ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ
«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»

Факультет информатики, математики и компьютерных наук

**Программа подготовки бакалавров по направлению
09.03.04 Программная инженерия**

Овсянников Артём Сергеевич

КУРСОВАЯ РАБОТА

Сервис генерации изображений с помощью диффузионных моделей.
Архитектура и разработка сервиса

Научный руководитель
старший преподаватель НИУ
ВШЭ - НН

Саратовцев Артем Романович

Нижний Новгород, 2025г.

Содержание

Выбор инструментальных средств

В процессе реализации проекта по разработке веб-сервиса был осуществлён тщательный выбор инструментальных средств, обеспечивающих эффективность, гибкость и масштабируемость разработки. Выбор инструментов основывался на их функциональности, удобстве использования и совместимости с поставленными задачами.

Схемы архитектуры: Draw.io. Для визуализации архитектуры проекта использовался онлайн-сервис Draw.io. Этот инструмент предоставляет обширный набор шаблонов и графических элементов для создания блок-схем, UML-диаграмм, инфографики и других видов визуальных моделей. Его преимущества заключаются в:

- интуитивно понятном интерфейсе;
- возможности работы в браузере без установки дополнительного ПО;
- наличии функции совместной работы в реальном времени;
- поддержке различных форматов экспорта диаграмм;
- возможности интеграции с другими сервисами, такими как Google Drive и GitHub.

Управление проектом: Kaiten и GitHub Issues. На начальном этапе для управления задачами использовался сервис Kaiten, поддерживающий методологии Kanban и Scrum. Он позволяет эффективно организовать рабочий процесс, визуализировать задачи, устанавливать дедлайны и следить за их выполнением. Однако в процессе работы было принято решение перейти на использование GitHub Issues в связи с его более тесной интеграцией с кодовой базой проекта. GitHub Issues обладает следующими преимуществами:

- минималистичный, но функциональный интерфейс;
- возможность назначения задач, приоритетов, меток и привязки к Pull Request'ам;
- интеграция с системой контроля версий Git;
- поддержка автоматизации рабочих процессов с помощью GitHub Actions;
- возможность создания шаблонов задач и форм для их заполнения.

Дизайн макетов: Figma. Для проектирования пользовательского интерфейса использовался облачный редактор Figma, получивший широкое распространение благодаря своей доступности и функциональности. Он предоставляет средства для создания интерактивных прототипов, работы с векторной графикой и совместного редактирования в реальном времени. Основные преимущества Figma включают:

- возможность работы в браузере без установки дополнительного ПО;
- поддержка совместного редактирования и комментирования;
- наличие обширной библиотеки компонентов и стилей;
- интеграция с другими инструментами, такими как Slack и Zeplin;
- поддержка плагинов для расширения функциональности.

Фронтенд-разработка: HTML, CSS и JavaScript. Для реализации клиентской части проекта было принято решение использовать стандартные технологии - HTML, CSS и JavaScript, без привлечения дополнительных фреймворков, чтобы не усложнять искусственно проект. Это объясняется относительной простотой функционала веб-сервиса и стремлением к минимализму, позволяющему сохранить контроль над структурой и логикой интерфейса. Такой подход обеспечил:

- прозрачность кода и снижение сложности отладки;
- облегчение поддержки и расширения функционала;
- более глубокое понимание основ фронтенд-разработки;
- упрощение интеграции пользовательского интерфейса с серверной частью проекта;
- возможность оптимизации производительности за счёт отсутствия лишних зависимостей.

Таким образом, выбор каждого инструмента и модуля был обусловлен необходимостью решения конкретных задач проекта, их совместимостью и способностью обеспечить эффективную и гибкую работу с данными.

Проектирование реализации

Проектирование реализации веб-сервиса включает в себя несколько ключевых этапов и компонентов, направленных на обеспечение надёжности, масштабируемости и удобства сопровождения всей системы.

1. **Моделирование архитектуры:**

   - **Инструмент:** Draw.io.
   - **Процесс:** На основе требований к системе создаются блок-схемы и UML-диаграммы. С помощью Draw.io визуализируется общая структура сервиса, взаимодействие компонентов (клиент, сервер, БД, API-шлюзы, каждый находится в отдельном Docker контейнере) и потоки данных между ними. Параллельно определяется стек технологий для каждого компонента.

2. **Управление проектом:**

   - **Инструменты:** Kaiten и GitHub Issues.
   - **Процесс:**
     - *Этап инициации:*
       * Постановка стратегических целей и определение ключевых результатов через доску Kanban.
       * Распределение ролей и зон ответственности внутри команды (владельцы эпиков, ответственные за модули).
       * Составление предварительного плана проекта и определение критериев минимально жизнеспособного продукта (MVP).
     - *Миграция в GitHub Issues:*
       * Перенос согласованных задач из Kaiten в репозиторий с присвоением меток.
       * Создание Milestones для основных релизов и привязка к ним соответствующих Issues.
       * Назначение ответственных, выставление сроков и связывание с Pull Request'ами для полной прозрачности статуса.
     - *Регулярные практики:*
       * Еженедельные стендапы и краткий отчёт о проделанной работе.
       * Актуализация бэклога в GitHub Issues: создание новых и пересмотр приоритетов существующих задач, перераспределение по Milestones.

3. **Прототипирование пользовательского интерфейса:**

   - **Инструмент:** Figma.
   - **Процесс:**
     - **Разработка пользовательских сценариев (user flows):** на основе анализа требований и реальных кейсов использования формируются подробные последовательности действий пользователя от входа в систему до завершения ключевых операций (регистрация, генерация изображений и т. д.). Прорабатываются альтернативные ветви поведения, учитываются ошибки ввода и способы их обработки.
     - **Создание UI-кита и дизайн-системы:** определяется набор базовых компонентов (кнопки, поля ввода, карточки, модальные окна, типографика, цветовые палитры), описываются их состояния (hover, active, disabled) и взаимоувязки. Все элементы систематизированы в Figma, что обеспечивает консистентность интерфейса и ускоряет дальнейшую разработку.
     - **Построение интерактивных прототипов:** на основе UI-кита в Figma создаются интерактивные макеты ключевых экранов и переходов между ними. Прототипы включают кликабельные области и анимации переходов, что позволяет проверить логику навигации, оценить удобство расположения элементов и зафиксировать замечания ещё до начала кодирования.
     - **Проведение юзабилити-тестирования:** с участием реальных пользователей или представителей целевой аудитории организуются сессии, в ходе которых испытуемые выполняют типовые задачи с прототипом. С помощью наблюдений, опросов и записи экранов собирается обратная связь по понятности интерфейса, расположению элементов и навигации. На основании результатов вносятся корректировки в макеты и дизайн-систему.

4. **Реализация фронтенда:**

   - **Инструменты:** HTML, CSS, JavaScript.
   - **Процесс:**
     - **Настройка проекта и базовая структура:** инициализация репозитория, создание единой структуры каталогов (`/pages`, `/css`, `/js`, `/static`), настройка системы сборки для обработки модулей и статических ресурсов.

– **Реализация основных компонентов интерфейса:** по заранее утверждённым макетам Figma сверстаны ключевые элементы (шапка, навигация, карточки контента, формы), вынесены в отдельные блоки. Каждый компонент поддерживает несколько состояний и стилизован согласно дизайн-системе.
– **Оптимизация и адаптивность:** реализация корректного масштабирования на телефонах, планшетах и десктопах; внедрены техники для изображений и динамических модулей; проверена кроссбраузерная совместимость.
– **Подготовка к интеграции с бэкендом:** разработан слой взаимодействия с API: настроены базовые HTTP-запросы через API; описана структура ожидаемых эндпоинтов и форматов запросов/ответов; реализованы обработчики ошибок и отображение состояний ожидания.

Описание программной системы

Программная система предназначена для обеспечения устойчивого и автоматизированного функционирования веб-сервиса, развернутого в контейнеризованной среде. Архитектура системы спроектирована с учётом масштабируемости, модульности и лёгкости в сопровождении.

Архитектура системы

Система развёрнута в инфраструктуре, построенной на основе Docker-контейнеров. Каждый контейнер изолирует определённый компонент системы, обеспечивая независимость и управляемость. Архитектура включает четыре ключевых контейнера:

1. **Контейнер frontend:** содержит статическую веб-часть (HTML, CSS, JavaScript). Отвечает за отображение интерфейса пользователю и передачу данных на сервер через API-запросы.

2. **Контейнер backend:** содержит серверную часть и базу данных. Реализует бизнес-логику и взаимодействует с хранилищем данных. Обрабатывает входящие запросы, выполняет операции с БД и возвращает ответы клиенту.

3. **Контейнер MinIO:** отвечает за объектное хранение файлов, таких как изображения, видео и другие вложения. Используется как S3-совместимое хранилище, что облегчает масштабирование и интеграцию с другими сервисами.

4. **Контейнер CI/CD:** обеспечивает непрерывную интеграцию и развёртывание. Связан с остальными контейнерами и выполняет сборку, тестирование и выкладку новой версии проекта при изменении репозитория.

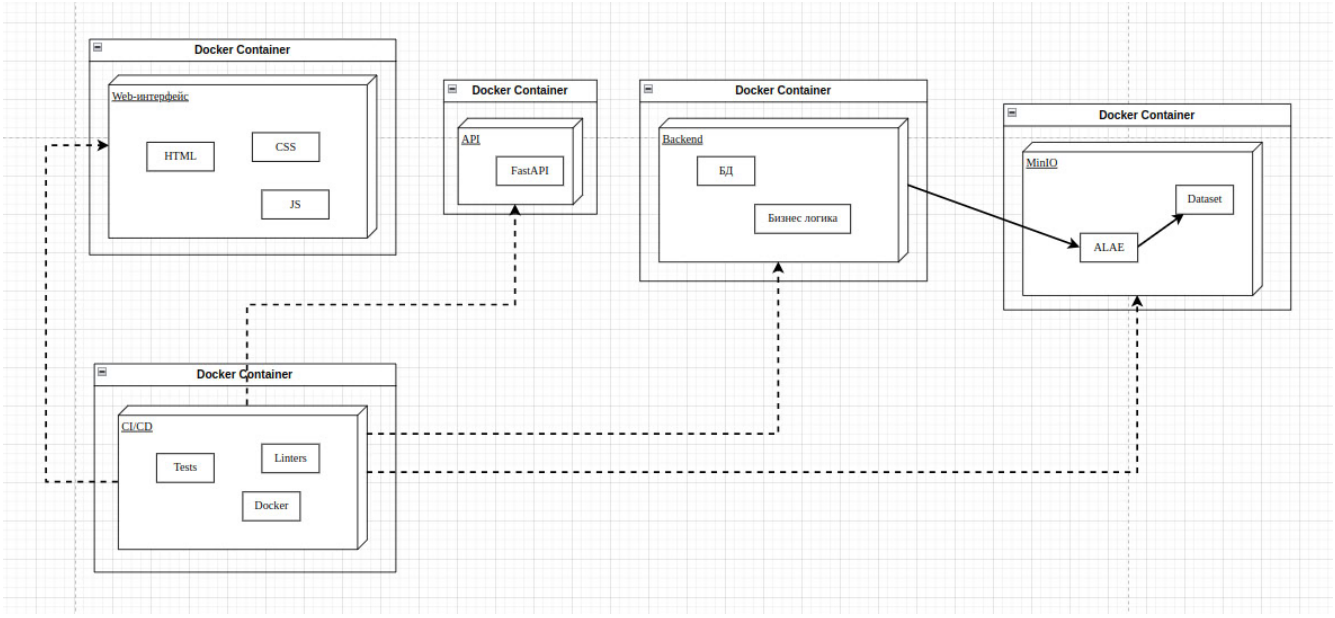

Рис. 1: Архитектура контейнерной системы веб-сервиса

Проектирование фронтенд составляющей

Фронтенд веб-сервиса реализован с использованием стандартных веб-технологий: HTML, CSS и JavaScript. Основная цель фронтенда - предоставить пользователю интуитивно понятный и функциональный интерфейс для взаимодействия с системой.

1. **HTML-структура:** каждая страница веб-сервиса оформлена в виде отдельного HTML-файла. Центральной точкой входа является `index.html`, который связывает остальные страницы между собой. В рамках архитектуры предусмотрена поддержка нескольких состояний для каждой страницы, например: отображение формы, результата или ошибки.

2. **CSS-оформление:** для стилизации используется единый файл стилей `style.css`, в котором определены базовые параметры внешнего вида, такие как цветовая палитра, отступы, шрифты, поведение элементов при наведении. Подход "разделения ответственности"облегчает сопровождение и масштабирование визуальной части.

3. **JavaScript-логика:** функциональность обеспечивается подключением JS-файлов. Для каждой страницы, где требуется динамика (например, валидация форм, отправка запросов, отображение полученных данных), используется собственный скрипт. Центральный скрипт `index.js` отвечает за обработку глобальных событий и маршрутизацию между страницами при необходимости.

4. **Организация кода:**

   - `/pages/` - директория с HTML-файлами;
   - `/css/` - содержит главный файл стилей;
   - `/js/` - JavaScript-скрипты по страницам;
   - `index.html` - точка входа в интерфейс;
   - `index.js` - логика начальной загрузки и обработки пользовательского взаимодействия.

5. **Особенности реализации:**

   - Поддержка адаптивного дизайна для корректного отображения на мобильных и десктопных устройствах;
   - Разделение данных и представления;

- Минимизация сторонних зависимостей для обеспечения высокой скорости загрузки и безопасности.

Файловая структура проекта представлена следующим образом:

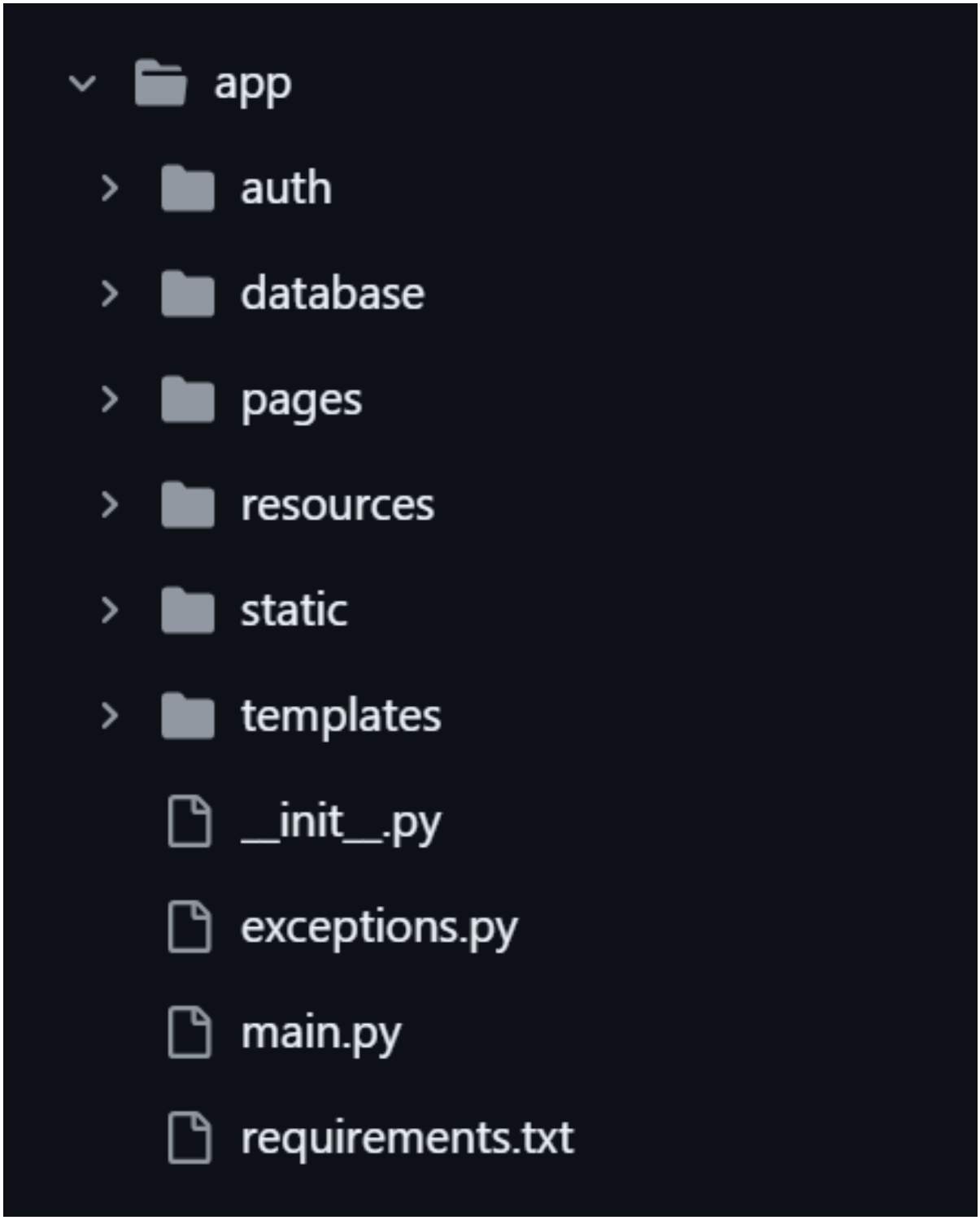

Рис. 2: Файловая структура сервиса

Таким образом, фронтенд спроектирован как легковесный и независимый компонент, тесно взаимодействующий с серверной частью через API-интерфейсы, предоставляя пользователю комфортный доступ к функциональности веб-сервиса.

Методика ручного тестирования пользовательских сценариев

Ручное тестирование представляет собой ключевой этап в обеспечении качества веб-приложения, при котором все сценарии взаимодействия пользователя с системой проверяются без использования автоматизированных скриптов. В процессе тестирования особое внимание уделялось как поведению авторизованного пользователя, так и действиям гостя (неавторизованного пользователя). Для каждого сценария фиксировались предварительные условия, точная последовательность действий, ожидаемый результат и фактическое поведение системы. Тестирование выполнялось при различных условиях сети (стабильное соединение, периодические задержки, полное отсутствие связи), на разных браузерах и устройствах (десктоп, планшет, мобильный), а также с учётом граничных значений вводимых данных. Все шаги прогонялись вручную членами команды по заранее составленному чек-листу, что позволило выявить как критичные, так и незначительные дефекты, а также проверить соответствие пользовательского интерфейса требованиям дизайна и удобства использования.

Сценарии тестирования

1. **Общие сценарии (доступно всем пользователям)**
   - Переход на `Home`: проверка отображения информации о проекте, актуальных ссылок на ключевые разделы и корректного рендеринга контента.
   - Просмотр раздела `About Us`: проверка отображения списка команды разработчиков, их ролей и контактных данных.
   - Навигация по общим ссылкам: проверка переходов и корректного открытия внешних/внутренних страниц.
   - Проверка адаптивности общего интерфейса на разных устройствах и в разных браузерах: правильная компоновка блоков, читаемость текста и доступность меню.
2. **Авторизованный пользователь**
   - Вход в профиль: ввод корректных учётных данных и отображения персональных настроек.
   - Просмотр и редактирование профиля: изменение имени, email, загрузка аватара; проверка сохранения и моментального обновления данных.

– Генерация изображений: отправка запроса и проверка корректного создания и отображения результата, возможность скачать сгенерированные изображения.
– Выход из системы: проверка завершения сессии, перенаправление на страницу `Home` и блокировка доступа к защищённым разделам.

Итог. По итогам ручного тестирования всех пользовательских сценариев можно заключить, что система полностью соответствует предъявленным требованиям по доступности и функциональности. Общие разделы (`Home` и `About Us`), включающие информацию о проекте, ссылки на документацию и представление команды разработчиков, отображаются корректно у всех пользователей без необходимости авторизации. Навигация по публичным страницам оказалась интуитивной и стабильной: переходы осуществляются без ошибок, адаптивная верстка сохраняет читаемость и удобство на разных устройствах и браузерах.

Сценарии для авторизованных пользователей подтвердили надёжность работы ключевых функций: вход в профиль и выход из системы проходят без задержек, изменения персональных данных мгновенно отражаются в интерфейсе, а механизм генерации изображений стабильно обрабатывает запросы.

Выявленные в ходе проверки небольшие несоответствия были задокументированы и переданы в систему отслеживания задач. Это позволит оперативно устранить мелкие недочёты и довести качество продукта до высшего уровня.

Таким образом, результаты ручного тестирования свидетельствуют о высокой степени готовности веб-приложения к развертыванию в продуктивной среде, обеспечивая надёжность, отказоустойчивость и удобство использования для всех категорий пользователей.

Пути возможного развития

Ниже приведены пути возможного развития системы в трёх ключевых направлениях: архитектура, дизайн и фронтенд. Они призваны обеспечить дальнейшую масштабируемость, удобство сопровождения и высокий уровень пользовательского опыта.

1. **Архитектурное развитие**
   - *Переход к микросервисной архитектуре:* декомпозиция монолитных контейнеров на узкоспециализированные сервисы (авторизация, генерация изображений, хранение метаданных и пр.) с собственными API и БД. Это позволит масштабировать наиболее нагруженные участки независимо и упростить тестирование каждого сервиса.
   - *Оркестрация и управление:* внедрение Kubernetes или Docker Swarm для автоматического развёртывания, масштабирования и самовосстановления контейнеров. Настройка Helm-чартов или Operator-ов для управления конфигурацией и релизами.
2. **Развитие в дизайне**
   - *Углубление дизайн-системы:* расширение UI-кита новыми компонентами (таблицы, диаграммы, уведомления), создание вариативных тем (светлая/тёмная, корпоративная), документирование всех правил использования с примерами кода (Storybook).
   - *Доступность (Accessibility):* приведение интерфейса в соответствие с WCAG 2.1: контрастность цветов, семантическая разметка, поддержка работы с клавиатурой и скринридерами. Регулярный аудит доступности с участием реальных пользователей.
   - *UX-исследования и тестирование:* проведение периодических A/B-тестов ключевых экранов и фич, опросов и глубинных интервью с пользователями. Использование инструментов анализа поведения (Hotjar, Google Analytics) для оценки эффективности навигации и вовлечённости.
   - *Документация и гайдлайны:* выпуск living-документации с рекомендациями по верстке, паттернам взаимодействия и лучшим практикам оформления контента. Регулярное обновление примеров в Figma и кодовых "reproduce"в Storybook.

3. **Фронтенд-развитие**

   - *Модернизация стека:* внедрение современного фреймворка (React, Vue или Svelte) для повышения повторного использования компонентов, упрощения управления состоянием и оптимизации рендеринга.
   - *Современные подходы к стилям:* переход на CSS-in-JS (Emotion, styled-components) или CSS Modules для локализации стилей, уменьшения конфликтов и возможности динамической генерации тем.
   - *Управление состоянием и данные:* интеграция Redux / Vuex / Pinia или применение более лёгких инструментов (zustand, recoil) для централизованного хранения состояния, кеширования запросов (RTK Query, Vue Query) и оптимизации перерендеринга.
   - *Производительность и PWA:* аудит bundle-размера с помощью Webpack Bundle Analyzer, внедрение lazy-loading для модулей и изображений, настройка Service Worker для поддержки офлайн-режима и push-уведомлений, подготовка к публикации как прогрессивного веб-приложения.
   - *Тестирование фронтенда:* расширение покрытия unit- и e2e-тестами с использованием Jest и Cypress (или Playwright), настройка тестовых сред в CI для проверки на реальных браузерах (BrowserStack, Puppeteer).

Список литературы

1. Bass L., Clements P., Kazman R. *Software Architecture in Practice* [Электронный ресурс]. - Boston: Addison-Wesley Professional, 2012. - 624 с. - URL: https://books.google.com/books?id=-II73rBDXCYC

2. Martin R.C. *Clean Architecture: A Craftsman's Guide to Software Structure and Design* [Электронный ресурс]. - Boston: Prentice Hall, 2017. - 432 с. - URL: https://books.google.com/books?id=uGE1DwAAQBAJ

3. Kleppmann M. *Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems* [Электронный ресурс]. - Sebastopol: O'Reilly Media, 2017. - 616 с. - URL: https://books.google.com/books?id=zFheDgAAQBAJ

4. Norman D.A. *The Design of Everyday Things* [Электронный ресурс]. - New York: Basic Books, 2013. - 368 с. - URL: https://books.google.com/books/about/The_Design_of_Everyday_Things.html?id=qBfRDQAAQBAJ

5. Gamma E., Helm R., Johnson R., Vlissides J. *Design Patterns: Elements of Reusable Object-Oriented Software* [Электронный ресурс]. - Boston: Addison-Wesley, 2005. - 395 с. - URL: https://books.google.com/books/about/Design_Patterns.html?id=iyIvGGp2550C

6. Krug S. *Don't Make Me Think!: A Common Sense Approach to Web Usability* [Электронный ресурс]. - Berkeley: New Riders, 2006. - 201 с. - URL: https://books.google.com/books/about/Don_t_Make_Me_Think.html?id=-PNSAAAAMAAJ

7. Haverbeke M. *Eloquent JavaScript: A Modern Introduction to Programming* [Электронный ресурс]. - San Francisco: No Starch Press, 2018. - 472 с. - URL: https://books.google.com/books/about/Eloquent_JavaScript_3rd_Edition.html?id=p1v6DwAAQBAJ

8. Duckett J. *HTML and CSS: Design and Build Websites* [Электронный ресурс]. - Hoboken: John Wiley & Sons, 2011. - 512 с. - URL: https://books.google.com/books/about/HTML_and_CSS.html?id=aGjaBTbT0o0C

9. Banks A., Porcello E. *Learning React: Functional Web Development with React and Redux* [Электронный ресурс]. - Sebastopol: O'Reilly Media, 2017. - 335 с. - URL: https://books.google.com/books/about/Learning_React.html?id=6bVZjwEACAAJ

Приложения

1. Исходный код: https://github.com/sh1ronchik/lightsb-service