Министерство науки и высшего образования Российской Федерации

ФБГОУ ВО «Кубанский государственный технологический университет»

(ФГБОУ ВО «КубГТУ»)

Институт компьютерных систем и информационной безопасности

Кафедра информационных систем и программирования

Направление подготовки 09.03.04 программная инженерия

Профиль проектирование и разработка программного обеспечения

КУРСОВОЙ ПРОЕКТ

по дисциплине «Разработка веб-приложений»

на тему: «Разработка приложения для управления коллекциями данных»

Выполнил студент Заболотнов Дмитрий Алексеевич

курса 3 группы 21-ЗКБ-ПР1

Допущен к защите _____

Руководитель (нормоконтролёр) проекта _____ Шумков Е.А.

Защищен _____ Оценка _____

Члены комиссии _____ ст. преподаватель Кушнир Н.В.

_____ ст. преподаватель Волик А.Г.

_____

_____

Краснодар

2024 г.

ФБГОУ ВО «Кубанский государственный технологический университет»
(ФГБОУ ВО «КубГТУ»)

Институт компьютерных систем и информационной безопасности
Кафедра информационных систем и программирования
Направление подготовки 09.03.04 программная инженерия
Профиль проектирование и разработка программного обеспечения

УТВЕРЖДАЮ
Зав. кафедрой ИСП
доц. М.В. Янаева
_____ 2024 г.

ЗАДАНИЕ
на курсовой проект

Студенту Заболотнову Дмитрию Алексеевичу
курса 3 группы 21-ЗКБ-ПР1
Тема проекта: Разработка приложения для управления коллекциями данных
(утверждена указанием директора института № ___ (от _____)
План проекта:
1. Описание предметной области
2. Проектирование
3. Реализация

Объем проекта:
А) пояснительная записка 54 с.
Б) Программа
В) Приложения
Рекомендуемая литература:
1. Прохорёнок Н.А., Дронов В.А. HTML, JavaScript, PHP и MySQL. – 4-е изд. – СПб.: БХВ-Петербург, 2015. – 768 с. /
2. Нараян Прасти. Введение в ECMAScript 6. – Пер. с англ. – М.: ДМК Пресс, 2016. – 176 с. /
3. Дунаев В.В. HTML, скрипты и стили. – 4-е изд. – СПб.: БХВПетербург, 2015. – 816 с. /

Срок выполнения работы: с "____" ____ по "____" ____ 2024 г.
Срок защиты: "____" ____ 2024 г.
Дата выдачи задания: "____" ____ 2024 г.
Дата сдачи проекта на кафедру: "____" ____ 2024 г.

Руководитель проекта _____ Шумков Е.А.
Задание принял студент _____ Заболотнов Д. А.

Реферат

Курсовая работа содержит — 54 страницы, 10 рисунков, 28 вставок листинга, 1 таблицу, 3 схемы, 15 источников.

ВЕБ-СЕРВИСЫ, САЙТЫ, WEB-ПРОГРАММИРОВАНИЕ, СЕРВЕР, ПРОЕКТИРОВАНИЕ, ДИЗАЙН, ТЕСТИРОВАНИЕ, БАЗЫ ДАННЫХ, FIGMA, JAVASCRIPT, REACT/NEXTJS, JEST, REACT TESTING LIBRARY PHP, MYSQL, PHPMYADMIN, NGINX.

Цель курсового проекта состоит в изучении технологий разработки и реализации веб-приложений и получении практических навыков в этой области знаний на конкретном примере. Приложение для управления коллекциями данных реализовано в виде веб-сервиса для просмотра достопримечательности Краснодарского края. В рамках курсового проекта были рассмотрены различные подходы к разработке веб-приложения, а также изучены имеющиеся на сегодня инструменты и методы его реализации.

При выполнении курсового проекта были закреплены основы и углублённые знания в технологиях разработки веб-приложений и сайтов в целом через практическую реализацию на индивидуальном примере. При работе над курсовым проектом были самостоятельно выполнены все этапы создания продукта, от постановки задачи до практической реализации, заканчивающейся верификацией и подготовкой отчетности. В процессе выполнения работы были получены навыки самостоятельного использования специальной литературы, справочников, стандартов.

Для реализации этой работы были задействованы следующие языки и инструменты: JAVASCRIPT с использованием NEXT.js и React, PHP и Composer, MySql и PHPmyAdmin, веб-сервер/прокси-сервер NGINX и Node.js, Microsoft Visio и NPM. Произведен анализ предметной области, разработана структура сайта и дизайн. Описана внутренняя структура движка сайта как для клиентской, так и для серверной части. Была написана тестовая база данных.

Содержание

Введение

Веб-приложение — приложение, с которым клиент взаимодействует через браузер. Основная особенность и отличие от обычного сайта — нет никакой перезагрузки страницы. Все запросы осуществляются в фоновом режиме по технологии AJAX (*Asynchronous Javascript and XML)*, это дает быстроту и удобство использование сайта.

Также в последнее время набирает большую популярность технология *WebSocket*, которая не требует постоянных запросов от клиента к серверу, а создает двунаправленное соединение, при котором сервер может отправлять данные клиенту без запроса от последнего. Таким образом появляется возможность динамически управлять контентом в режиме реального времени.

Для разработки дизайна и макета сайта выбран онлайн-сервис для разработки интерфейсов Figma, а также был задействован Photoshop и ряд других технологий. Для написания клиентской части веб-приложение, используется браузерный язык JavaScipt в сочетании с фреймворком NEXT.js в основе которого лежит библиотека React. За работу сервера и обработку входящих запросов будет отвечать серверный язык программирования PHP вместе с библиотекой Composer, так же запуск всего проекта будет осуществлять веб-сервер/прокси-сервер NGINX и Node.js в качестве “клиент-сервера”. Для разработки базы данных выбран язык запросов Sql, его диалект MySql с применением приложения для администрирования баз данных PHPMySql. Среда разработки: редактор исходного кода Microsoft Visio с менеджером пакетов NPM.

Задачи:

1. Смоделировать макет сайта, его структуру;
2. Дизайн сайта;
3. Клиентская часть;
4. Серверная часть и база данных;
5. Различные тесты готового продукта.

1 Спецификации задачи

Приложение для управление коллекциями базы данных должно иметь удобный интерфейс для просмотра этих данных. Это приложение написано в виде веб-сервиса. Веб-сервис — интернет программа с общим доступом, позволяющая пользователям онлайн просматривать различные услуги/товары/новости и другой контент с различными дополнительными возможностями, к примеру добавлять в избранное, вести поиск, расчеты и т.п..

Основная цель этого интернет-ресурса — демонстрация достопримечательностей Краснодарского Края, их описание и сбор оценок.

Разрабатываемая программа должна обеспечить возможность выполнения перечисленных ниже функций:

1. Удобный просмотр каталога;
2. Просмотр каталога с выбранными фильтрами и сортировками;
3. Предоставлять пользователю возможность оценивать достопримечательность;
4. Предоставлять пользователю добавлять достопримечательность в избранное;
5. Предоставлять пользователю возможность вести поиск достопримечательностей;
6. Программа должна корректно отображать все данные, которые были заложены в базу данных;
7. Каждый пользователь может пройти голосование за лучшую достопримечательность;
8. У пользователя есть возможность отправить форму с обратной связью или подписаться на обновления;
9. Пользователь может просматривать раздел новости;
10. Веб-приложение должно вести учет просмотров каждой веб-страницы.

2 Описание среды разработки

Для написание современного веб-приложения не обойтись без одной из JavaScript библиотеки или фреймворка для разработки пользовательских интерфейсов. Выделю несколько популярных из них на данный момент:

1. React — библиотека с открытым исходным кодом, от компании *Meta* (деятельность компании запрещена на территории Российской Федерации), представила она его в 2013 г.. Библиотека используется для разработки одностраничных сайтов, многостраничных, веб-приложений и мобильных приложений. На этой библиотеке работают самые популярные соц. сети мира.
2. Vue.js — JavaScript фреймворк появившийся в 2014 г. Как личный сайд-проект от Эвана Ю, на сегодняшний день был представлен Vue 3 который набирает все больше популярности и по моему мнению это второе место в списке инструментов для разработки веб-приложений. Разработчики заявляют, что в стресс-тестах он обходит React и Angular.
3. Angular — JavaScript фреймворк появившийся в 2016 г. От компании Google. В 2010 году появился Angularjs, но потом его полностью переписали в новую универсальную платформу Angular.

Приложение будет работать на библиотеке React. Помимо одной библиотеки нужен еще фреймворк, с помощью которого производится маршрутизация, а также возможность использовать SSR. Для React это NEXT.js.

Для обработки запросов с клиента на сервере используется язык PHP, это самый популярный серверный язык. В качестве СУБД выбрана MySQL вместе с phpMyAdmin для удобного управления данными.

Получается есть сервер Node.js для поддержки SSR и еще нужен сервер для обработки PHP скриптов, а также для запуска MySQL, для этого будет использоваться NGINX в качестве прокси сервера.

3 Анализ рынка в данной области. Сайты аналоги

В сети интернет есть много различных ресурсов, веб-сервисов для просмотра достопримечательностей, в качестве примера я выберу некоторые из них, отмечу плюсы и минусы и покажу что бы я сделал по-своему:

1. Tripadvisor — это сайт для просмотра и оценки мест, отелей, городов, достопримечательностей и т. д. Есть быстрые оценки мест, подробные отзывы, личный кабинет, добавление в избранное, так же удобное описание мест, удобный просмотр изображений места и много различного всего. Сайт сделан веб-приложением, что делает его достаточно быстрым и отзывчивым. Не могу отметить каких-то выраженных минусов, мой веб-сервис будет примерно похожим. Адрес: https://www.tripadvisor.ru/.

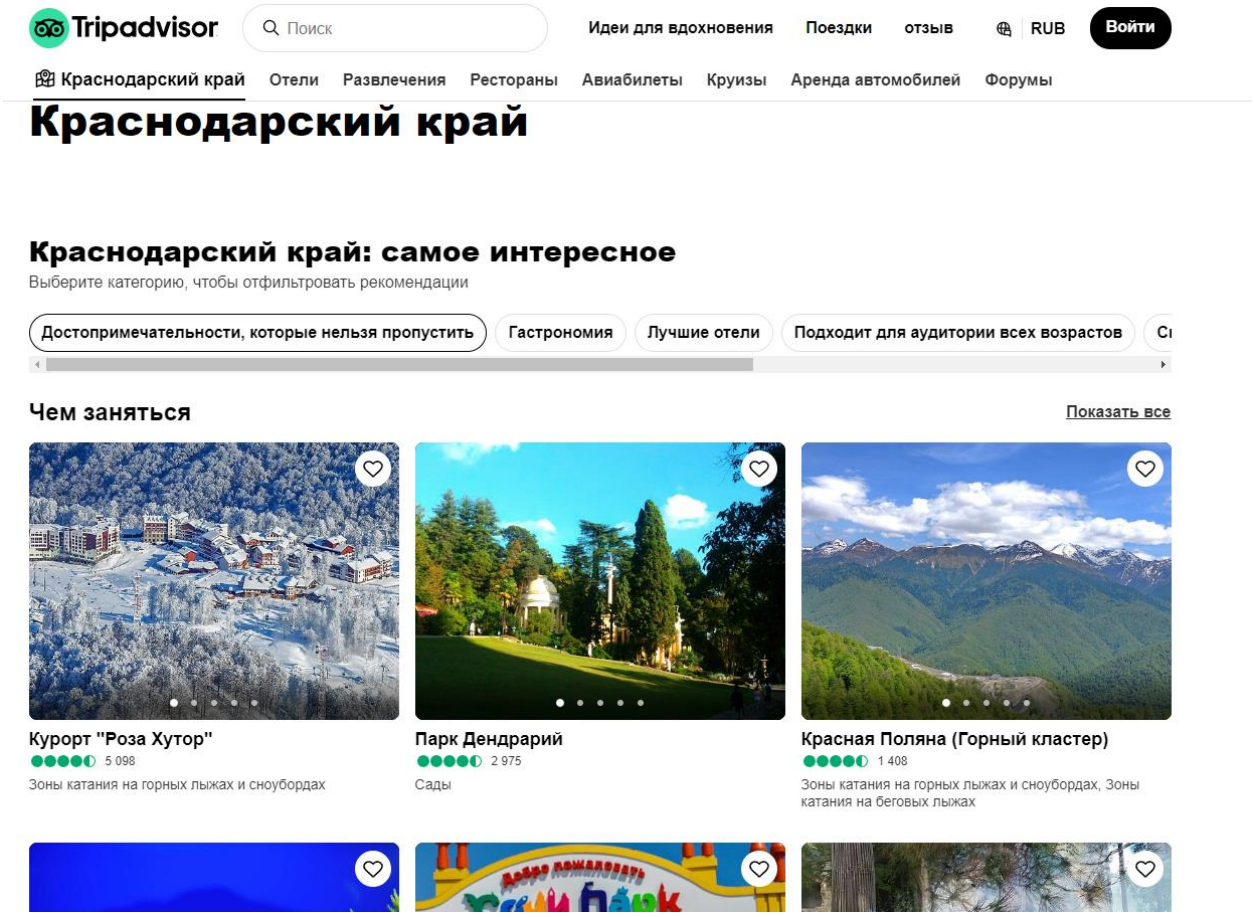


Рисунок 1 — Сайт аналог Tripadvisor

2. Кукарта.ру — сайт, немного другой по структуре, в нем нет реализации каталога, вся информация расписана в блогах и меню, это нельзя назвать минусом, но информации в нем умещается гораздо меньше. В нем есть

возможности поиска по сайту, так же удобный просмотр каждого места и блога, возможность оставлять комментарии. Из минусов можно назвать что сайт статический, мой же сайт будет реализован как веб-приложение. Адрес: https://kukarta.ru/.

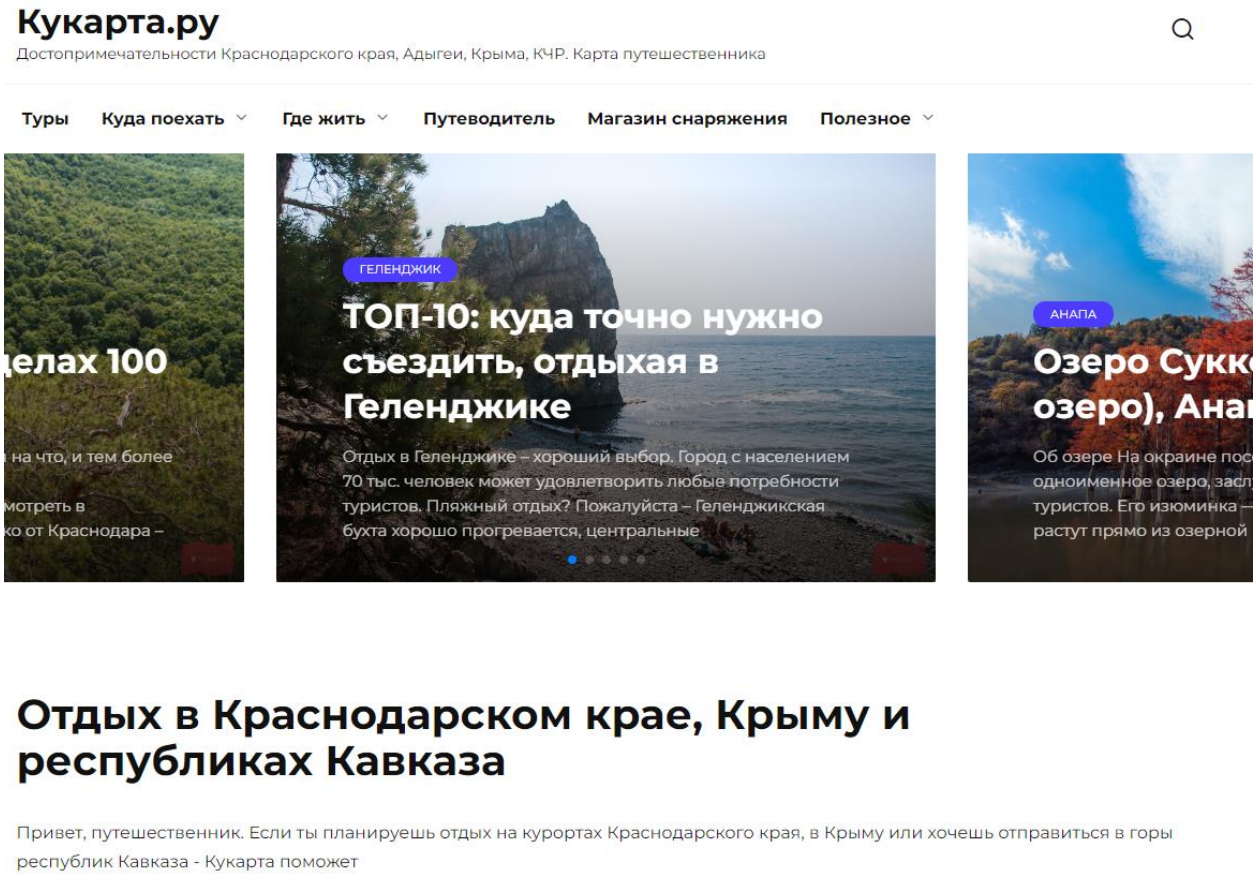


Рисунок 2 — Сайт аналог Kukarta

3. Экстрагид — сайт просмотра достопримечательностей и экскурсий по всему миру. Есть вполне удобный каталог всех мест, включая большое количество фильтры и сортировку. Из минусов — простой дизайн, сайт не работает как веб-приложение. Плюсы — много функционала и взаимодействия, такие как оставление отзывов, оценок, ссылки на бронирования поездок.

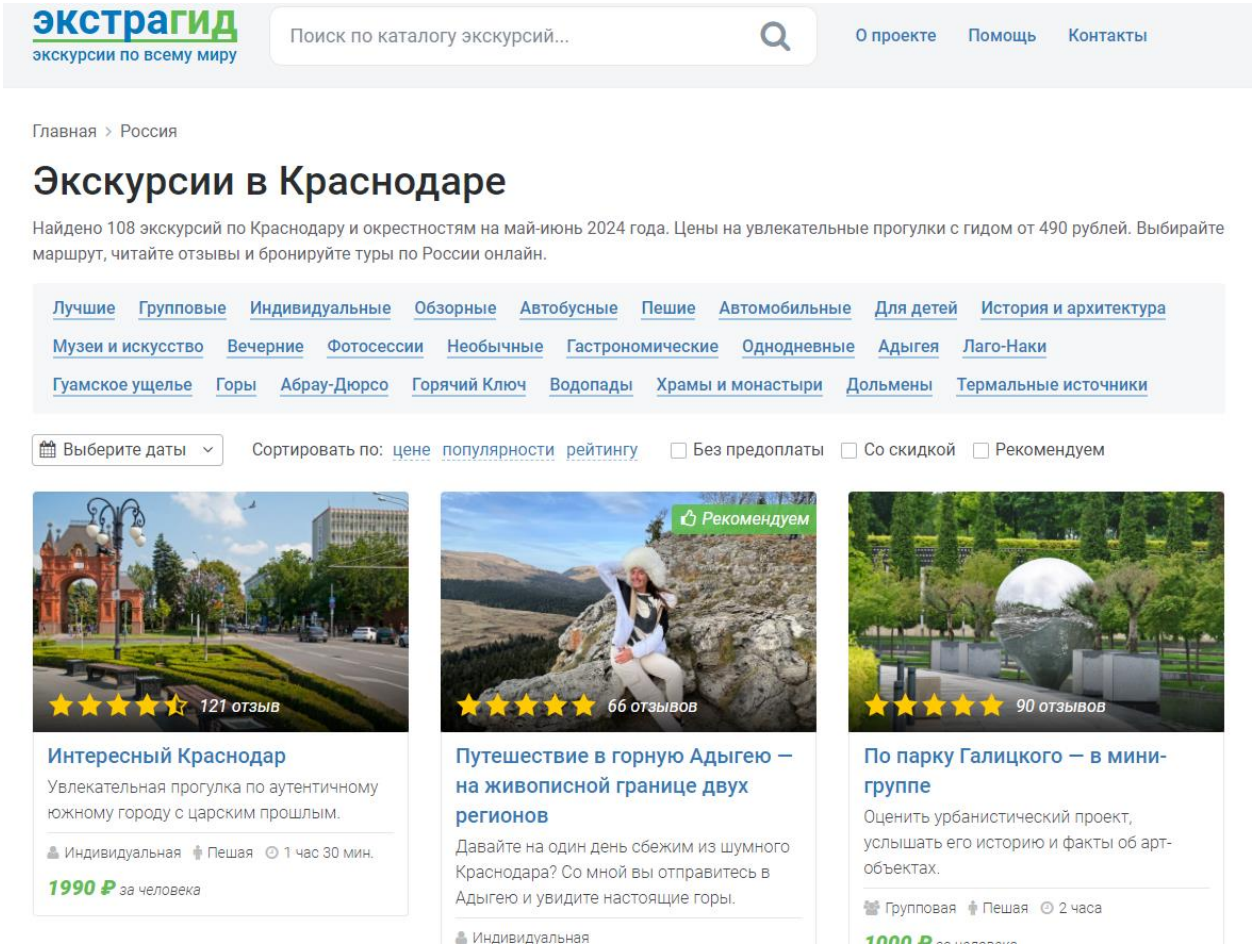


Рисунок 3 — Сайт аналог Экстрагид

4 Планирование проекта

Перед началом любой разработки нужно спланировать проект. С использованием программы Microsoft Project Document были отражены основные этапы разработки программного продукта.

Название задачи	Длительность	Начало	Окончание	Названия ресурсов	Трудозатраты
Веб-сервис	**14,88 дней**	**Вс 14.04.24**	**Вс 28.04.24**		**138 часов**
Этап 1	**2,88 дней**	**Вс 14.04.24**	**Вт 16.04.24**		**16 часов**
Разработка дизайна сайта	**2 дней**	**Вс 14.04.24**	**Пн 15.04.24**		**16 часов**
Дизайн всех страниц и изображений	2 дней	Вс 14.04.24	Пн 15.04.24	Заболотнов Д. А.	16 часов
Этап 2	**9 дней**	**Вт 16.04.24**	**Ср 24.04.24**		**98 часов**
Написание клиентской части	**5,88 дней**	**Вт 16.04.24**	**Вс 21.04.24**		**48 часов**
Разработка клиентской маршрутизации	1 день	Вт 16.04.24	Вт 16.04.24	Заболотнов Д. А.	8 часов
Статичная верстка всех страниц	2 дней	Ср 17.04.24	Чт 18.04.24	Заболотнов Д. А.	16 часов
Написание логики	**3 дней**	**Пт 19.04.24**	**Вс 21.04.24**		**24 часов**
Оживление активных элементов - слайдеры, подгрузки, фильтры, спинеры. Добавление оставшейся анимации	2 дней	Пт 19.04.24	Сб 20.04.24	Заболотнов Д. А.	16 часов
Написание запросов к серверу	1 день	Пт 19.04.24	Пт 19.04.24	Заболотнов Д. А.	8 часов
Разработка базы данных	**2 дней**	**Пт 19.04.24**	**Сб 20.04.24**		**22 часов**
Создание представление базы, проектирование таблиц и их данных	1 день	Пт 19.04.24	Пт 19.04.24	Заболотнов Д. А.	8 часов
Разработка базы данных - создание таблиц, связывание таблиц, прочие триггеры	1 день	Пт 19.04.24	Пт 19.04.24	Заболотнов Д. А.	8 часов
Заполнение базы данных	1 день	Сб 20.04.24	Сб 20.04.24		6 часов
Разработка серверной части	**4 дней**	**Вс 21.04.24**	**Ср 24.04.24**		**28 часов**
Написание серверной маршрутизация	1 день	Вс 21.04.24	Вс 21.04.24	Заболотнов Д. А.	8 часов
Написание логики обработки запросов с клиентской части	1 день	Пн 22.04.24	Пн 22.04.24	Заболотнов Д. А.	8 часов
Написание запросов к базе данных	1 день	Вт 23.04.24	Вт 23.04.24	Заболотнов Д. А.	8 часов
Серверная безопасность	0,5 дней	Ср 24.04.24	Ср 24.04.24	Заболотнов Д. А.	4 часов
Этап 3	**3 дней**	**Пт 26.04.24**	**Вс 28.04.24**		**24 часов**
Тестирование	**2,88 дней**	**Пт 26.04.24**	**Вс 28.04.24**		**24 часов**
Тестирование клиентской части	2 дней	Пт 26.04.24	Сб 27.04.24	Заболотнов Д. А.	16 часов
Тестирование серверной части	1 день	Вс 28.04.24	Вс 28.04.24	Заболотнов Д. А.	8 часов

Таблица 1 — Таблица общего плана проекта

5 Проектирование и дизайн веб-приложения

5.1 Построение структуры веб-приложения

Веб-сервис будет содержать следующие блоки (страницы):

1. Шапка и подвал сайта — статическая обертка, которая будет доступна на каждой странице для удобной навигации. Шапка будет содержать в себе следующие элементы: ссылки на просмотр каталога достопримечательностей, на страничку новостей, и на контакты; блок поиска достопримечательностей; блок просмотра избранных товаров. Подвал будет содержать в себе контактную информацию (почта, телефон, соц. сети), ссылку на политику конфиденциальности и быструю форму на подписку.

2. Главная страница — на ней представлен большой слайдер с некоторой информацией, далее блок с последними новостями, ниже блок с достопримечательностями которые главные в рейтинге, блок со случайно подобранными достопримечательностями, и форма для обратной.

3. Страничка контактов — содержит почту, номер телефона, соц. Сети, форму быстрой подписки и форму обратной связи.

4. Каталог — содержит в себе карточки с достопримечательностями, фильтрацию и сортировку каталога.

5. Новости — отображает все новости, которые динамически подгружаются в зависимости от прокрутки страницы пользователем.

6. Динамическая страница карточки места — содержит информацию об месте, город, возможность поставить оценку достопримечательности и добавить в избранное.

Схема сайта и расположение всех блоков на страницах:

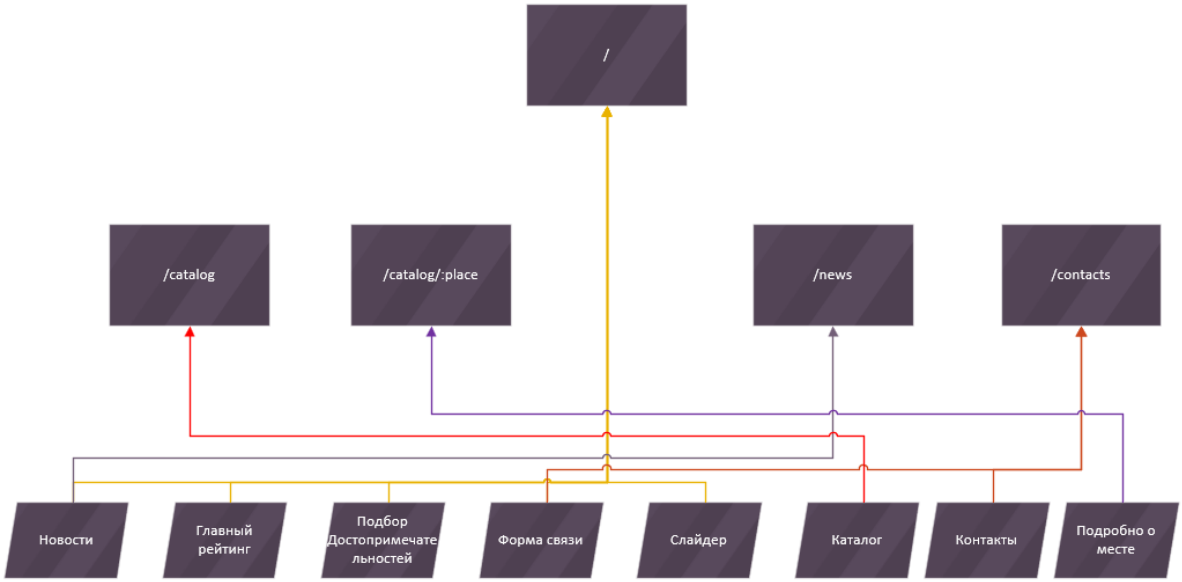


Схема 1 — Схема расположения блоков

5.2 Дизайн страниц

Проектирование и дизайн страниц сделано в веб-приложении — Figma.

Дизайн шапки и подвала. Слева в шапке будет располагаться логотип, по середине 3 ссылки, справа блок поиска и блок просмотра избранного. В подвале сайта снизу слева будет располагаться ссылка-логотип ведущая на главную, под ней политика конфиденциальности, по середине контакты и слева форма быстрой подписки на новости.

Дизайн главной страницы. Сверху будет располагаться слайдер — слева текст, снизу управление слайдером, на весь экран изображение к слайду. Ниже идет блок новостей, карточка новости содержит — справа картинку и дату размещения, слева текст и заголовок. После идут два блока с достопримечательностями: первый главные в рейтинге — показывает 6 достопримечательностей и ниже кнопку для просмотра всего каталога; второй блок в виде каталога, это случайно подобранные достопримечательности. И последнее это большая форма обратной связи.

Дизайн страницы каталога. В вверху слева кнопка открытия меню сортировки, справа кнопка открытия меню фильтрации, по середине отображаются выбранные фильтры. Ниже фильтров основной контент, показывается по 6 карточек, в низу расположена пагинация.

Дизайн страницы новостей. Показывается по 3 блока новости, далее подгружаются остальные.

Дизайн страницы контактов. Слева в столбик расположены почта, номер телефона, соц. сети, а справа форма подписки. Снизу большая форма обратной связи.

Дизайн динамической страницы места. Сначала идет заголовок, ниже расположен блок оценки достопримечательности, справа от него модуль избранного, еще правее информация о городе. Ниже идет большое изображение и текст к нему.

6 Разработка веб-приложения

6.1 Написание клиентской части

6.1.1 Клиентская маршрутизация

Для веб-приложения, помимо серверной маршрутизации требуется еще и клиентская. Схема клиентской маршрутизации, которая написана в приложении:

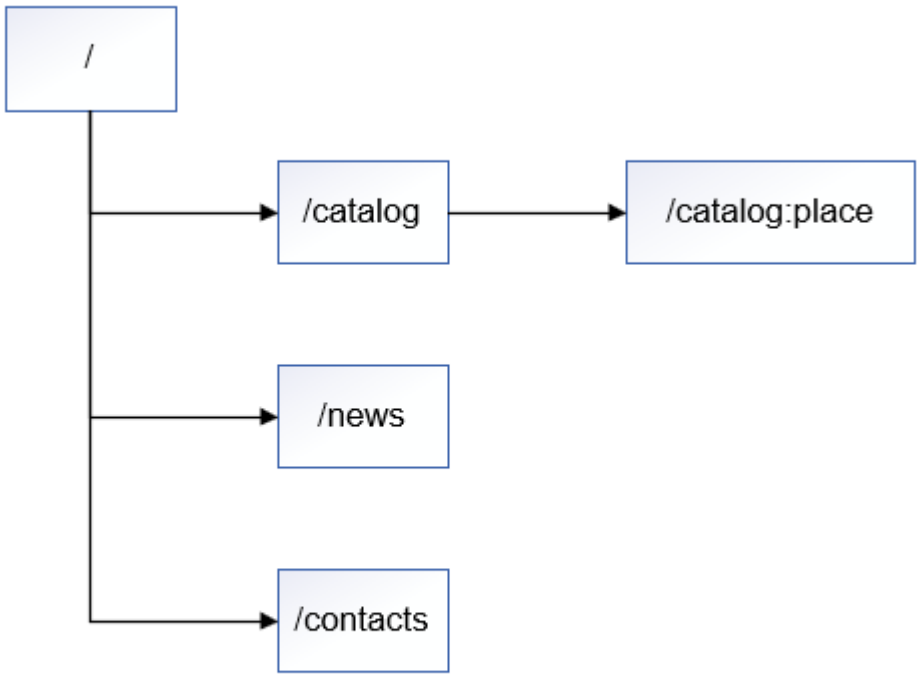


Схема 2 — Клиентская маршрутизация

6.1.2 Верстка страниц и реализация логики элементов

Верстка будет осуществляется с использованием **JSX (JavaScriptXML)**, в качестве препроцессора для стилей будет использоваться **SCSS**.

Первое с чего надо начать это верстка статичных элементов — шапка и подвал, они будут на каждой странице, то есть все следующие страницы будут помещаться между двумя этими блоками. Далее по порядку — главная страница, каталог, новости и контакты. Такие элементы как заголовки, кнопки, ссылки, формы, карточки, они будут пере используемые.

Рисунок 4.1 — Главная страница сайта

Рисунок 4.2 — Главная страница сайта

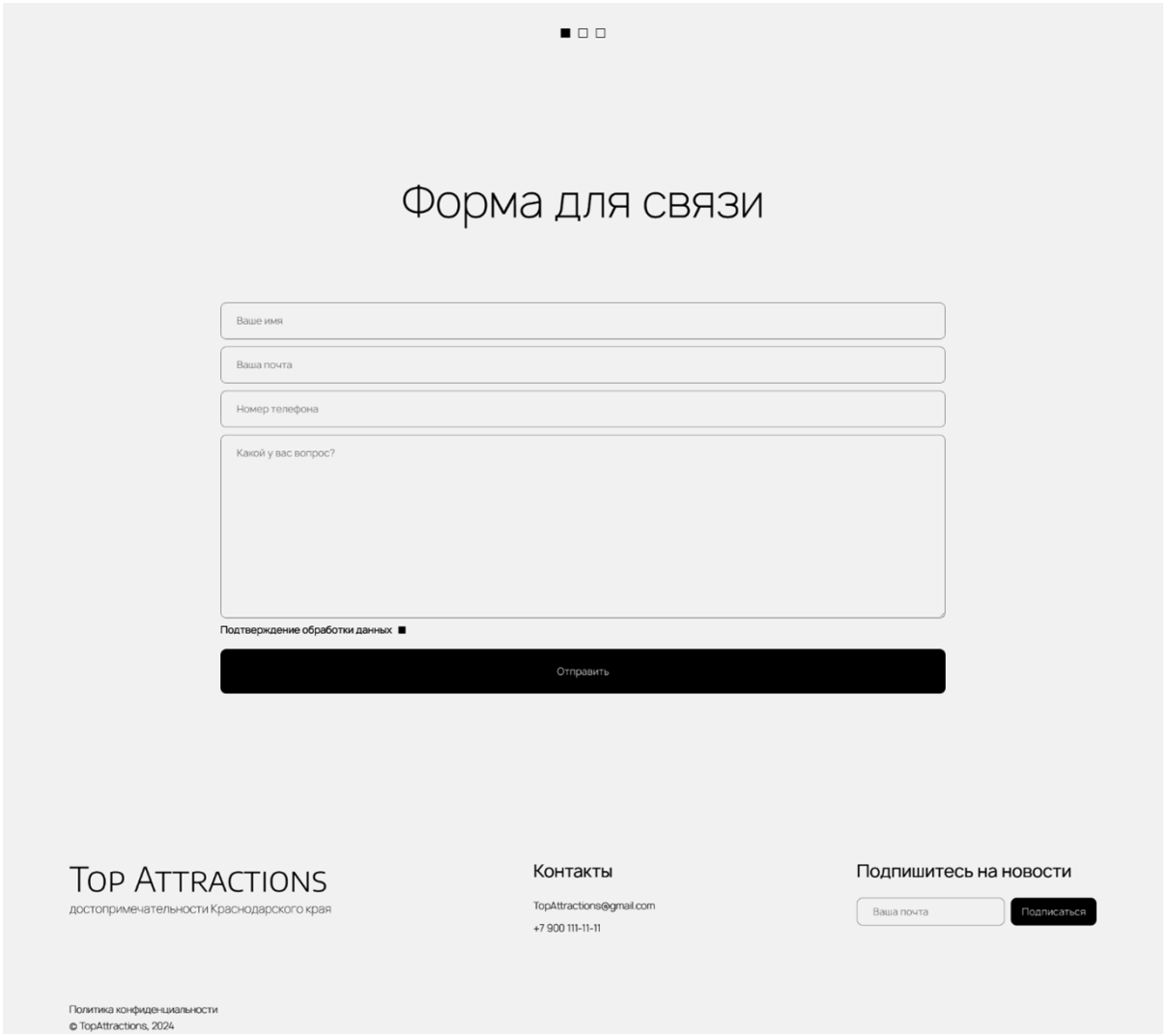


Рисунок 4.3 — Главная страница сайта

Рисунок 5 — Страница каталога

Рисунок 6 — Страница новостей

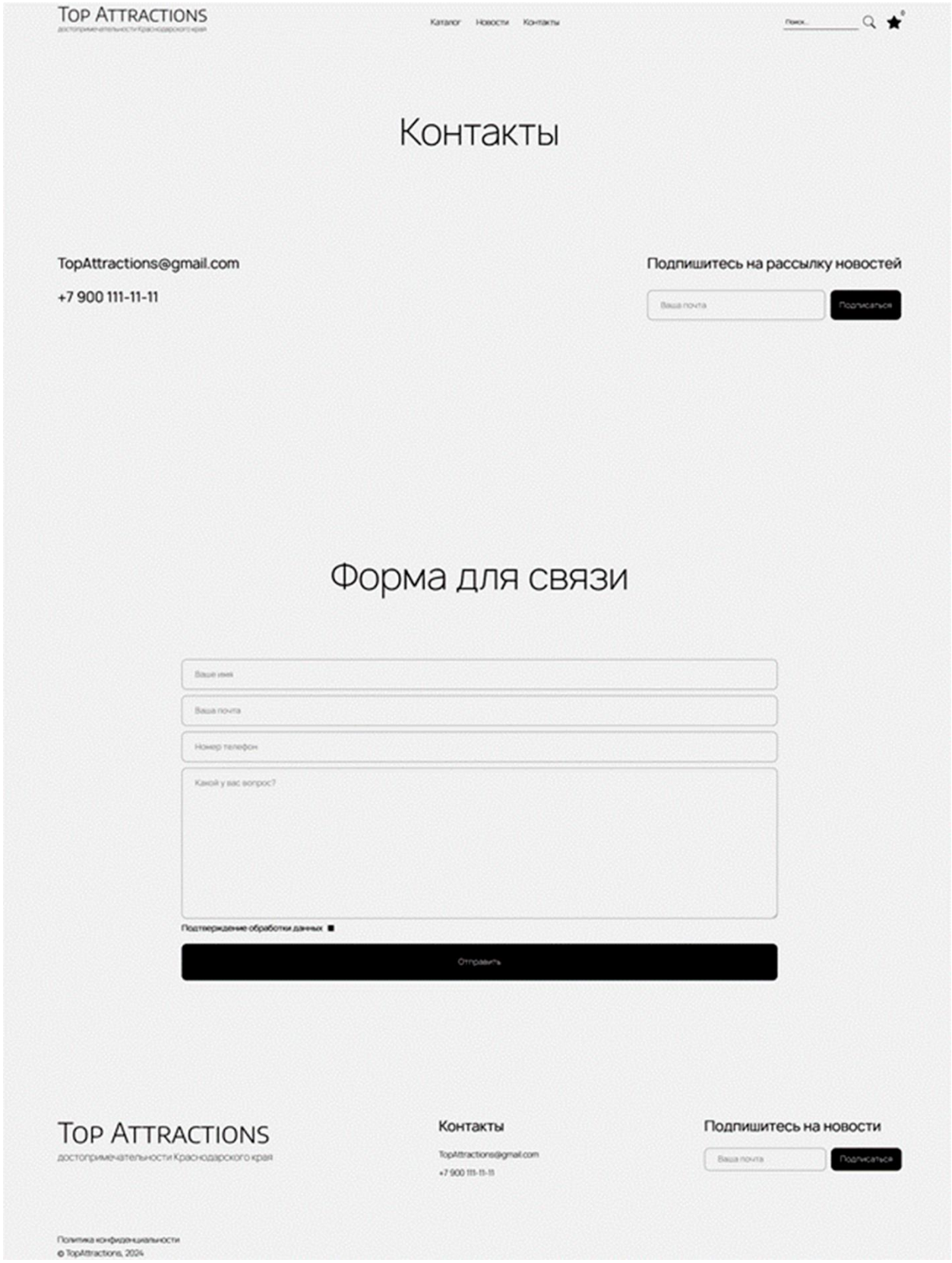


Рисунок 7 — Страница контактов

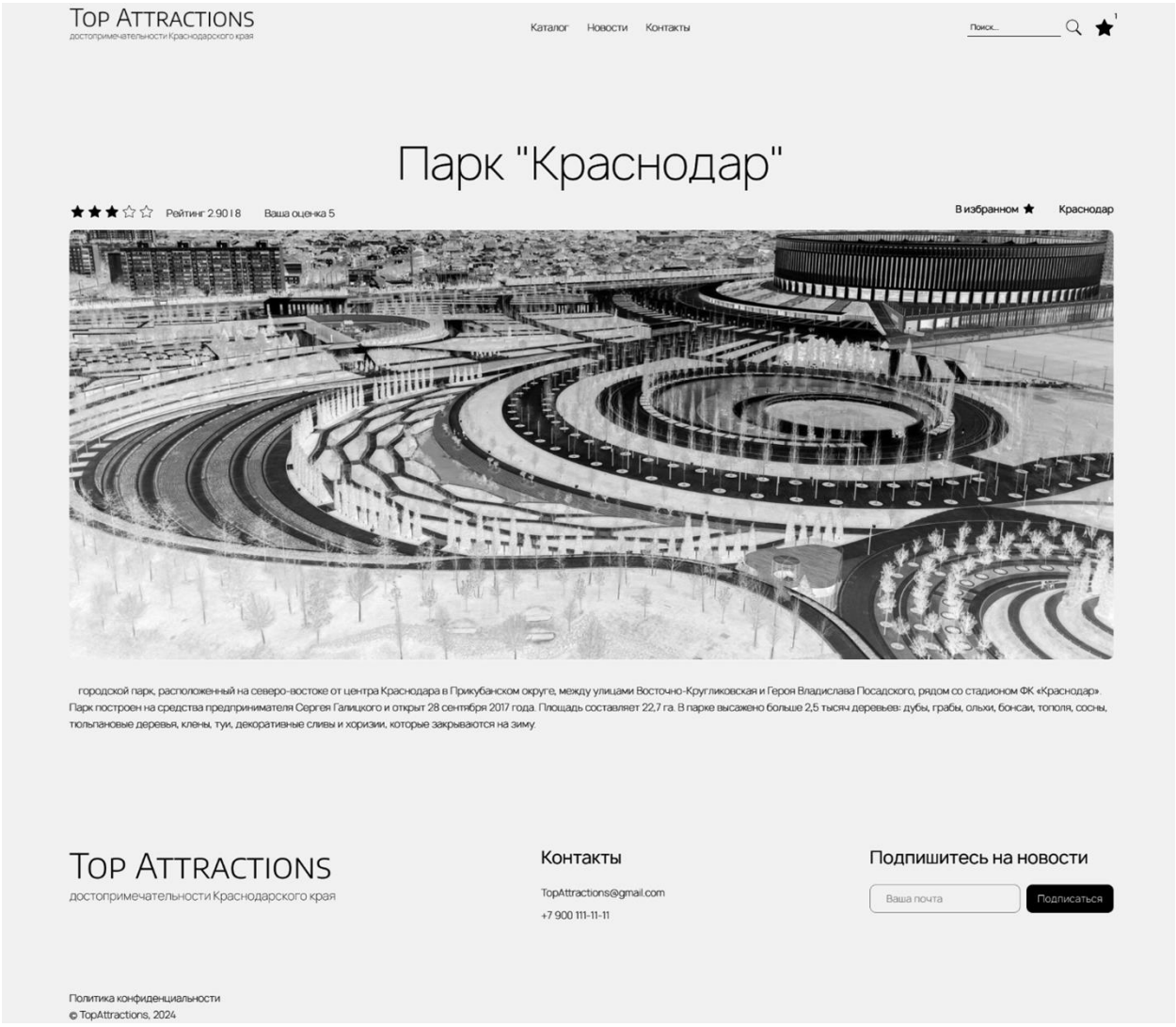


Рисунок 8 — Динамическая страница места

6.1.3 Написание запросов к серверу

Для написания запросов к серверу и формирования логики от них будет использоваться технология асинхронных запросов Fetch, так же будет задействована технология SSR, что позволяет делать запрос, на получение информации, еще на сервере, сгенерировать страницу и отдать клиенту.

При возврате любой уже сгенерированной страницы клиенту, уходят следующие запросы с его браузера на сервер: проверка на голосование пользователя, если пользователь не голосовал, то откроется форма голосования, в которой если пользователь проголосует, то вернутся

результаты голосования. Так же уходит запрос на то, что была просмотрена текущая страница, если такой ip адрес еще не просматривал эту страницу, тогда просмотры выбранной страницу увеличиваются на 1.

- Для главной страницы будет браться информации путем SSR: об слайдере — текста и картинки; информация о новостях — последнние 3 новости, каждая содержит заголовок, изображение, текст и дату публикации; 2 запроса на получение информациии об главных в рейтинге достопримечательностей и подбор достопримечательностей.
- Страница контактов будет статичная.
- Для страницы новостей загружаются последние 3 новости, далее с помощью запроса методом GET с параметром count, берутся следующие новости.
- На странице каталога есть 2 запроса: первый на получение первых шести мест и максимум страниц, второй на получение списка городов, для которых есть достопримечательности. Два запроса выполняются синхронно и асинхронно друг от друга. При выборе каких-либо фильтров запросы выполняются повторно, но уже с дополнительными GET параметрами.
- На динамической странице достопримечательности, выполняется один запрос методом GET который передает параметром текущий артикул места.
- Отдельные запросы из клиентского браузера:

1. При добавлении места в избранное отправляется запрос с его id, далее идет проверка есть ли место с таким id и если да, то оно добавляется JSON форматом в куки;
2. При удаление места из избранного, отправляется так же запрос с его id, в php скрипте идет проверка существует ли такой id в куке пользователя, если да тогда это место удаляется из избранного, если нет очевидно что запрос был отправлен через консоль самостоятельно пользователем и соответственно выбрасывается ошибка со статусом 400 (Bad Request);

3. Запрос на отправку формы обратной связи: при успешном заполнении всех обязательных полей и нажатии кнопки отправки, идет проверка на клиенте всех полей через регулярные выражения, если все поля прошли проверку они отправляются на сервер методом POST, в php скрипте дальше идет глубокая проверка всех полей на правильность ввода и на их безопасность. Если все успешно, то данная форма отправляется в базу данных.

4. Форма подписки на новости: при нажатии на кнопку подписаться, идет проверка поля через регулярное выражение, при успешности отправляется запрос методом POST, на сервере идет еще проверка поля, и далее записывается в базу данных.

5. Запрос на поиск по достопримечательностям: поиск реализован динамический, то есть при каждом нажатии появляются результаты. Что бы не грузить сервер запросами при каждом нажатии, запрос отправляется раз в 2 сек, тоесть при нажатии отправляется запрос с ожиданием, если за эти две секунды было введено новое значение, то это значение заменяет предыдущее в запросе. Схема запроса:

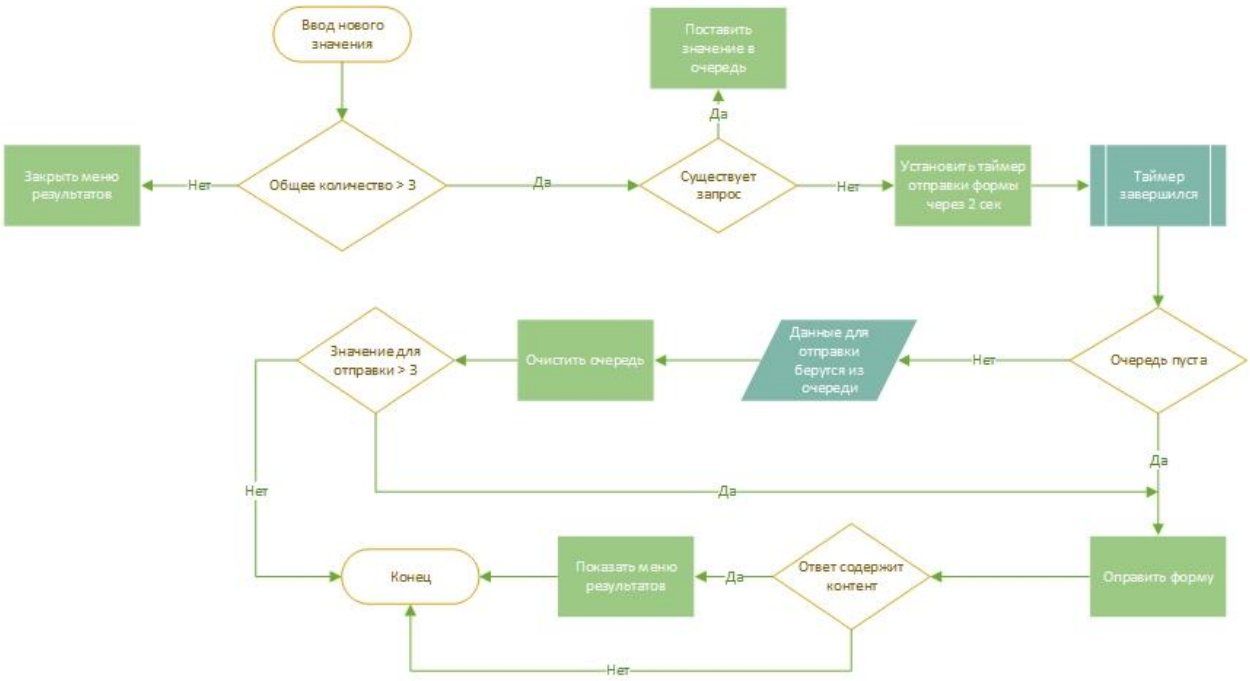


Схема 3 — динамический поиск

6. Запрос на изменение рейтинга места: при клике пользователем на звезду рейтинга места, отправляется запрос методом GET с параметрами id места и его оценки. В php скрипте выполняется проверка параметров, проверка существования места с таким id, далее идет выборка из бд текущего рейтинга и суммы голосов этого места, и идет пересчет рейтинга:

- Если пользователь уже голосовал то формула будет следующая:

$$\frac{X * V - (O - N)}{V}$$

- Если пользователь голосует впервые:

$$\frac{X * V + N}{(V + 1)}$$

Где:

X — текущий рейтинг;

V — текущее кол-во голосов;

N — новая оценка пользователя;

O — старая оценка пользователя.

Далее уходит запрос в бд на обновление данных, если сценарией при котором пользователь голосует в первые, то общее кол-во голосов очевидно инкриментится. Оценка пользователя добавляется/обновляется в JSON файле.

Все запросы будут кэшироваться средствами NEXT.js для повышения производительности.

6.2 Разработка базы данных

6.2.1 Проектирование базы данных

В базе данных содержится информация об каталоге, новостях, слайдере, голосовании, подписчиках, обратной связи и посещаемости.

Каталог (catalog) — в таблице есть следующие поля: уникальный индификатор (id), путь к изображению (image), заголовок (title), артикул (article), описание (text), рейтинг (rating), голоса (voices), город (city). Поле *id*

— первичный ключ, *text* — fulltext индекс, *title*, *city*, *article* — обычные индексы.

Новости (news) — содержит поля уникальный индификатор (id), заголовок (title), описание (text), дата (date), путь к изображению (image). Первичный ключ *id*, *text* — fulltext индекс, *title* — обычный индекс.

Слайдер (slider) — в таблице находится информация об слайдах для слайдера на первой странице, есть поля уникальный индификатор (id), путь к изображению (image), заголовок (title). Есть только первичный ключ *id*.

Таблица обратной связи (incoming) — в неё приходят пользовательские данные из формы обратной связи. Поля: уникальный индификатор (id), имя пользователя (user_name), почта пользователя (user_mail), телефон пользователя с возможностью оставить поле пустым (user_number), комментарий пользователя (text), дата отправки (date). Первичный ключ *id*, *text* — fulltext индекс, *user_name*, *user_mail*, *user_number* — обычные индексы.

Таблица подписчиков (subscribers) — в нее приходят данные пользователей, которые подписались на рассылку новостей. Есть поля уникальный индификатор (id), почта пользователя (user_mail), дата добавления (date). Из индексов только первичный *id*.

Таблица голосования (voting) — содержит поля ip пользователя хранящиеся в числовом формате и город (city), за который пользователь проголосовал. Есть уникальный индекс для поля *ip*.

Таблица посещения страниц (webrating) — в таблице содержатся поля для каждой странице и общее число. Поля *total, main, news, contacts, catalog и single* в качестве единичного индекса ограничения записей. В этой таблице существует одна запись, в которой обновляются результаты. При каждом обновлении таблице, то есть добавлении просмотра, срабатывает триггер, который инкрементит поле total.

База данных разработа в СУБД MySQL при помощи веб-приложения для администрирования MySQL — phpMyAdmin. Схема бд, составленная автоматически в phpMyAdmin, представлена ниже:

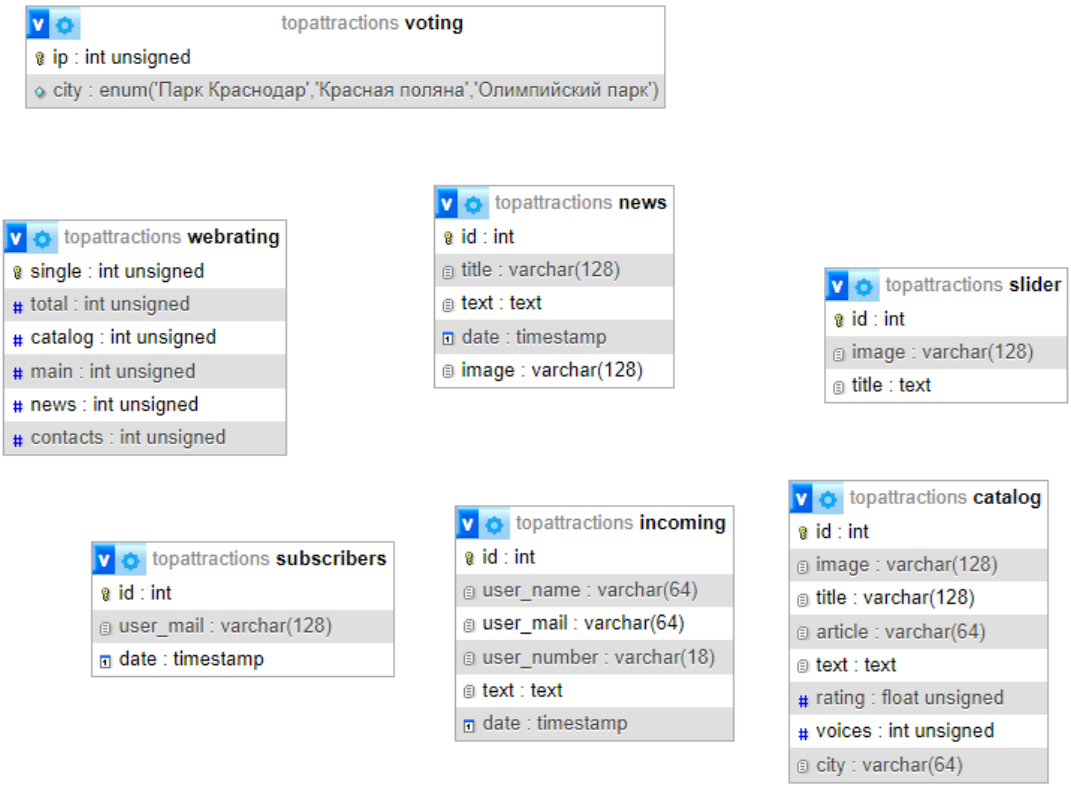


Рисунок 9 — Структура базы данных

6.3 Серверная часть

6.3.1 Создание базовой структуры сервера

Серверная часть сайта была написана на серверном языке программирования PHP с использование библеотеки Composer и была развернута на веб-сервере NGINX.

Разработка базовой структуры сайта (движка), начинается с расположения папок и файлов в каталоге. Весь серверный код лежит в папке server.

6.3.2 Серверная маршрутизации

В конфиге NGINX прописан блок для переадресации запроса на php скрипт index, если он начинается с /api/ и после имеет два уровня глубины, то есть /api/…/….

```
location ^~ /api/ {
    location ~* ^/api/([a-z]+)/([a-z-]+)$ {
        # alias nexttestServer/index.php;
        # alias finalServer/index.php;
        alias webcursachServer/index.php;
        fastcgi_buffering off;
        # fastcgi_index index.php;
        fastcgi_pass 127.0.0.1:9000;
        fastcgi_param SCRIPT_FILENAME $request_filename;
        include fastcgi_params;
        fastcgi_ignore_client_abort off;
    }
```

Листинг 1 — Блок переадресации NGINX

Далее распределение серверных маршрутов будет осуществлять php. Серверная маршрутизация будет следующая:

```
<?php
    return [
        '^api/slider/([a-z0-9-_]+)$'=>'_Slider65',
        '^api/news/([a-z0-9-_]+)$'=>'_News32',
        '^api/catalog/([a-z0-9-_]+)$'=>'_Catalog12',
        '^api/form/([a-z0-9-_]+)$'=>'_Form98',
        '^api/user/([a-z0-9-_]+)$'=>'_User35',
        '^api/metric/([a-z0-9-_]+)$'=>'_Metric4',
        '^api/vote/([a-z0-9-_]+)$'=>'_Vote66'
    ];
```

Листинг 2 — Серверная маршрутизая

Все запросы начинающиеся с api — означают обращения с клиентской части серверу.

slider — запросы связанные с получение информации об слайдере;

news — запросы связанные с блоком новостей;

catalog — запросы касающиеся получения карточек достопримечательностей;

form — отвечает за обработку форм на сайте;

user — запросы касающиеся индвидуального пользователя;

metric — запросы для отслеживания посещаемости;

vote — запросы касающиеся голосования.

6.3.3 Логика обработки запросов с клиентской части

Логика ответа на поступающие запросы следующая: все запросы поступают в главный файл и далее идет распределение через маршрутизатор в другие файлы-классы в зависимости от строки запроса.

Далее все файлы-классы по работе стандарты — при поподании в существующий класс, вызывается вызваная функция, если не переданы все нужные аргумены то сервер возвращает ошибку 400 Bad Request и завершает работу скрипта.

Есть файл functions.php в котором содержаться некоторые повторяющиеся функции.

6.3.4 Запросы к базе данных и обработка куки

Запросы к базе данных осуществляются с помощью встроенного модуля-класса для работы с бд — PDO (PHP data objects). PDO предоставляет единые методы для работы с базой данных в независимости от ее разновидности. Для удобства был написан класс Db в котором прописаны функции для отправки запросов и сразу обработки ответов. Функция `getPreparedQuery` обрабатывает подготовленный, “безопасный” запрос, методом PDO prepare, принимает в себя строку запроса, параметры для нее, и различные настройки. Функция `getQuery` отправляет обычный запрос без

предварительной обработки, методом PDO query, принимает строку запроса и различные настройки.

Далее представлены запросы во всех файлах-классах:

- Файл-класс catalog:

Получение контента для блока “Главные в рейтинге”:

```
try {
    $result = Db::getQuery("SELECT * FROM `catalog` WHERE voices >= 10 ORDER BY rating DESC LIMIT 6");

    exit(json_encode($result));

} catch (\Exception $e) {
    header("HTTP/1.0 500 Internal Server Error");
    die;
}
```

Листинг 3 — Запрос для блока “главные в рейтинге”

Запрос на получения контента для блока “Случайная подборка”:

```
try {
    $result = Db::getQuery("SELECT * FROM `catalog` ORDER BY rand() LIMIT 9");
    exit(json_encode($result));

} catch (\Exception $e) {
    header("HTTP/1.0 500 Internal Server Error");
    die;
}
```

Листинг 4 — Запрос для блока “случайная подборка”

При приходе запроса на получение каталога, сначала идет проверка, через цикл и конструкцию switch, GET параметров, в них передаются фильтры, далее параметры добавляются в строку запроса и в параметры. После выполняется запрос на получения числа страниц по данным фильтрам и запрос на получения каталога по выбранным фильтрам. 2 результата запроса отправляется клиенту массивом.

Где знаки вопроса, туда подставляются значения, переданные параметрами в функцию.

```php
if(isset($_GET['page']) && !is_numeric($_GET['page'])) {
    header("HTTP/1.0 400 Bad Request");
    die;
}
$query = "";
$sort = "";
$params = [];

foreach($_GET as $key=>$val) {
    if(empty($_GET[$key])) continue;
    $temp =  "(";
    switch($key) {
        case "cities":
            foreach(explode(',',$_GET['cities']) as $key=>$val) {
                if($val ==  "") continue;
                $temp .= " city = ? OR";

                array_push($params, ["VALUE"=>HgetSafeString($val), "PARAMVALUE"=>64]);
            }
            $query .= substr_replace($temp, '', -2) . ') AND ';
            break;

        case "favs":
            if(!isset($_COOKIE['favs']) || empty($_COOKIE['favs'])) break;
            foreach($_COOKIE['favs'] as $key=>$id) {
                if($val =="true") $temp .= " id = ? OR";
                    else if($val == "false") $temp .= " id != ? OR";

                array_push($params, ["VALUE"=>HgetSafeString($id), "PARAMVALUE"=>64]);
            }
            $query .= substr_replace($temp, '', -2) . ') AND ';
            break;
```

Листинг 5 — Запрос получение каталога

```php
        case "rating":
            $rating = explode(',', $_GET['rating']);
            if(!empty($rating[0]) && !is_numeric(($rating[0]))) break;
            if(!empty($rating[1]) && !is_numeric(($rating[1]))) break;
            if(!empty($rating[0]) && !empty($rating[1])) {
                $temp .= " rating >= ? AND rating <= ?";
                array_push($params, ["VALUE"=>HgetSafeString($rating[0]), "PARAMVALUE"=>64],["VALUE"=>HgetSafeString($rating[1]), "PARAMVALUE"=>64]);

            } else if (!empty($rating[0])) {
                $temp .= " rating >= ?";
                array_push($params, ["VALUE"=>HgetSafeString($rating[0]), "PARAMVALUE"=>64]);

            } else if (!empty($rating[1])) {
                $temp .= " rating <= ?";
                array_push($params, ["VALUE"=>HgetSafeString($rating[1]), "PARAMVALUE"=>64]);
            }
            $query .= $temp . ') AND ';
            break;
        case "sort":
            switch($val) {
                case "name_asc": $sort = " ORDER BY title ASC"; break;
                case "name_desc": $sort = " ORDER BY title DESC"; break;
                case "rating_desc": $sort = " ORDER BY rating DESC"; break;
                case "rating_asc": $sort = " ORDER BY rating ASC"; break;
                default: break;
            }
            break;
        default: break;
    }
}
if(!empty($query)) $query = " WHERE " . $query;
$query = substr_replace($query, '', -4) . $sort;

try {
    $page = isset($_GET['page']) ? ($_GET['page'] - 1) * 6 : 0;
    $pages = Db::getPreparedQuery("SELECT COUNT(*) as max FROM `catalog` " .$query, $params);
    array_push($params, ["VALUE"=>$page, "PARAMVALUE"=>64]);
    $result = Db::getPreparedQuery("SELECT * FROM `catalog` " .$query. " LIMIT ?, 6", $params);

    exit(json_encode(["data"=> $result, "pages"=>ceil($pages[0]['max']/6)]));

} catch (\Exception $e) {
    header("HTTP/1.0 500 Internal Server Error");
    die;
```

Листинг 6 — Запрос получение каталога

Запрос на обновление рейтинга места, сначала идет проверка обязательных параметров: id, он должен существовать и являться числом; mark, должен существовать, являться числом, его длина не должна быть больше 1 и число не должно превышать 5. Далее принцип обновления рейтинга был расписан в 4.1.2.

```
if(!isset($_GET['id']) || empty($_GET['id']) || !is_numeric($_GET['id'])) {header("HTTP/1.0 400 Bad Request"); die;}

if(!isset($_GET['mark']) || empty($_GET['mark']) || !is_numeric($_GET['mark']) || (strlen($_GET['mark']) > 1) || ($_GET['mark'] > 5))
    {header("HTTP/1.0 400 Bad Request"); die;}

$ips = json_decode(file_get_contents(DATA . 'rating.json', true), true);

if($ips[$_GET['id']] === null) {header("HTTP/1.0 400 Bad Request"); die;}

try {
    $currentRating = Db::getPreparedQuery("SELECT rating, voices FROM catalog WHERE id = ?",
        [["VALUE"=>HgetSafeString($_GET['id']), "PARAMVALUE"=>64]])[0];
    if(empty($currentRating))
        {header("HTTP/1.0 400 Bad Request"); die;}
    $newrating = '';
    $query = "";
    if(array_key_exists($_SERVER['REMOTE_ADDR'], $ips[$_GET['id']])) {
        $newrating = (($currentRating['rating'] * $currentRating['voices'])
                    - ($ips[$_GET['id']][$_SERVER['REMOTE_ADDR']] - $_GET['mark']))
                    / ($currentRating['voices']);

        $query = "voices = ".($currentRating['voices']);

    } else {
        $newrating = (($currentRating['rating'] * $currentRating['voices']) + $_GET['mark']) / ($currentRating['voices'] + 1);
        $query = "voices = ".($currentRating['voices'] + 1);
    }

    Db::getPreparedQuery("UPDATE `catalog` SET `rating`= ?, $query WHERE id = ?",
        [["VALUE"=>HgetSafeString($newrating), "PARAMVALUE"=>64], ["VALUE"=>HgetSafeString($_GET['id']), "PARAMVALUE"=>64]]);

    $ips[$_GET['id']][$_SERVER['REMOTE_ADDR']] = $_GET['mark'];
    file_put_contents(DATA . 'rating.json', json_encode($ips));

    header("HTTP/1.0 200 OK");
    die;

} catch (\Exception $e) {
    header("HTTP/1.0 500 Internal Server Error");
```

Листинг 7 — Запрос на обновление рейтинга

В запросе на получение контента по поиску, есть проверка обязательного параметра content, он должен существовать, и его длина должна быть в диапазоне от 3 до 50. Далее вся строка контента переводится в нижний регистр и осуществляется поиск в бд с помощью оператора LIKE.

```
if(!isset($_GET['content']) || empty($_GET['content']) || (strlen($_GET['content']) < 3 && strlen($_GET['content']) > 50))
{header("HTTP/1.0 400 Bad Request"); die;}
try {
    $s = "%" . HgetSafeString(mb_strtolower($_GET['content'], 'UTF-8')) . "%";
    $result = Db::getPreparedQuery("SELECT title, article, image FROM catalog WHERE LOWER(title) LIKE BINARY ? OR  LOWER(city) LIKE BINARY ?",
        [["VALUE"=>$s, "PARAMVALUE"=>64], ["VALUE"=>$s, "PARAMVALUE"=>64]]);
    exit(json_encode($result));

} catch (\Exception $e) {
    header("HTTP/1.0 500 Internal Server Error");
```

Листинг 8 — Запрос на поиск через строку

•Файл-класс news:

Запрос на получение новостей, если есть GET параметр ‘count’, идет проверка через регулярное выражение на то, что в нем есть только числа. Далее подготовленный запрос в бд.

```
if(isset($_GET['count']) && !preg_match('/^[0-9]*$/', $_GET['count']))
    {header("HTTP/1.0 400 Bad Request");die;}

try {
    $result = Db::getPreparedQuery("SELECT * FROM `news` ORDER BY date DESC LIMIT ?",
                                   [["VALUE"=>$_GET['count'] ?? 3, "INT"=> true]]);
    exit(json_encode($result));

} catch (\Exception $e) {
    header("HTTP/1.0 500 Internal Server Error");
    die;
}
```

Листинг 9 — Запрос получение новостей

•Файл-класс user:

При приходе любого запрос, он проходит через конструктор, в котором проверяется существует ли кука избранного, если нет то она создается, в случае если она существует в поле temp присваивается ее значение, предварительно распарсив ее из JSON формата.

```
public function __construct() {
    if(!isset($_COOKIE["favs"])) {
        setcookie("favs",json_encode([]), time()+3600, '/');
        die;
    }
    $this->temp = json_decode($_COOKIE["favs"], true);
}
```

Листинг 10 — Конструктор класса user

Запросы на добавление в избранное и удаления из него начинаются с проверки на существования параметра id и является ли он числом.

При добавлении в избранное далее происходит выборка из бд id, названия, артикула и изображения места, и в поле темп, где временно хранится значение куки, добавляется новое место и отправляется пользователю в виде новой куки.

При удалении из избранного, проверяется существования переданного id в куке, далее это значение удаляется и пользователю отдается новая кука.

```
if(!isset($_GET['id']) || empty($_GET['id']) || !is_numeric($_GET['id'])) {
    header("HTTP/1.0 400 Bad Request");
    die;
}
try {
    $id = Db::getPreparedQuery("SELECT id, title, article, image FROM catalog WHERE id = ?",
        [["VALUE"=>HgetSafeString($_GET['id']), "PARAMVALUE"=>64]]);

    if(isset($id[0]['id']) && !key_exists($id[0]['id'], $this->temp)) {

        $this->temp[$id[0]['id']] = [$id[0]['title'], $id[0]['article'], $id[0]['image']];

        setcookie("favs", json_encode($this->temp), time()+3600, '/');
        die;

    } else {
        header("HTTP/1.0 400 Bad Request");
        die;
    }

} catch (\Exception $e) {
```

Листинг 11 — Запрос на добавление в избранное

```
if(!isset($_GET['id']) || empty($_GET['id']) || !is_numeric($_GET['id'])) {
    header("HTTP/1.0 400 Bad Request");
    die;
}
if(key_exists($_GET['id'], $this->temp)) {
    unset($this->temp[$_GET['id']]);
    setcookie("favs", json_encode($this->temp), time()+3600, '/');
    die;
} else {
    header("HTTP/1.0 400 Bad Request");
```

Листинг 12 — Запрос на удаление из избранного

- Файл-класс vote:

В классе есть два метода, на проверку голосовал ли пользователь и на добавление голоса пользователя.

Запросом в бд осуществляется поиск ip текущего пользователя с помощью метода INET_ATON, который переводит ip в цифровой формат. Если ip не найдено, то возвращается массив вариантов голосования.

```php
try {
    $ip = Db::getQuery("SELECT ip FROM `voting` WHERE ip = INET_ATON('".$_SERVER['REMOTE_ADDR']."')");
    if(empty($ip)) {
        exit(json_encode(['Парк Краснодар', 'Олимпийский парк' , 'Красная поляна']));

    } else {
        die;
    }

} catch (\Exception $e) {
    header("HTTP/1.0 500 Internal Server Error");
```

Листинг 13 — Запрос на проверку голоса пользователя

В запросе на добавление нового голоса, сначала идет проверка на существования значения voice в массиве POST, далее конструкцией switch определяется переданное значение и отправляется в бд, после из таблицы голосования выбираются города, кол-во проголосовавших за них, и результат отправляют клиенту.

```php
if(empty($_POST) || !isset($_POST['voice']) || empty($_POST['voice']))
    {header("HTTP/1.0 400 Bad Request");die;}

$switch = "";

switch($_POST['voice']) {
    case "Красная поляна":
        $switch = "Красная поляна";
        break;
    case "Парк Краснодар":
        $switch = "Парк Краснодар";
        break;
    case "Олимпийский парк":
        $switch = "Олимпийский парк";
        break;
    default:
        header("HTTP/1.0 400 Bad Request");
        die;
}
try {
    Db::getQuery("INSERT INTO `voting`(`ip`, `city`) VALUES (INET_ATON('".$_SERVER['REMOTE_ADDR']."'),'".$switch."')");
    $count = Db::getQuery("SELECT city, COUNT(*) AS 'count' FROM `voting` GROUP BY city");
    foreach($count as $k=>$v) {
        if($v['city'] == $switch) {
            $count[$k]['selected'] = true;
            break;
        }
    }
    exit(json_encode($count));

} catch (\Exception $e) {
```

Листинг 14 — Запрос на добавления голоса пользователя

- Файл-класс slider:

Есть один метод, который возвращает выборку из бд всей таблицы slider.

•Файл-класс form:

В классе есть конструктор, в котором проверяется на пустоту массив POST, и с помощью функции `HcheckUserInput` проверяются все значения массива POST.

```
public function __construct() {
    if(empty($_POST)) {header("HTTP/1.0 400 Bad request");die;}
    $this->data = HcheckUserInput();
    if(empty($this->data)) {header("HTTP/1.0 400 Bad request"); die;}
}
```

Листинг 15 — Конструктор класса form

Обработка формы добавления подписки на новости. В поле текущих переданных параметров data должно быть одно значение, далее в таблицу subscribers отправляется новое значение.

```
if(count($this->data) != 1) {header("HTTP/1.0 400 Bad request"); die;}
try {
    Db::getPreparedQuery("INSERT INTO `subscribers` (`user_mail`) VALUES (?)",
        [["VALUE"=>HgetSafeString($this->data['mail']), "PARAMVALUE"=>64]]);
    exit(json_encode(["message"=>"Ваша заявка отправлена."]));

} catch (\Exception $e) {
    header("HTTP/1.0 500 Internal Server Error");
    die;
}
```

Листинг 16 — Обработка формы подписки

Обработка большой формы обратной связи. Переданных данных должно быть 5 или 4, но без номера телефона. Далее через конструкцию switch подготавливаются параметры для запроса и в зависимости от кол-ва их количества отправляются разные запросы в бд.

```php
if((count($this->data) == 4 && isset($this->data['number'])) || (count($this->data) != 4 && count($this->data) != 5))
    {header("HTTP/1.0 400 Bad request"); die;}
try {
    $params = [];
    foreach ($this->data as $k=>$v) {
        switch($k) {
            case "number":
                array_push($params, ["VALUE"=>HgetSafeString($v), "PARAMVALUE"=>18]);
                break;
            case "mail":
            case "name":
                array_push($params, ["VALUE"=>HgetSafeString($v), "PARAMVALUE"=>64]);
                break;
            case "question":
                array_push($params, ["VALUE"=>HgetSafeString($v), "PARAMVALUE"=>300]);
                break;
            default: break;
        }
    }
    $query = count($this->data) == 5
        ? "(`user_name`, `user_number`, `user_mail`, `text`) VALUES (?, ?, ?, ?)"
        : "(`user_name`, `user_mail`, `text`) VALUES (?, ?, ?)";
    Db::getPreparedQuery("INSERT INTO `incoming` ". $query, $params);
    exit(json_encode(["message"=>"Ваша заявка отправлена."]));

} catch (\Exception $e) {
    header("HTTP/1.0 500 Internal Server Error");
    die;
```

Листинг 17 — Обработка формы обратной связи

Файл-класс metric:

В классе есть метод increase, который принимает в себя один параметр — страницу, с которой пришел запрос. Метод запускается с разными параметрами, в зависимости от переданной странице в запросе.

Работа метода: из папки распарсивается JSON-файл переданной страницы, и проверяется существует ли в нем текущее ip пользователя, если нет, то в бд инкрементится эта страница, а в JSON файл добавляется новое ip.

```php
$metric = json_decode(file_get_contents(DATA . 'metric_'.$type.'.json', true), true);

if(!in_array($_SERVER['REMOTE_ADDR'], $metric)) {
    try {
        Db::getQuery("UPDATE `webrating` SET `".$type."` = `".$type."` + 1 WHERE single = 1");

        array_push($metric, $_SERVER['REMOTE_ADDR']);
        file_put_contents(DATA . 'metric_'.$type.'.json', json_encode($metric), LOCK_EX);
        header("HTTP/1.0 200 OK");
        die;

    } catch (\Exception $e) {
        header("HTTP/1.0 500 Internal Server Error");
        die;
    }
} else {
    header("HTTP/1.0 200 OK");
    die;
}
```

Листинг 18 — Метод increase

6.4 Тестирование

6.4.1 Типы тестирования

Веб-тестирование — это процесс проверки и оценки качества веб-приложений или сайтов, чтобы обеспечить их работоспособность, надежность, производительность, безопасность и совместимость с различными браузерами, устройствами и операционными системами. Типы веб-тестирования:

1. *Тестирование функциональности*: Функциональное тестирование является самым базовым, но чрезвычайно важным для любого приложения, в том числе веб-приложения. Функциональное тестирование гарантирует, что веб-приложение работает верно и без ошибок. Инструменты веб-тестирования проверяют, например, то, что каждая ссылка на сайте ведет на нужную страницу. Функциональное тестирование охватывает:

- Модульное тестирование: на этом этапе функционального тестирования проверяются небольшие отдельные области приложения на ранних стадиях разработки, чтобы снизить вероятность появления более серьезных ошибок в дальнейшем.
- Смоук-тестирование, тестирование билдов и тестирование полноты проводятся после каждого билда, чтобы убедиться, что веб-приложение стабильно и готово к дальнейшему тестированию.
- Тестирование работоспособности (санити): после завершения проверки билда этот тест проверяет применение нового кода и выбранную функциональность.
- Регрессионное тестирование: повторно проверяет определенный набор тестовых случаев, чтобы определить области, которые наиболее чувствительны к вносимым изменениям, чтобы убедиться, что существующая функциональность осталась неизменной.

- Интеграционное тестирование: находит ошибки в работе взаимосвязанных модулей (например, перенаправление на страницу почтового ящика после успешной регистрации).
- Тестирование удобства использования (юзабилити): используется для поиска мест, в которых может быть улучшен интерфейс пользователя в соответствии с полученными отзывами и реальными пользовательскими сценариями.

Функциональное тестирование также проверяет формы ввода, чтобы убедиться, что они работают.

2. *Тестирование веб-API.* Веб-API, как следует из названия, представляют собой интерфейсы прикладного программирования для веб. Тестирование API требует выполнения запросов к нескольким конечным точкам API для проверки ответа, включая функциональность, безопасность и производительность. Оно имеет ключевое значение, потому что оно проверяет узкие места логики, ответов, безопасности и производительности.

В терминах API проверяется обмен данными HTTP на шифрование и защищённость от потенциальных атак. При этом следует обращать внимание на такие негативные сценарии как:

- Доступ к API без соблюдения правил аутентификации;
- Проверка недопустимых значений в JSONS (например, невозможно зарегистрировать пользователя с несуществующим адресом электронной почты);
- Тестирование файлов cookie — это еще один аспект, на который следует обратить внимание при отправке пользовательских запросов и сеансов. При использовании API в качестве посредника способы удаления или хранения информации о сеансах и действиях пользователей должны включаться в тестовые сценарии API.

3. *Тестирование баз данных.* Тестирование баз данных гарантирует, что значения данных, хранящихся в базе данных, верны. Это поможет

предотвратить потерю данных, сохранить сведения об изменениях и предотвратить несанкционированный доступ к информации.

Тестирование включает проверку схемы базы данных, таблиц и триггеров распространенных баз данных, таких как Excel/CSV, GraphQL, Oracle SQL и SQL Server. Оно также часто включает стресс-тестирование и использование сложных запросов на одном или нескольких файлах данных. Тестирование баз данных обеспечивает уверенность в том, что все данные передаются успешно, независимо от числа запросов к базе.

4. *Тестирование совместимости с браузерами, операционными системами и мобильными устройствами*. Тестирование совместимости гарантирует, что основные функции веб-приложения будут доступны пользователям в различных браузерах и на разных устройствах. Причем это касается не только разных операционных систем, но и разных их версий.

5. *Тестирование пользовательского интерфейса и визуальных элементов (UI/UX)*. Такое тестирование требует пристального внимания и зоркого глаза, чтобы убедиться, что все тексты читаются так, как должны, а все изображения находятся в правильных местах и имеют верные пропорции.

6. *Тестирование веб-безопасности* — один из самых важных этапов веб-тестирования. В частности, надежное веб-тестирование гарантирует, что ваше веб-приложение выдержит любые попытки взлома или утечки данных.

Так же существуют: *Тестирование производительности и скорости загрузки, регрессионное тестирование*.

6.4.2 Описание технология, инструментов и план тестирования

В этом веб-приложении будут применяться следующие виды тестирования:

- Тестирование UI/UX;
- Тестирование веб-API;
- Тестирование базы данных;

- Функциональное тестирование.

Функциональное тестирование будет проводится с помощью инструментов тестирования — JEST и REACT TESTING LIBRARY.

JEST — это фреймворк для тестирования JavaScript, разработанный для обеспечения уверенности в правильной работе любого JavaScript кода. Он позволяет вам писать тесты с приемлемым, знакомым и функциональным API, и быстро достигать желаемых результатов.

REACT TESTING LIBRARY — это библиотека для тестирования компонентов React. Она предоставляет простые функциональные возможности поверх react-dom и react-dom/test-utils, что способствует улучшению методов тестирования.

Остальные виды тестирования будут проходить в ручном режиме.

Функциональное тестирование будет проходить по модульному принципу. *План тестирования*:

1. Тестирование статичных отдельных компонентов на странице, таких как каталоги и новости. Проводится тестирование правильности их рендеринга и обработки полученных данных из API;
2. Тестирование динамичных отдельных компонентов на странице, таких как формы и слайдеры. Будет проводится так же тестирование успешности рендеринга, а также взаимодействия с ними.
3. Тестирование динамичных дополнительных компонентов, таких как фильтры, пагинация, форма поиска, меню и т.п.. Проверка на рендеринг и успешности взаимодействия.

В тестировании будут применятся такие технологии как:

- Тестирование с использованием снимков. это инструмент для проверки, что пользовательский интерфейс не изменяется неожиданным образом. Типичный тест с использованием снимков сначала рендерит UI-компонент, создает снимок на основе рендера, затем сранивает его с эталонным снимком, который хранится вместе с тестом;

- Запросы — это методы, которые библиотека тестирования предоставляет для поиска элементов на странице (`getBy…`, `findBy…`, `queryBy…`);
- Mocks (имитации) —используются для имитации функционала с помощью мок данных. Например, вместо того, чтобы обращаться к удаленному ресурсу такому как веб-сайт или база данных, можно создать собственную заглушку, которая позволит использовать имитируемые данные. Это гарантирует, что тесты будет быстрыми и надежными.

Всего вышло 32 теста. Для каждого пункта плана я прикреплю несколько тестов.

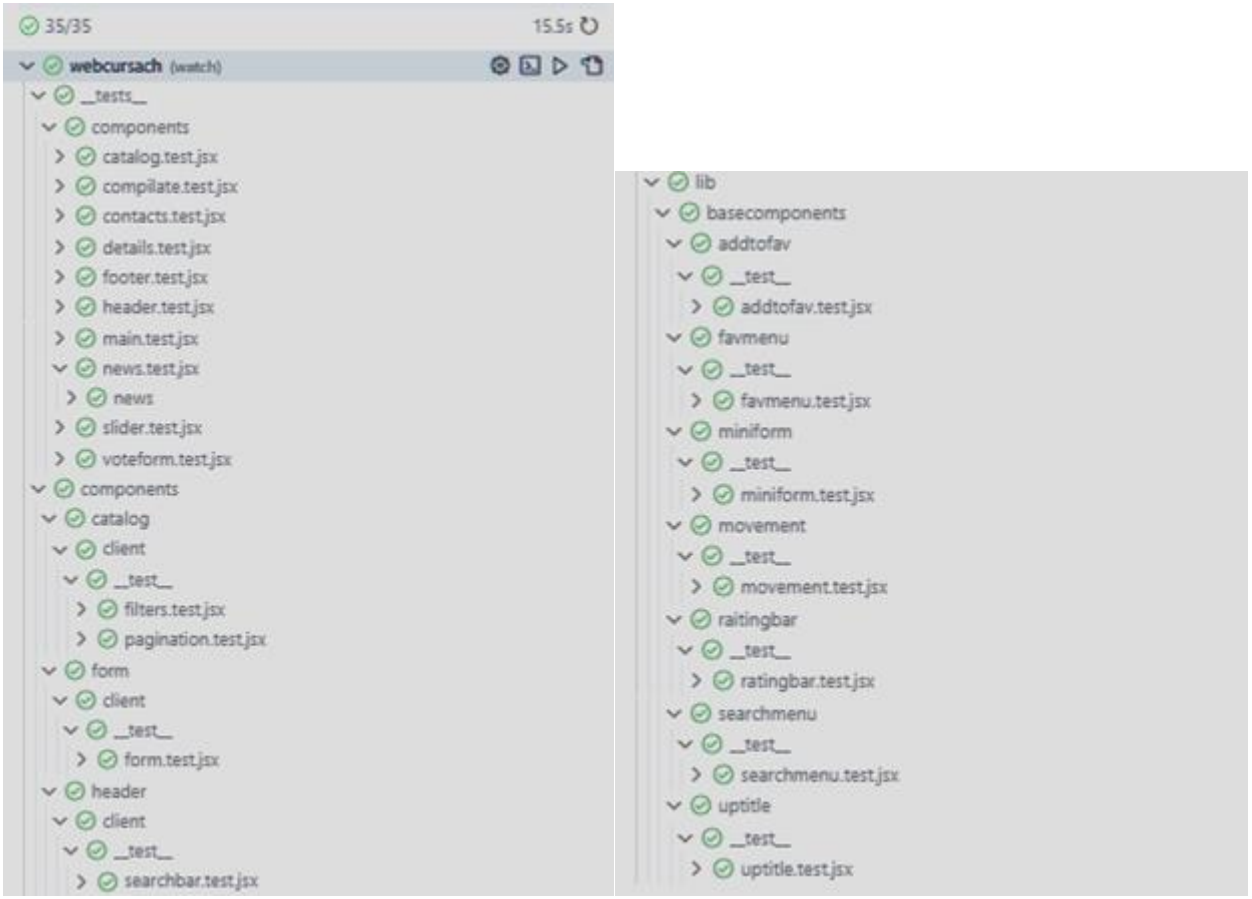


Рисунок 10 — Все тесты

Перед началом тестирования нужно определить 2 мока функций, которые понадобятся в дальнейшем.

Первый мок — добавление контекста к обычному методу render. В него передается сам компонент и “мокнутые” функции контекста, используемые в переданном компоненте.

```
__mocks__ > JS customRender.js > ...
1  import ClientContext from '@/lib/context/ClientContext';
2  import { render } from '@testing-library/react';  209.5k (gzipped: 44k)
3
4  export default customRender = (ui, {providerProps}) => {
5      return render(
6          <ClientContext.Provider value={{...providerProps}}>{ui}</ClientContext.Provider>,
7      )
8  }
```

Листинг 19 — Мок функции render

Второй мок — замена js API fetch на свою “мокнутую” функцию. Из тест файлов не получится обращаться напрямую к серверу через fetch api, плюс это будет занимать больше времени, поэтому для тестирования функционала, в эту функцию будет передаваться ответ, который должен получить компонент, далее возвращается промис объекта с полем ok в значении истинно и полем json которое возвращает еще один промис с запрашиваемым контентом, так же в эту функцию будет передаваться колбэк функция в которой есть возможность разграничить запросы и выдавать разный, это нужно если из компонента уходят сразу два запроса на сервер одновременно.

```
__mocks__ > JS fetchMock.js > ...
1  export default function settingFetchResult (val, callback = false){
2
3      global.fetch = jest.fn((recived) => {
4          let result = callback ? callback(recived) : val;
5
6          return Promise.resolve({
7          ok: true,
8          json: () => Promise.resolve(result)
9
10     })});
11
12 }
```

Листинг 20 — Мок API fetch

6.4.3 Тестирование статичных компонентов

Тест компонента catalog начинается с мока хуков Next из раздела navigation, так как они задействуются в клиент-компонентах, которые находятся в catalog. Далее прописывается контекст, а именно поле favorites.

В функции callback прописаны разграничения по запросом, во втором случае выдается объект data с массивом объектов городов. Из вызванного метода-мока customRender выбирается container для сравнения снимков. После

рендеринга, через цикл for, запрос findByText и метод toBeInDocument, проверяется наличие всех блоков. Далее сравниваются снимки.

```
__tests__ > components > catalog.test.jsx > ...
import settingFetchResult from "@/__mocks__/fetchMock";
import 'intersection-observer';  9.6k (gzipped: 3.3k)
import Catalog from "@/components/catalog/server/Catalog";
import { usePathname, useSearchParams, useRouter } from 'next/navigation';  13.3k (gzipped: 3.1k)
import customRender from "@/__mocks__/customRender";
import { screen } from "@testing-library/react";  207.4k (gzipped: 43.3k)

jest.mock('next/navigation');
useRouter.mockReturnValue({
    push: jest.fn(),
});
usePathname.mockReturnValue("localhost");
useSearchParams.mockReturnValue({});

const context = {
    favorites: {
        2: ["Озеро Сукко", "Ozero Sukko", "catalog/kiparis_ozero.webp"],
    }
}

describe('catalog testing', ()=>{
    it('catalog test',async ()=>{
        const blocks = ['one1', 'Озеро Сукко', 'one3'];
        const cities = blocks.map((each, i)=>{
            return {
                article: each+i,
                image: '/img/image1.jpg',
                title: each,
                id: i,
                text: 'sometext',
                rating: 5,
                voices: 3
            }
        });
        function callback(recived) {
            if(recived.includes('catalog/get-cities')) {
                return ['somecity'];

            } else if (recived.includes('catalog/get-catalog')) {
                return {data: cities};
            }
        }
        settingFetchResult([], callback);
        const {container} = customRender(await Catalog({searchParams: ""}), {providerProps: context});
        for(let block of blocks) {
            expect(await screen.findByText(block)).toBeInTheDocument();
        }
        expect(container).toMatchSnapshot();
    });
```

Листинг 21 — Тестирование компонента Catalog

Компонент News требует так же мока хуков navigation и тестируется 3 тестами. Первый тест — стандартное поведение компонента, передается контент, потом проверка на наличие этого контента, сравнение снимков. Второй тест — стандартное поведение уменьшенной версии компонента, здесь нужно проверить на наличие ссылки “Смотреть все новости” и ее svg элемента. Третий тест — проверка на выброс ошибки, если пришел пустой

массив, то должен быть элемент “Нет новостей”, а если контент не пришел вовсе, то должен быть элемент “Ошибка загрузки”.

```
__tests__ > components > news.test.jsx > ...
import customRender from "@/__mocks__/customRender";
import News from "@/components/news/server/News";
import 'intersection-observer';  9.6k (gzipped: 3.3k)
import { screen } from "@testing-library/react";  207.4k (gzipped: 43.3k)
import settingFetchResult from "@/__mocks__/fetchMock";
import { usePathname, useSearchParams, useRouter } from 'next/navigation';  13.3k (g

jest.mock('next/navigation');
useRouter.mockReturnValue({
    push: jest.fn(),
});
usePathname.mockReturnValue("localhost");
useSearchParams.mockReturnValue({});

describe('news',()=>{
    it('full news', async ()=>{
        const titles = ['some1', 'some2', 'some3'];

        settingFetchResult(titles.map((each,i)=>{
            return  {
                date: "22.02.22",
                title: each,
                text: "text"+each,
                image: `/img/image${i}.png`
            }

        }));
        const {container} = customRender(await News({full: true}), {});
        for(let title of titles) {
            expect(await screen.findByText(title)).toBeInTheDocument();
        }
        expect(container).toMatchSnapshot();
    });
    it('lite news', async ()=>{
        settingFetchResult([]);
        customRender(await News({full: false}), {});
        expect(await screen.findByText('Смотреть все новости')).toBeInTheDocument();
        expect(await screen.findByRole('arrow')).toBeInTheDocument();
    });
    it('full news exeptions', async ()=>{
        settingFetchResult([]);
        customRender(await News({full: true}), {});
        expect(await screen.findByText('Нет новостей')).toBeInTheDocument();
        expect(await screen.findByRole('anchor')).toBeInTheDocument();
        settingFetchResult(false);
        customRender(await News({full: true}), {});
        expect(await screen.findByText('Ошибка загрузки')).toBeInTheDocument();
    });
```

Листинг 22 — Тестирование компонента News

Тестирование подвала веб-приложения. Здесь нет никаких запросов к API, поэтому будет обычное тестирование с помощью снимка. После рендеринга проверяется наличие всех изображений, их должно быть 3, всех заголовков — 2, остальных блоков с информацией — 3, должна быть ссылка, и существование мини формы — должна быть кнопка “Подписаться” и поле ввода.

```
__tests__ > components > footer.test.jsx > ...
1   import 'intersection-observer';  9.6k (gzipped: 3.3k)
2   import customRender from '@/__mocks__/customRender';
3   import Footer from '@/components/footer/server/Footer';
4   import { screen } from '@testing-library/react';  207.4k (gzipped: 43.3k)
5
6   it('footer snapshot', async () => {
7       const {container} = customRender(<Footer/>, {});
8       const images = screen.getAllByRole('img');
9       const titles = screen.getAllByRole('heading');
10      const info = screen.getAllByRole('info');
11      expect(images.length).toBe(3);
12      expect(titles.length).toBe(2);
13      expect(info.length).toBe(3);
14      expect(await screen.findByRole('link')).toBeInTheDocument();
15      expect(await screen.getByText('Подписаться')).toBeInTheDocument();
16      expect(await screen.getByPlaceholderText('Ваша почта')).toBeInTheDocument();
17
18      expect(container).toMatchSnapshot();
19  });
```

Листинг 23 — Тестирование компонента Footer

6.4.4 Тестирование динамичных отдельных компонентов

Компонент form, сначала нужен объект контекста, далее функция делегирования элементов формы, что бы для каждого теста было удобно выбирать нужные поля.

Первый тест — проверка на правильный рендеринг, по “плэйсхолдерам” берутся все поля ввода формы, берутся все лэйблы с ошибками, далее сравнивается их количество с нужным, так же проверяется наличие чекбокса и кнопки отправки формы.

Второй тест — проверка функциональности формы, из функции делегирования достаются элементы: поле ввода имени, почты, номера, сообщения; чекбокс и кнопка отправки. Все поля заполняется в правильной форме, проверяется обработка маски номера телефона, далее клик по чекбоксу и по кнопки отправки формы, после проверяются поля на отсутствие значений.

Следующий тест — проверка на отклонение отправки формы, если не нажат чекбокс. Заполняются все поля, нажимается кнопка, проверяется на то, что поле ввода не очищено.

Четвертый тест — отклонение отправки формы, если не правильный адрес почты. Заполняются корректно все поля, кроме почты, отправляется форма, далее проверяется поле почты на то же значение, и появление ошибки.

После вводится корректно почта, проверяется на исчезновение ошибки и отправляется, поле ввода почты должно быть пустым.

components > form > client > __test__ > form.test.jsx > ...

```jsx
import { render, screen } from "@testing-library/react";  209.5k (gzipped: 44.1k)
import ClientForm from "../ClientForm";
import customRender from "@/__mocks__/customRender";
import userEvent from "@testing-library/user-event";  58.8k (gzipped: 17.8k)
import settingFetchResult from "@/__mocks__/fetchMock";
const notifications = {
    setNotification: ()=>{},
    notification: {
        status: false,
        message: ""
    }
}

const delegation = ()=>{
    return {
        name: screen.getByPlaceholderText('Ваше имя'),
        mail: screen.getByPlaceholderText('Ваша почта'),
        text: screen.getByPlaceholderText('Какой у вас вопрос?'),
        submit: screen.getByRole('button'),
        mailerror: screen.getAllByRole('error')[1],
        checkbox: screen.getByRole('checkbox'),
        number: screen.getByPlaceholderText('Номер телефона')
    }
}

describe('test form', ()=>{
    settingFetchResult({message:"done"});
    it('test render', async ()=>{
        const {container} = customRender(<ClientForm/>, {providerProps: notifications});
        const errors = screen.getAllByRole('error');
        const inputs = screen.getAllByPlaceholderText(/(Ваше имя){1}|(Ваша почта){1}|(Какой у вас вопрос?){1}|(Номер телефона){1}/);

        expect(errors.length).toBe(3);
        expect(inputs.length).toBe(4);

        expect(screen.getByRole('checkbox')).toBeInTheDocument();
        expect(screen.getByRole('button')).toBeInTheDocument();

        expect(container).toMatchSnapshot();

    });
    it('functional test - success way',async ()=>{
        customRender(<ClientForm/>, {providerProps: notifications});
        const {name, mail, number, text, submit, checkbox} = delegation();

        await userEvent.type(name, 'Имя');
        await userEvent.type(mail, 'example@mail.ru');
        await userEvent.type(text, "some question");
        await userEvent.type(number, "79182195584");

        expect(name).toHaveValue("Имя");
        expect(number).toHaveValue("+7 (918) 219-55-84");

        await userEvent.click(checkbox);
        await userEvent.click(submit);

        expect(name).toHaveValue("");
        expect(number).toHaveValue("");
    });
```

Листинг 24 — Тестирование компонента form (1)

Последний тест — проверка реактивного (динамического) изменения значения поля имени. При вводе неподходящих символов в поле, они должны удалятся из него.

```
60      it('functional test - rejected due to checkbox',async ()=>{
61          customRender(<ClientForm/>, {providerProps: notifications});
62          const { name, mail, text, submit } = delegation();
63
64          await userEvent.type(name, 'Имя');
65          await userEvent.type(mail, 'example@mail.ru');
66          await userEvent.type(text, "some question");
67
68          await userEvent.click(submit);
69
70          expect(name).toHaveValue("Имя");
71      });
72      it('functional test - rejected due to wrong mail',async ()=>{
73
74          customRender(<ClientForm/>, {providerProps: notifications});
75          const {name, mail, text, submit, checkbox, mailerror} = delegation();
76
77          await userEvent.type(name, 'Имя');
78          await userEvent.type(mail, 'example@mailru');
79          await userEvent.type(text, "some question");
80
81          await userEvent.click(checkbox);
82          await userEvent.click(submit);
83
84          expect(mail).toHaveValue("example@mailru");
85          expect(mailerror.className.includes('Form__form__input__error_show')).toBeTruthy();
86
87          await userEvent.clear(mail);
88          await userEvent.type(mail, 'example@mail.ru');
89
90          expect(mailerror.className.includes('Form__form__input__error_show')).toBeFalsy();
91
92          await userEvent.click(submit);
93          expect(mail).toHaveValue("");
94
95      });
96      it('functional test - test wrong name',async ()=>{
97          customRender(<ClientForm/>, {providerProps: notifications});
98          const { name } = delegation();
99
100         await userEvent.type(name, 'Имя4223');
101         expect(name).toHaveValue("Имя");
102     });
103 });
```

Листинг 25 — Тестирование компонента form (2)

Компонент voteform (форма голосования). Для тестирования понадобится контекст, в котором в том числе будут и варианты для голосования. Первый тест — тестирование корректности рендеринга, проверяются кнопки, их должно быть 2, проверяется наличие заголовка, наличие всех вариантов голосования и в конце сравнение со снимком. Второй тест — функциональность, здесь используется метод spyOn позволяющий отслеживать вызов какого-то метода, в данном случае для API fetch. Далее в контекст вариантов голосования заменятся на один вариант, для корректности тестирования, берется вариант голосования, кнопка отправки и имитируются

клики в этом же порядке, потом через spyOn проверяется была ли вызвана fetch API и изменился ли заголовок на ‘Результаты голосования’.

```
__tests__ > components > voteform.test.jsx > ...
1  import { screen } from "@testing-library/react";  207.4k (gzipped: 43.3k)
2  import customRender from "@/__mocks__/customRender";
3  import Voteform from "@/components/voteform/client/Voteform";
4  import userEvent from "@testing-library/user-event";  58.8k (gzipped: 17.8k)
5  import settingFetchResult from "@/__mocks__/fetchMock";
6  const context = {
7      voteMenu: ['testcity1','testcity2', 'testcity3'],
8      setVoteMenu (val) { context.voteMenu = val},
9      setNotification: (val)=>{},
10 }
11
12 describe('test voteform', ()=>{
13     it('test render', async ()=>{
14         const {container} = customRender(<Voteform/>, {providerProps: context});
15         const btns = screen.getAllByRole('button');
16         expect(btns.length).toBe(2);
17
18         expect(screen.getByText('Проголосуйте за лучшую достопримечательность')).toBeInTheDocument();
19         context.voteMenu.forEach(each=>{
20             expect(screen.getByText(each)).toBeInTheDocument();
21         });
22
23         expect(container).toMatchSnapshot();
24     });
25     it('functional test', async ()=>{
26         settingFetchResult([]);
27         const spy = jest.spyOn(global, 'fetch');
28         context.voteMenu =['city'];
29         customRender(<Voteform/>, {providerProps: context});
30         const btns = screen.getAllByRole('button')[1];
31         const input = screen.getByRole('radio');
32
33         await userEvent.click(input);
34         await userEvent.click(btns);
35
36         expect(spy).toHaveBeenCalled();
37         expect(screen.getByText('Результаты голосования')).toBeInTheDocument();
38     });
```

Листинг 26 — Тестирование компонента voteform

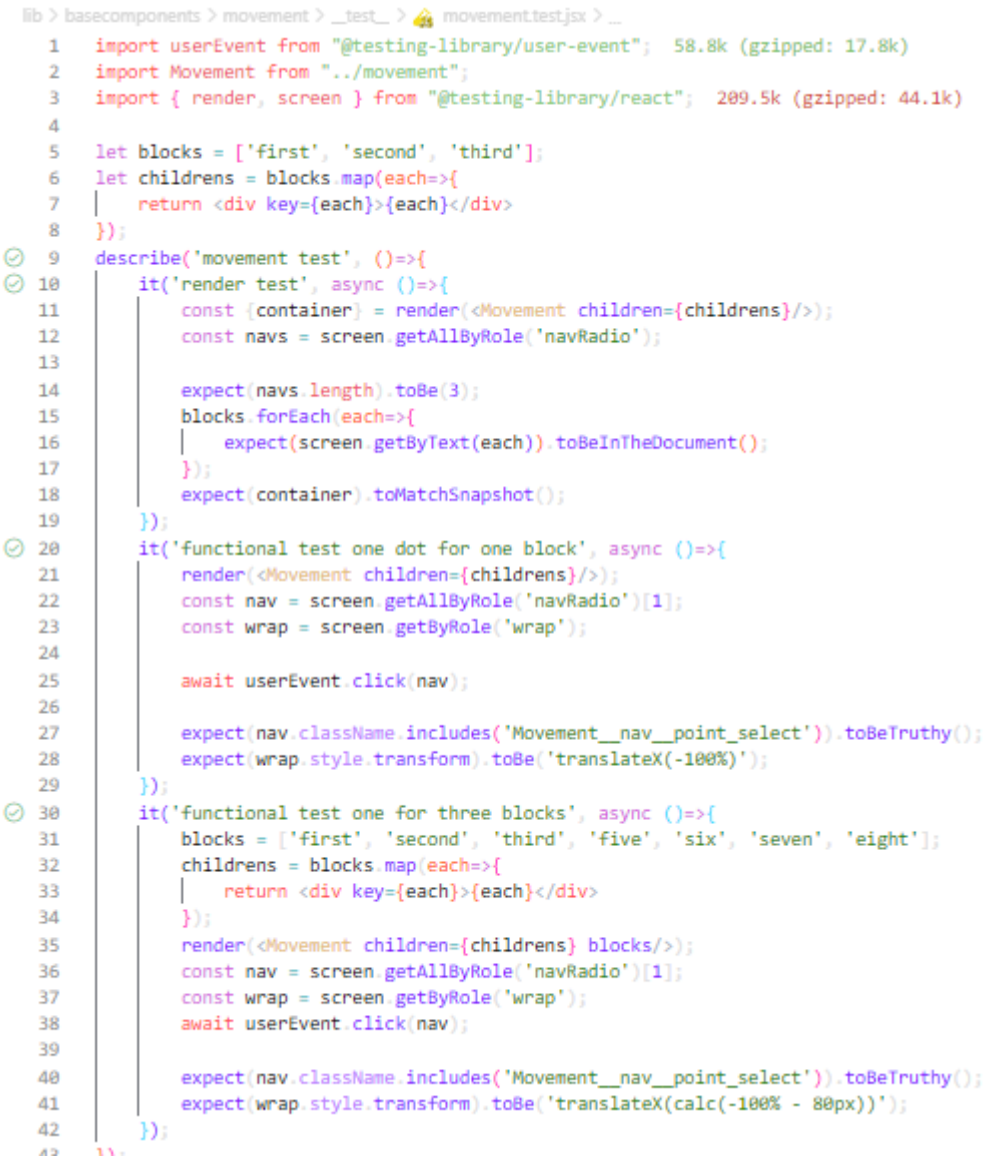
6.4.5 Тестирование динамичных дополнительных компонентов

Компонент movement, он отвечает за движение блоков слайдера и каталога подбора на главной странице. С начала нужно объявить и сгенерировать тестовые блоки. Далее первый тест — рендеринг, берутся все сгенерированные точки для навигации и проверяется эквивалентность 3-м их общее число, проверяется наличие всех блоков, потом сравнение со снимком.

Второй тест — тест функциональности компонента для слайдера, берется 2 точка и окно просмотра элементов, осуществляется имитация клика

по точке и проверяется наличие нового класса у этой точки, а также смещение окна просмотра.

Последний тест — тест функциональности для блоков (1 точка на 3 блока). Блоков понадобится больше (7) и все по тому же сценарию что и во втором тесте.



Листинг 27 — Тестирование компонента movement

components > catalog > client > __test__ > pagination.test.jsx > ...

```jsx
import { usePathname, useSearchParams, useRouter } from 'next/navigation'
import { render, screen } from "@testing-library/react";  209.5k (gzipped
import Pagination from '../Pagination';
jest.mock('next/navigation');
useRouter.mockReturnValue({
    push: jest.fn(),
});
usePathname.mockReturnValue("localhost");
useSearchParams.mockReturnValue({});

const urlparams = new URLSearchParams();
global.URLSearchParams = jest.fn(x => (urlparams));


describe('pagination test', ()=>{
    it('render test start', ()=>{
        const {container} = render(<Pagination page={1} max={40}/>);
        const dots = screen.getAllByText('...');

        expect(dots.length).toBe(1);
        expect(screen.getByText(1)).toBeInTheDocument();

        Array.from({length: 5}).forEach((_,i)=>{
            expect(screen.getByText(1+i+1)).toBeInTheDocument();
        });

        expect(screen.getByText(40)).toBeInTheDocument();

        expect(container).toMatchSnapshot();
    });
    it('render test middle', ()=>{
        const {container} = render(<Pagination page={8} max={40}/>);
        const dots = screen.getAllByText('...');


        expect(screen.getByText('1')).toBeInTheDocument();
        expect(dots.length).toBe(2);
        expect(screen.getByText(8)).toBeInTheDocument();
        expect(screen.getByText(8-1)).toBeInTheDocument();
        expect(screen.getByText(8-2)).toBeInTheDocument();

        Array.from({length: 5}).forEach((_,i)=>{
            expect(screen.getByText(8+i+1)).toBeInTheDocument();
        });
        expect(screen.getByText(40)).toBeInTheDocument();

        expect(container).toMatchSnapshot();
    });
    it('render test end', ()=>{
        const {container} = render(<Pagination page={40} max={40}/>);
        const dots = screen.getAllByText('...');

        expect(dots.length).toBe(1);
        expect(screen.getByText(1)).toBeInTheDocument();

        Array.from({length: 6}).forEach((_,i)=>{
            expect(screen.getByText(40-6+i)).toBeInTheDocument();
        });

        expect(screen.getByText(40)).toBeInTheDocument();

        expect(container).toMatchSnapshot();
    });
```

Листинг 28 — Тестирование компонента pagination

Компонент pagination. Сначала нужен мок раздела next/navigation, есть 3 теста — пагинация в начала, в середине и в конце.

В начале — проверяется эквивалентность единице количество элементов “троеточия”, наличие первой и последней страницы, и через цикл проверяется наличие всех страниц от 2 до 6, в конце сравнение со снимком.

В середине — проверяется наличие двух элементов “троеточия”, наличие 1, 7 6 и последней страницы, через цикл проверятся наличие страниц от 9 до 13, потом сравнение со снимком.

В конце — проверяется наличие одного элемента “троеточия”, наличие первой и последней страницы, через цикл проверяется наличие страниц от 34 до 39, и сравнение со снимком.

Заключение

В ходе выполнения курсового проекта было спроектировано и разработано веб-приложение веб-сервиса по просмотру достопримечатлеьностей.

Разработка программного продукта проводилась в редакторе исходного кода Visual Studio Code на языках программирования: JavaScript с использование библетеки React и фреймворка NEXT.js; PHP с использованием библиотеки Composer. Для разработки дизайна использовалось веб-приложение Figma. Для разбраотки базы данных использовалась СУБД MySQL и веб-приложение phpMyAdmin. Тестирование проводилось на фреймворке JEST и библиотеки REACT TESTING LIBRARY.

Было проведено описание структуры веб-приложения, кода клиентской и серверной части, базы данных и пользовательского интерфейса в целом.

В результате выполнения проекта были повторены старые и получены новые знания, новый опыт, а также готовый веб-сервис, полностью соответствующий поставленным требованиям и готовый к будущим улучшениям.

Список используемых источников

1. https://dev.mysql.com/doc/ — интернет документация MySQL;
2. https://survey.stackoverflow.co/2023/#most-popular-technologies-webframe-prof — самый популярные веб технологии, выбор пользователей StackOverflow;
3. https://habr.com/ru/articles/715376/ — Веб-тестирование;
4. https://developer.mozilla.org/ru/docs/Web — Руководство по веб-разработки и веб-технологиям, документация к языку программирования JavaScript;
5. https://fonts.google.com/ — Шрифты Google;
6. https://fusionbrain.ai/editor/ — Нейросеть для создания изображений;
7. https://icons.getbootstrap.com/ — Иконки Bootstrap;
8. https://nextjs.org/docs — ссылка на документацию NEXT.js;
9. https://ru.legacy.reactjs.org/ — интернет документация библеотеки React;
10. https://jestjs.io/ru/docs/tutorial-react — Документация фреймворка для тестирования JEST;
11. https://testing-library.com/docs/react-testing-library/intro/ — документация библиотеки для тестирования react компонентов REACT TESTING LIBRARY;
12. https://nginx.org/ru/docs/ — документация NGINX;
13. https://www.php.net/manual/ru/ — руководство по языку программирования PHP;
14. Динамические веб-сайты с помощью PHP, MySQL, JavaScript, CSS и HTML5. 6-е изд. | Никсон Робин, 2023г., 1200 стр.
15. Гагарина Л.Г., Кокорева Е.В., Сидорова-Виснадул Б.Д. Технология разработки программного обеспечения: учеб. пособие [Электронный ресурс]. – М.: ИНФРА-М, 2019. – 400 с. Режим доступа: http://znanium.com/catalog/product/1011120.