавторизованный пользователь)

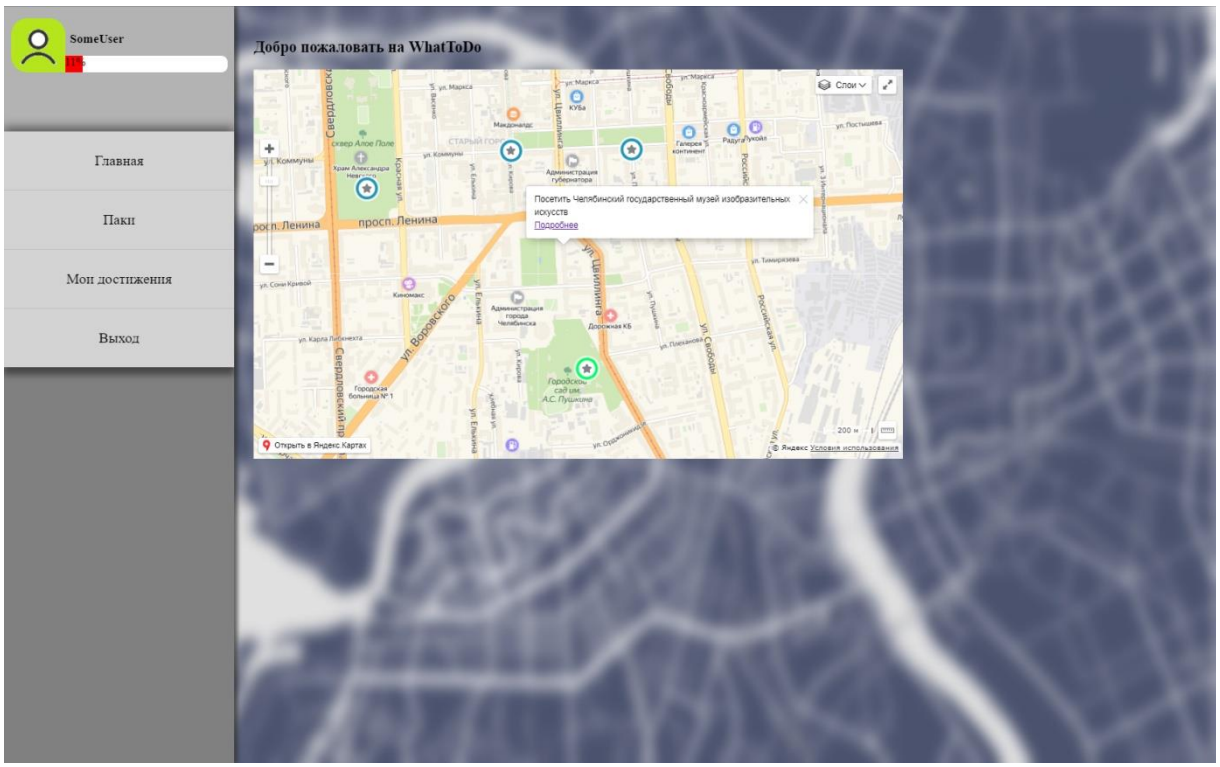

Рис. 7 Главная страница (Авторизованный пользователь)

При переходе на страницу «Паки» пользователь видит экран, изображенный на рисунке 8. При клике на пак, показывается список с описанием вложенных в него достижений.

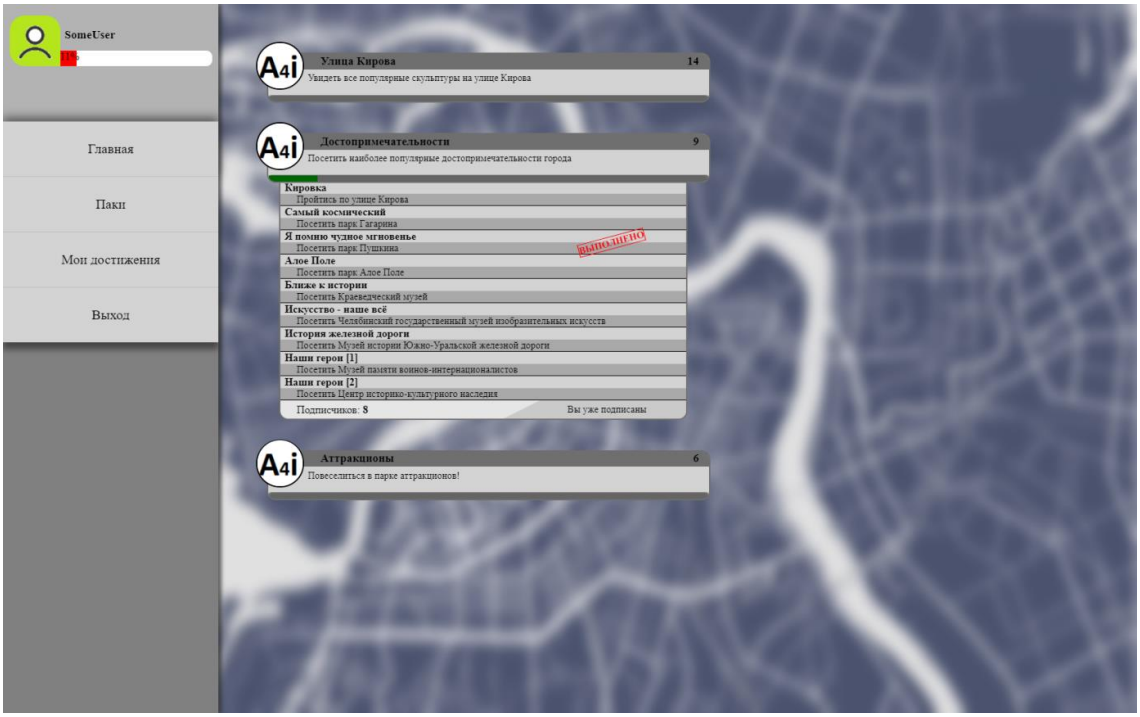

Рис. 8 Страница списка паков

В разных ситуациях, нижняя часть списка достижений оформляется по-разному. На рисунке 8 виден пример, когда пользователь уже подписан на пак. Если же он еще не подписан, он увидит кнопку для подписки.

Если пользователь нажмет кнопку «Подписаться», ему нужно будет подтвердить свое решение во всплывающем окне.

Когда пользователь определится с наборами, которые ему интересно выполнять, он сможет увидеть полный список достижений, которые он может выполнить, на странице «Мои достижения». Пример страницы изображен на рисунке 9.

Рис. 10 Страница доступных достижений пользователя

Чтобы не влиять на статистику пользователя, но отмечать его достижения в разовых мероприятиях, были введены значки. Они отображаются под иконкой пользователя, если имеются. Примеры на рисунке 10.

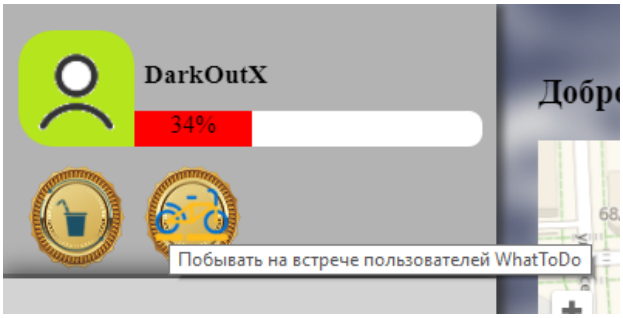

Рис. 9 Значки пользователя

3. РЕАЛИЗАЦИЯ ВЕБ-ПРИЛОЖЕНИЯ

3.1. Архитектура проекта

Проект включает в себя ряд компонентов, изображенных на рисунке 11. Компонент «Интерфейс пользователя» отвечает за отображение элементов непосредственно на веб-странице и содержит файлы js-скриптов анимаций и CSS-стилей. Компонент «Сервер» отвечает за логику взаимодействия между «Клиент» и базой данных и представляет из себя REST API, запущенный как отдельный процесс. Компонент «Клиент» содержит файлы, обеспечивающие работу пользователя и сервера, он также запускается отдельным процессом. Компонент «База данных» представляет собой реализацию хранения данных веб-приложения с использованием СУБД MySQL.

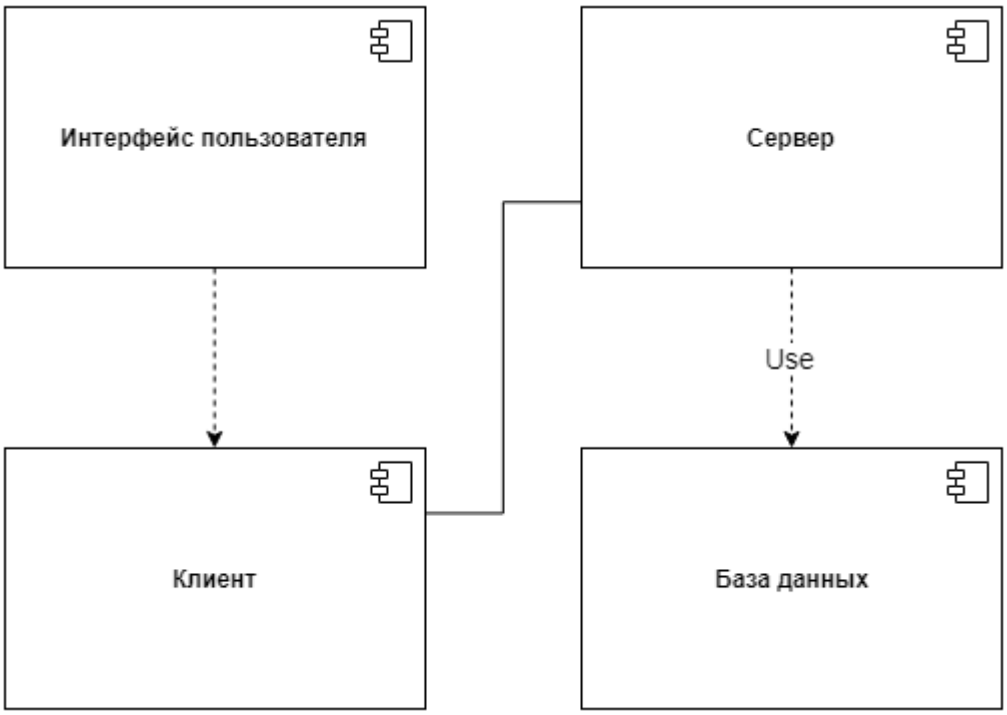

Рис. 11 Диаграмма компонентов

3.2. Примеры исходного кода

3.2.1. Frontend с использованием фреймворков Svelte и Sapper

Svelte – это фреймворк, который позволяет создать компонентную структуру сайта и использовать реактивность, т.е. обрабатывать изменения переменных и автоматически отображать обновленные данные на сайте.

Sapper – это расширение Svelte, вносящее некоторые изменения в организацию проекта, позволяющее быстро организовать роутинг (URL-дерево сайта, по которому он будет отвечать на запросы), динамические URL и создать независимые компоненты.

Для создания веб-сайта с использованием указанных выше фреймворков, необходимо дополнительно установить и настроить сборщик – это Node.js-модуль, который будет преобразовывать написанный мной код в код, понятный браузеру.

Я использовал сборщик rollup с настройками, изображенными на рисунке 12.

```
export default {
    client: {
        input: config.client.input(),
        output: config.client.output(),
        plugins: [
            replace({
                'process.browser': true,
                'process.env.NODE_ENV': JSON.stringify(mode)
            }),
            svelte({
                dev,
                hydratable: true,
                emitCss: true,
                preprocess
            }),
            resolve({
                browser: true,
                dedupe: ['svelte']
            }),
            commonjs(),

            legacy && babel({
                extensions: ['.js', '.mjs', '.html', '.svelte'],
                runtimeHelpers: true,
                exclude: ['node_modules/@babel/**'],
                presets: [
                    ['@babel/preset-env', {
                        targets: '> 0.25%, not dead'
                    }]
                ],
                plugins: [
                    '@babel/plugin-syntax-dynamic-import',
                    ['@babel/plugin-transform-runtime', {
                        useESModules: true
                    }]
                ]
            }),

            !dev && terser({
                module: true
            })
        ],

        onwarn,
    },
```

Рис. 12 Настройки Rollup

С помощью sapper, для организации роутов мне было достаточно расположить компоненты в папках в желаемом порядке. В моем случае порядок представлен на рисунке 13.

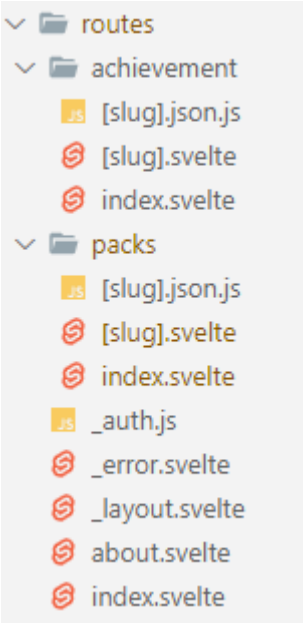

Рис. 13 Структура папки routes

Каждый *.svelte файл представляет собой код, который будет отображаться на странице. _error.svelte и _layout.svelte – технические файлы. Первый отображается, если человек попытался перейти по неправильной ссылке, второй отображается на всех страницах сайта. Таким образом, например, когда клиент переходит по адресу http://*/packs, за отображение страницы отвечает файл /routes/packs/index.js и т.д.

На некоторых страницах было необходимо отображать множество элементов со сложной версткой или программной логикой. Такие элементы были вынесены в отдельные компоненты, которые подключались в файлам в папке routes и выводились как стандартные HTML-теги. Примеры изображены на рисунках 14 и 15.

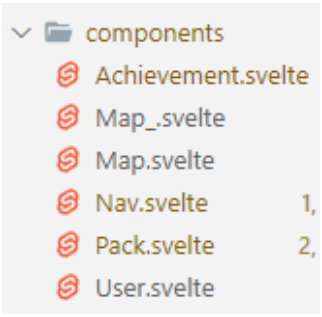

Рис. 14 Папка компонентов

```
<script context="module">
    import config from './../../config.js';
    import Pack from './../../components/Pack.svelte';
<script>

{#each packs as pack_data}
    <Pack data={pack_data}/>
{/each}
```

Рис. 15 Пример вызова компонента

3.2.2. Backend с использованием Node.js, JavaScript в формате REST API

Backend был реализован в качестве отдельного сервера, который может быть запущен на другом устройстве. Он отвечает за работу с Базой данных и отправляет данные в формате JSON при обращении.

Для работы с БД был установлен Node.js-модуль Sequelize, созданы и подключены модели таблиц.

Sequelize – это один из наиболее популярных модулей ORM для Node.js, позволяющих работать с MySQL на языке JavaScript.

Для начала работы были созданы модели таблиц в БД, пример представлен на рисунке 16. Модели были подключены к основному коду, пример запроса к БД представлен на рисунке 17

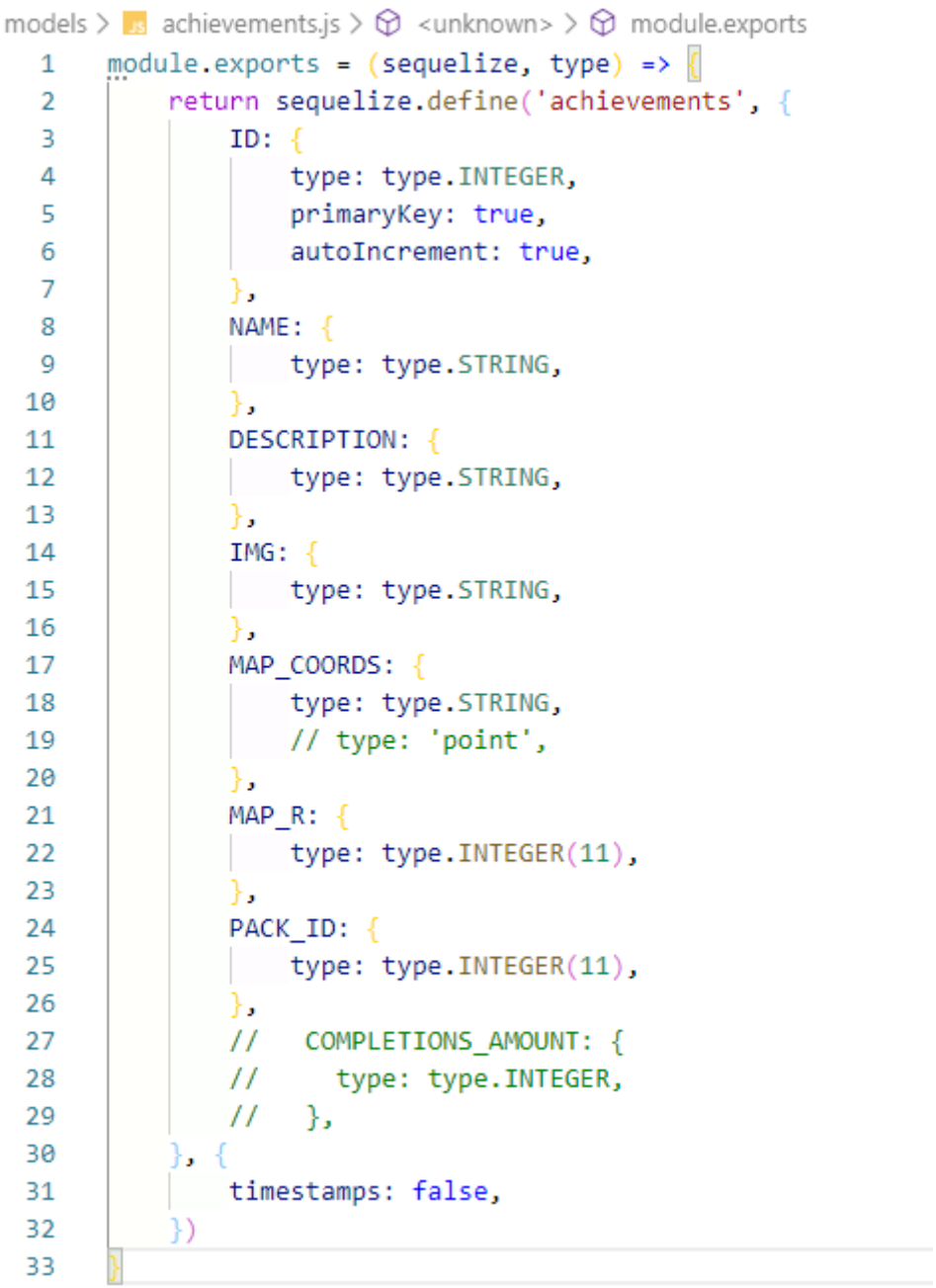

Рис. 16 Модель таблицы achievements

```
module.exports = function (app) {
    const
        config = app.config,
        achievements = app.middleware.db.models.achievements;

    this.get = function (req, res, next) {
        const {
            id
        } = req.params;

        if (isEmpty(id)) {
            return res.status(400).send({
                status: false
            });
        }

        achievements.findOne({
            where: {
                id: id
            }
        }).then((result) => {
            if (result != null) {
                const
                    data = result.dataValues;

                return res.json(
                    data
                )
            } else {
                return res.status(400).send({
                    status: false
                });
            }
        }).catch((error) => {
            return res.status(400).send({
                status: false
            });
        })
    }
```

Рис. 17 Пример запроса к базе данных

Для доступа к API были прописаны маршруты (роуты), к каждому из которых привязывалась функция, отвечающая за конкретный запрос. Пример изображен на рисунке 18.

```
module.exports = function(app) {
    const
        complete = app.controllers.achievements.complete,
        get = app.controllers.achievements.get,
        getByPack = app.controllers.achievements.getByPack;

    app.route('/api/achievements/complete')
        .post(complete);

    app.route('/api/achievements/pack/:id')
        .get(getByPack);

    app.route('/api/achievements/:id')
        .get(get);
};
```

Рис. 18 Маршруты API связанные с Achievements

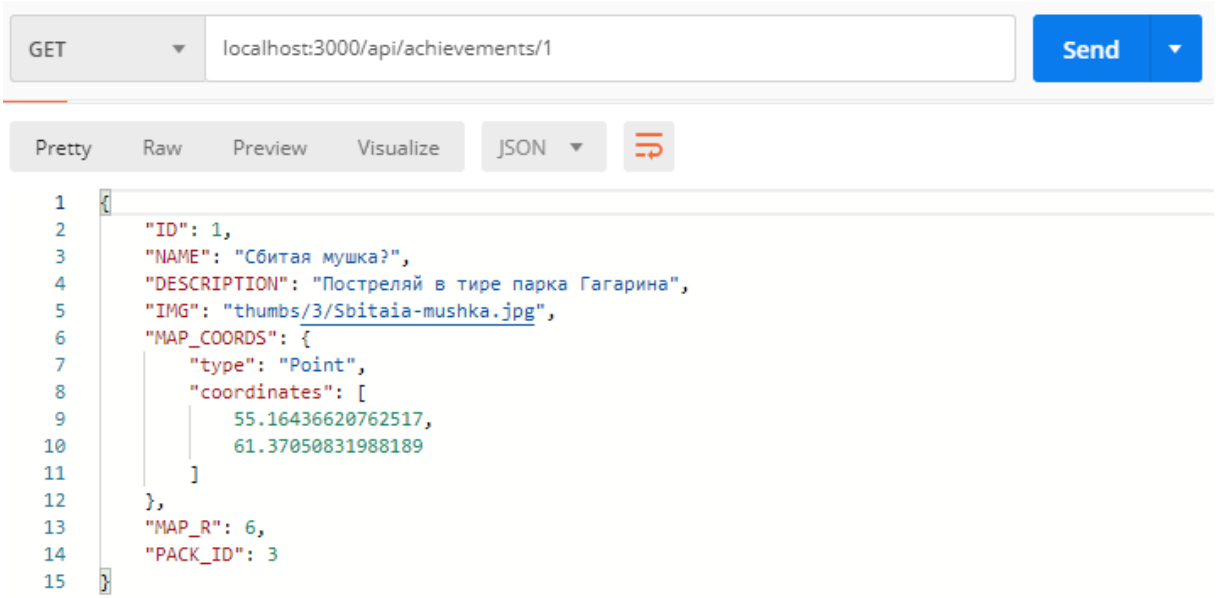

Рис. 19 Пример ответа на запрос к API

3.2.3. Авторизация с JsonWebToken

Авторизация с JWT – одна из наиболее сложных задач в рамках данной практики, потому что не имеет единственной общепринятой реализацией. Программисты до сих пор используют разные способы передачи, хранения и обновления JWT.

В своей реализации я храню JWT в куки и отправляю на backend в качестве header'а "Authorization" и в теле некоторых запросов. На клиентской части, в фоне каждую секунду идет проверка куки. Если JWT скоро сгорит – отправляется запрос на его обновление, пример на рисунке 20. Проверка идет только пока пользователь находится на сайте, поэтому если пользователь надолго покинет сайт – его сессия сгорит, что полностью удовлетворяет желаемой логике работы.

Для дополнительной безопасности используется 2 JWT – короткий (существует несколько минут) и долгосрочный (существует несколько дней). Они оба обновляются по таймеру краткосрочного, но долгосрочный передается только для обновления, а краткосрочный фигурирует в запросах. Таким образом, человек может покинуть сайт на несколько дней, не прервав сессию, но если его JWT перехватят при каком-то запросе, он будет актуален лишь несколько минут.

В листинге 1 представлен код обработки запроса на обновление JWT на стороне Backend.

Листинг. 1 Обработка запроса на обновление JWT на Backend

```
        token_list.findOne({
            where: {
                refresh_token: refresh_token
            }
        }).then((result) => {
            if(result != null) {
                const
                    data = result.dataValues,
                    new_refresh_token = jwt.sign(
                        { username: data.username },
                        config.jwt.refreshTokenSecret,
                        { expiresIn: config.jwt.refreshTokenExpiration }
                    ),
                    token = jwt.sign(
                        { username: data.username },
                        config.jwt.tokenSecret,
                        { expiresIn: config.jwt.tokenExpiration }
                    );
                token_list.update(
                    {
                        token: token,
                        refresh_token: new_refresh_token
                    },
                    {
                        where: {
                            refresh_token: refresh_token
                        }
                    }
                ).then((result) => {
                    if(result) {
                        return res.json({
                            token: token,
                            refresh_token: new_refresh_token,
                            status: true
                        })
                    } else {
                        return res.status(400).send({
                            status: false
                        });
                    }
                })
            }
        })
```

```
export let time = readable(null, function start(set) {

    const interval = setInterval(() => {
        let secondsLeft;
        if(process.browser) {
            //если мы в браузере, то пытаемся найти оставшееся время в куки
            let exp_dateVal = parseInt(document.cookie.replace(/.+exp_date=(\d+)/gi, "$1"));
            secondsLeft = Math.floor((exp_dateVal - (new Date()).getTime()) / 1000);
            // console.log(expTime);
            if(secondsLeft && secondsLeft <= 1) {
                //обновляем токен, если он сгорел
                console.log("updating token");
                let refresh_token = getCookie("refresh_token");
                apiClient(`${config.apiURL}/authentication/refreshToken`, { data: refresh_token })
                    .then(res => {
                        document.cookie = "token="+res.data.token;
                        document.cookie = "refresh_token="+res.data.refresh_token;
                        let exp_date = (new Date()).getTime() + 1000 * 900; //900s
                        document.cookie = "exp_date=" + exp_date;
                    });
            }
        }
        set(secondsLeft || null);
    }, 1000);

    return function stop() {
        clearInterval(interval);
    };

});
```

Рис. 20 Проверка и автоматическое обновление JWT

3.2.4. Динамические URL

Так как количество контента на сайте не ограничено и должно постоянно увеличиваться, было необходимо создать механизм динамического построения URL, чтобы можно было получить ссылку на любые Достижение или Набор.

Для решения данной задачи использовались средства Sapper. Как было указано ранее, в routes хранятся файлы, которые отвечают за отображение страниц, на которые перешел пользователь. Sapper позволяет использовать функционал slug для задания компонентов, которые будут отвечать за отображение страниц по заранее неизвестным адресам.

В частности, в данном проекте таких ссылок 2:

/packs/*id набора* и /achievement/*id достижения*

Чтобы пользователь мог получить информацию по конкретному Набору или Достижению, в папках packs и achievements были созданы файлы [slug].svelte

Пример реализации компонента для страницы Набора достижения представлен на рисунках 21 и 22.

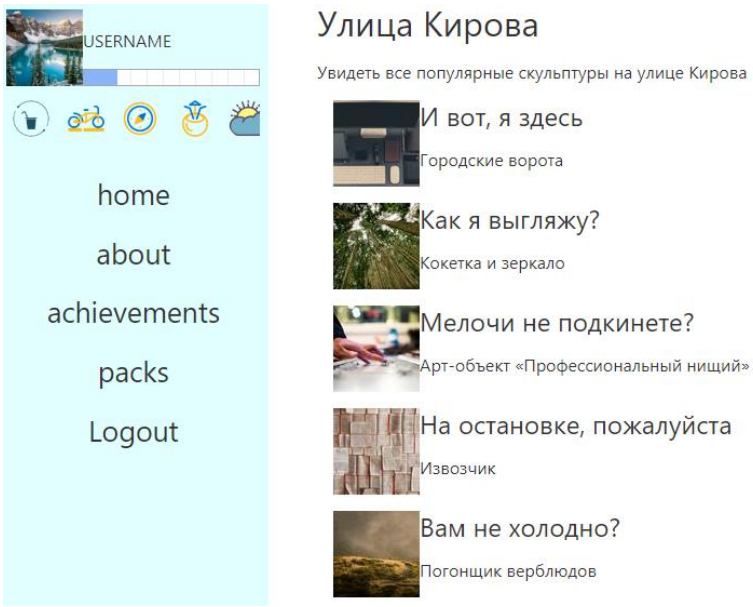

Рис. 21 Результат работы /packs/[slug].svelte по адресу http://*/packs/1

src > routes > packs > [slug].svelte > ...

```svelte
<script context="module">
    import config from '../../../config.js';
    import Achievement from '../../../components/Achievement.svelte';

    export async function preload({ params, query }) {

        // the `slug` parameter is available because
        // this file is called [slug].svelte
        const res = await this.fetch(`${config.restURL}/api/packs/${params.slug}`);
        const pack = await res.json();

        if (res.status === 200) {
            return { pack };
        } else {
            this.error(res.status, data.message);
        }
    }
</script>

<script>
    export let pack;
</script>

<style lang="scss">
    @import "./../style/pack.scss";
</style>

<svelte:head>
    <title>{pack.NAME}</title>
</svelte:head>
<h1 class="pack__title">{pack.NAME}</h1>
<p class="pack__description">{pack.DESCRIPTION}</p>
{#each pack.achievements as achi_data}
    <Achievement data={achi_data}/>
{/each}
```

Рис. 22 Код страницы Набора

3.2.5. Адаптивная верстка с использованием SCSS

Сразу хотелось бы отметить, что отсутствие дизайна не мешает созданию адаптивной верстки, которая при введении дизайна практически не будет нуждаться в корректировках. В рамках этой задачи нужно организовать позиционирование компонентов таким образом, чтобы сайт отображался ровно на подавляющем большинстве экранов, включая мобильные. То есть все элементы должны помещаться на экране по ширине и не перекрывать друг друга.

SCSS имеет множество преимуществ по сравнению с CSS. Хоть в конечном счете он и преобразуется в обычный CSS, на этапе разработки синтаксис SCSS намного более компактен и читаем. Основные преимущества

перед CSS: возможность вложенности, упрощенная система использования переменных, mixins (часто-используемые блоки стилей, которые можно написать один раз, а потом подключать ко всем желаемым элементам 1 строкой).

Пример синтаксиса изображен на рисунке 23.

Примеры верстки на рисунках 24-26.

src > style > component > achievement.scss > @keyframes stamp

```scss
$achi-height: 80px;

.achievement {
    display: flex;
    position: relative;
    width: 80%;
    height: $achi-height;

    img {
        position: relative;
        width: $achi-height;
        object-fit: cover;
        pointer-events: none;
    }

    &__content {
        position: relative;

        h2 {
            width: 100%;
            white-space: nowrap;
        }
        p {

        }
    }

    &.completed:after {
        display: flex;
        position: absolute;
        content: "COMPLETED";
        border: 2px dotted red;
        border-radius: 10pt;
        right: 5%;
        font-size: 20pt;
        font-weight: 900;
        padding: 3pt 6pt;
        animation: 0.3s stamp forwards;
    }
}
```

Рис. 23 Пример синтаксиса SCSS

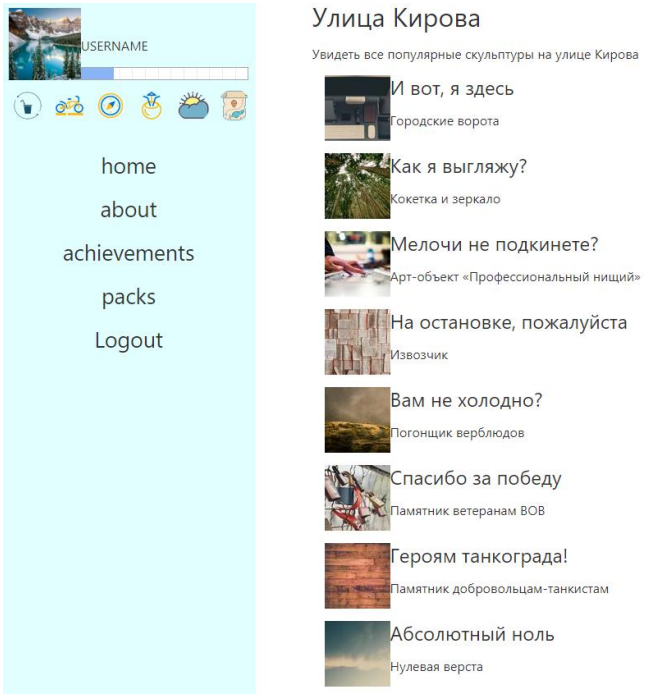

Рис. 24 Сайт на экране ПК

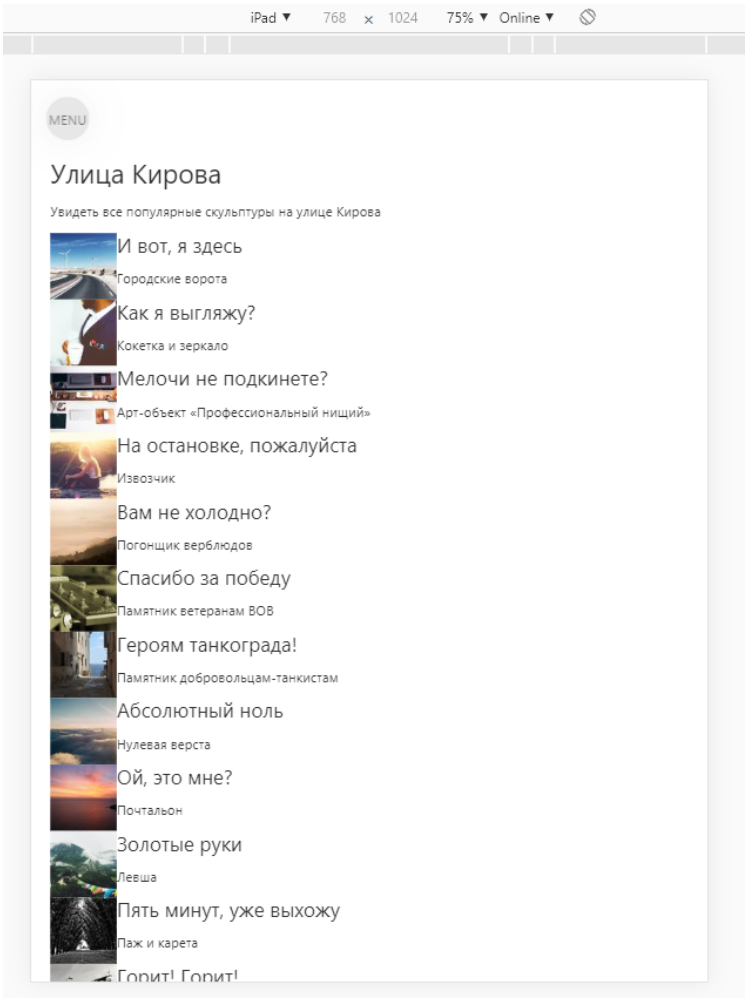

Рис. 25 Сайт на экране iPad

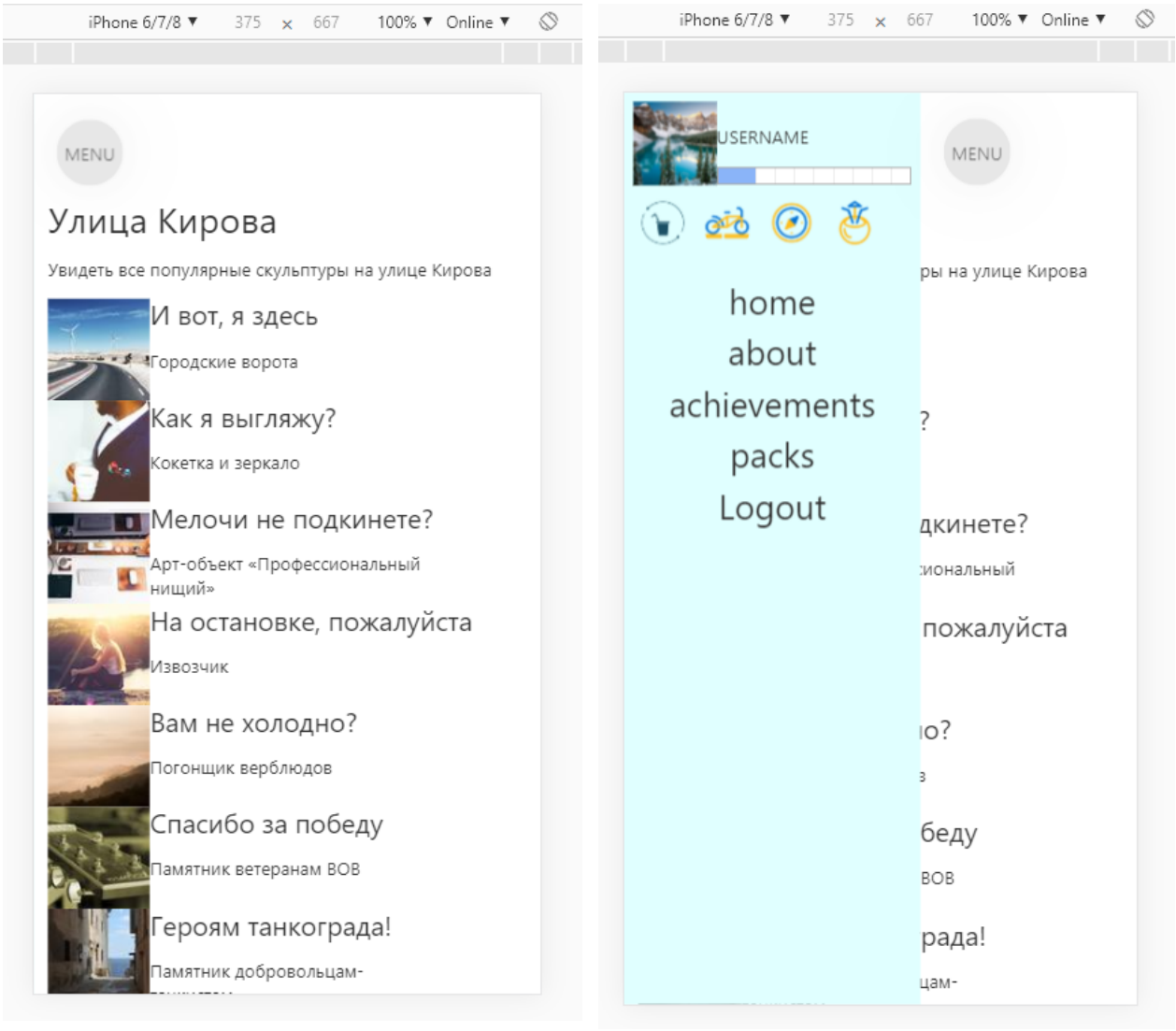

Рис. 26 Сайт на экране iPhone с открытым и закрытым меню

3.2.6. Определение геолокации пользователя

Для определения точной геолокации необходимо подключаться к сайту именно по протоколу https. Для этого был создан и подключен самоподписанный сертификат. Код подключения сертификата и запрос переадресации изображены на рисунке 27. Определение геолокации на рисунке 28.

```
let
    credentials = {
        key: fs.readFileSync('./my.key', 'utf8'),
        cert: fs.readFileSync('./my.crt', 'utf8')
    };


app.use((req, res, next) => {
    if(!req.secure)
        res.redirect('https://' + req.headers.host.replace(/:(\w)+/gi,":"+HTTPS_PORT) + req.url);

    next();
})

var httpServer = http.createServer(app);
var httpsServer = https.createServer(credentials, app);

httpServer.listen(HTTP_PORT);
httpsServer.listen(HTTPS_PORT);
```

Рис. 27 Код подключения сертификата

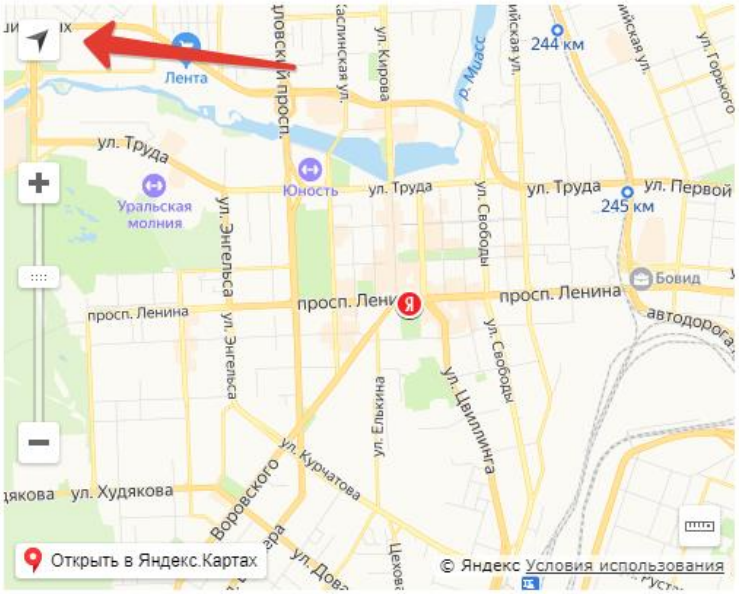

Рис. 28 Геолокация пользователя

4. ТЕСТИРОВАНИЕ

4.1. Функциональное тестирование

Тестирование – это процесс работы системы или компонента, при определенных тестировщиком условиях, для наблюдения и для оценки некоторых аспектов работы системы либо компонента. В процессе тестирования производится анализ ПО для выявления несоответствия между существующими особенностями выполнения ПО и требованиями к ПО. Цель тестирования состоит не в том, чтобы показать удовлетворительную работу программы, а в том, чтобы четко определить, в чем работа программы неудовлетворительна.

Тестирование веб-приложения производилось методом функционального тестирования.

Функциональное тестирование – это тестирование ПО в целях проверки реализуемости функциональных требований, то есть способности ПО в определенных условиях решать задачи, нужные пользователям.

Результаты функционального тестирования представлены в таблице 1.

Табл. 1. Результаты функционального тестирования.

№ п/п	Тестируемая функция	Ожидаемый результат	Полученный результат	Вывод
1.	Клик по пункту меню кроме “Выйти”	Переход на соответствующую страницу	Переход на соответствующую страницу	Тест пройден
2.	Клик по пункту меню “Выйти”	Выход пользователя, уничтожение куки	Выход пользователя, уничтожение куки	Тест пройден
3.	Клик по достижению на карте	Появление всплывающей подсказки с ссылкой на страницу достижения	Появление всплывающей подсказки с ссылкой на страницу достижения	Тест пройден
4.	Клик по паку достижений	Появление списка включенных достижений	Появление списка включенных достижений	Тест пройден

Окончание таблицы 1

№ п/п	Тестируемая функция	Ожидаемый результат	Полученный результат	Вывод
5.	Клик по достижению	Открывается страница достижения	Открывается страница достижения	Тест пройден
6.	Клик по уменьшенному изображению на странице достижения	Открытие выбранного изображения	Открытие выбранного изображения	Тест пройден
7.	Клик по кнопке “Подписаться” под паком достижений	После подтверждения добавляется соответствующая запись в таблице LINK_USER_PACK, вместо кнопки появляется надпись “Вы уже подписаны”, счетчик подписчиков увеличивается	После подтверждения добавляется соответствующая запись в таблице LINK_USER_PACK, вместо кнопки появляется надпись “Вы уже подписаны”, счетчик подписчиков увеличивается	Тест пройден
8.	Клик по кнопке “Выполнить” на странице достижения	После подтверждения добавляется соответствующая запись в таблице LINK_USER_ACHIEVEMENT, текст кнопки меняется на “Выполнено”, кнопка становится недоступной	После подтверждения добавляется соответствующая запись в таблице LINK_USER_ACHIEVEMENT, текст кнопки меняется на “Выполнено”, кнопка становится недоступной	Тест пройден

4.2. Тестирование интерфейса

Результаты тестирования интерфейса приведены в таблице 2.

Табл. 2. Результаты тестирования интерфейса.

№ п/п	Тестируемый элемент	Ожидаемый результат	Полученный результат	Вывод
1.	Достижения на карте	Неавторизованный пользователь видит все, авторизированный только доступные для выполнения.	Неавторизованный пользователь видит все, авторизированный только доступные для выполнения.	Тест пройден
2.	Кнопка около количества подписчиков пака достижений	Неавторизованный пользователь не видит, авторизованный либо видит кнопку, либо надпись «Вы уже подписаны»	Неавторизованный пользователь не видит, авторизованный либо видит кнопку, либо надпись «Вы уже подписаны»	Тест пройден
3.	Кнопка выполнения достижения на странице достижения	Неавторизованный пользователь не видит, авторизованный либо видит доступную кнопку «Выполнить», либо недоступную кнопку «Выполнено»	Неавторизованный пользователь не видит, авторизованный либо видит доступную кнопку «Выполнить», либо недоступную кнопку «Выполнено»	Тест пройден
4.	Блок над меню	Неавторизованный пользователь не видит форму авторизации, авторизованный – свои данные	Неавторизованный пользователь не видит форму авторизации, авторизованный – свои данные	Тест пройден
5.	Блок меню	На мобильных устройствах свернут и открывается по кнопке “Menu”. На больших экранах показывается постоянно	На мобильных устройствах свернут и открывается по кнопке “Menu”. На больших экранах показывается постоянно	Тест пройден

ЗАКЛЮЧЕНИЕ

В настоящее время людям приходится перебирать разные источники, чтобы подобрать список интересных мест города и найти информацию о них. Кроме того, чтобы хранить информацию о посещенных местах, необходимо использовать дополнительные ресурсы.

В рамках работы было разработано веб-приложение для поиска и отслеживания посещения примечательных мест города Челябинска. Приложение позволяет расширять базу мест для любых других городов.

Для достижения данной цели были решены следующие задачи:

- разработать Frontend с использованием фреймворков Svelte и Sapper;
- разработать Backend с использованием Node.js, JavaScript в формате REST API;
- реализовать авторизацию с JsonWebToken;
- реализовать динамические URL;
- реализовать адаптивная верстку с использованием SCSS;
- добавить возможность определения геолокации пользователя.

ЛИТЕРАТУРА

1. Пользовательский интерфейс. [Электронный ресурс] URL: https://spravochnick.ru/informatika/arhitektura_personalnogo_kompyutera/polzovatelskiy_interfeys/ (дата обращения: 02.05.2020).

2. Крэг Ларман. Применение UML 2.0 и шаблонов проектирования. – 3-е изд. – Москва. «Вильямс», 2006. – 736 с.

3. Руководство пользователя для работы с Open Server Panel. [Электронный ресурс] URL: https://ospanel.io/docs (дата обращения: 07.05.2020).

4. Документация MySQL. [Электронный ресурс] URL: http://www.mysql.ru/docs (дата обращения: 10.05.2020).

5. Справочник по HTML. [Электронный ресурс] URL: http://htmlbook.ru (дата обращения: 12.05.2020).

6. Документация по Svelte [Электронный ресурс] URL: https://svelte.dev/docs (дата обращения: 12.05.2020).

7. Документация по Sapper [Электронный ресурс] URL: https://sapper.svelte.dev/docs/ (дата обращения: 15.05.2020).

8. Документация по Express [Электронный ресурс] URL: https://expressjs.com/ru/api.html (дата обращения: 18.05.2020).

9. Работа с куки в JavaScript [Электронный ресурс] URL: https://learn.javascript.ru/cookie (дата обращения: 20.05.2020).

10. Token, refresh token и создание асинхронной обертки для REST-запроса [Электронный ресурс] URL: https://habr.com/ru/post/456188/ (дата обращения: 20.05.2020).

11. Nodejs Authentication Using JWT and Refresh Token [Электронный ресурс] URL: https://codeforgeek.com/refresh-token-jwt-nodejs-authentication (дата обращения: 01.06.2020).

12. Документация Yandex.API [Электронный ресурс] URL: https://tech.yandex.ru/maps/jsapi/doc/2.1/quick-start/index-docpage/ (дата обращения: 02.06.2020).

13. Калбертсон Роберт, Браун Крис, Кобб Гэри. Быстрое тестирование. – Москва. «Вильямс», 2002. – 374 с.

14. Документация по Sequelize [Электронный ресурс] URL: https://sequelize.org/v5/.

15. Дэвид Флэнаган. JavaScript. Подробное руководство – Москва. «Символ-Плюс», 2013. – 1080 с.