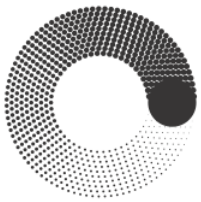
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ



МОСКОВСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Факультет Информационных технологий
Кафедра Информатики и информационных технологий

направление подготовки
09.03.02 «Информационные системы и технологии»

КУРСОВОЙ ПРОЕКТ

Дисциплина: Технологии прикладного программирования

Тема: Разработка веб-конструктора для создания персональных сайтов

Выполнил: студент группы 241-335

Кондрашов Михаил Иванович

(Фамилия И.О.)

Дата, подпись _____ _____
(Дата) (Подпись)

Проверил: _____
(Фамилия И.О., степень, звание)

Дата, подпись _____ _____
(Дата) (Подпись)

Замечания: _____

_____

_____

Москва

2025

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

МОСКОВСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Факультет информационных технологий
Кафедра Информатики и информационных технологий

_____ ***Утверждаю***

Зав. кафедрой Е.В. Булатников

« ___» _____ 2025 г.

ЗАДАНИЕ
на курсовое проектирование

Направление 09.03.02 «Информационные системы и технологии»

группа 241-335

дисциплина: ***Технологии прикладного программирования***

Студент Кондрашов Михаил Иванович

1. Тема проекта Разработка веб-конструктора для создания персональных сайтов

2. Срок сдачи студентом законченного проекта июнь 2025 года

3. Исходные данные к проекту разработка веб-сайта, позволяющего пользователям без навыков программирования создавать персональные сайты.
   Основной функционал: интуитивный редактор с визуальным интерфейсом, набор готовых шаблонов (визитка, портфолио, мини-блог), возможность кастомизации: текст, изображения, цветовая схема, экспорт результата в виде HTML-страницы.
   Технологический стек: HTML5, CSS3, JavaScript (React), Python (Flask)

4. Содержание расчетно-пояснительной записки (перечень подлежащих разработке вопросов) Введение (актуальность, цели и задачи), Анализ существующих решений, Выбор технологий разработки, Проектирование архитектуры системы, Реализация основных модулей (интерфейс редактора, система шаблонов, генератор HTML-кода

5. Перечень графического материала (с точным указанием обязательных чертежей) листинг кода, скриншоты этапов программирования, пример работы сайта

6. Литература и прочие материалы, рекомендуемые студенту для изучения Флэнаган Д. "JavaScript. Подробное руководство", Фримен Э. "Изучаем HTML, XHTML и CSS", Макфарланд Д. "JavaScript и jQuery. Исчерпывающее руководство", Документация React, Статьи по веб-дизайну и UX/UI

7. Промежуточные сроки исполнения _____
_____
_____
_____
_____
_____
_____
_____
8. Дополнительные указания _____
_____
_____
_____
_____
_____
_____
9. Дата выдачи задания февраль 2025 года

Руководитель проекта Забалуев Владислав Сергеевич

Задание принял студент Кондрашов Михаил Иванович

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

На сегодняшний день индустрия информационных технологий стремительно развивается и совершенствуется. Появляется множество программ и веб-сервисов, которые так или иначе способствуют упрощению жизни людей. Несомненно, самым прогрессирующим направлением в сфере информационных технологий считается разработка веб-страниц.

В современном мире сайт-визитка или портфолио становится неотъемлемой частью профессионального имиджа для специалистов различных сфер: фрилансеров, художников, IT-разработчиков и предпринимателей. Однако многие пользователи не обладают достаточными техническими навыками для самостоятельного создания веб-страниц, а существующие конструкторы (Tilda, Wix) часто избыточны по функционалу и сложны в освоении.

Целью проекта стала разработка прототипа WYSIWYG редактора для создания персональных сайтов с использованием библиотеки «React», в которой рядовой пользователь сможет создавать персональный сайт без использования навыков программирования. Основной акцент данного проекта был сделан на функциональность и реализацию системы, которая позволяет превращать созданный сайт в HTML код.

При заходе на сайт нас встречает первая страница с панелью авторизации, которая реализована через простейшие операции в виде ввода Email и Пароля. Для успешной реализации системы входа понадобилось создать файл page.tsx в директории src/app/auth/login. Основная задача данного файла заключается в отправке учетных данных на сервер, обработке ответа (успешный вход/ошибка), перенаправление после авторизации на основную страницу page.tsx.

Также была добавлена регистрация новый пользователей. Исходный код системы регистрации находится в файле page.tsx в директории src/app/auth/register. Данная страница тесно связана со страницей login1.

Однако она выполняет функции добавления новых данных на сервер. Также, в случае получения ответа о том, что пользователь уже зарегистрирован, сервер выводит Ошибку. В противном случае, перенаправляет в редактор.

Следующим шагом разработки стало создание панели администратора. Механизм доступа к административной панели реализован через последовательное нажатие клавиши V пять раз. Такой подход позволяет: быстро активировать панель без необходимости запоминать отдельный URL, обеспечить дополнительный уровень защиты, так как случайное открытие панели практически исключено, упростить процесс тестирования и отладки приложения в процессе разработки. Подобная реализация управления доступом к административной панели демонстрирует баланс между удобством использования и безопасностью системы. Основной функционал админ-панели позволяет создавать новых пользователей и редактировать пароли существующих. В дальнейшем планируется расширение функционала панели и добавление дополнительных методов аутентификации для обеспечения максимальной защиты административного интерфейса.

После успешного входа в систему, открывается главная страница с редактором, в котором мы можем наблюдать редактор сайтов. Основой редактора стал компонентный подход на «React», позволяющий пользователям без технической подготовки создавать профессиональные веб-страницы. Система поддерживает восемь базовых типов элементов, включая заголовки, текстовые блоки, изображения и адаптивные контейнеры, каждый из которых автоматически генерирует оптимизированный HTML-код. Особое внимание уделено механизму управления состоянием приложения – реализована история изменений с возможностью отмены и повтора до 50 последних действий, что значительно упрощает процесс редактирования. Глобальные элементы, такие как шапка и подвал сайта, автоматически синхронизируются между всеми страницами проекта. Редактор адаптирован для работы на различных устройствах благодаря продуманной системе выдвижных панелей в мобильной версии.

Ключевой особенностью стала интеллектуальная система контейнеров, поддерживающая автоматическую верстку в две или три колонки, а также динамическая генерация навигационного меню на основе структуры сайта. Редактор мгновенно отображает все изменения, позволяя сразу видеть результат без необходимости перезагрузки.

В результате была создана функциональная демонстрационная версия WYSIWYG редактора, включающая в себя все его ключевые системы:

- Добавление/удаление элементов через интерфейс
- Редактирование стилей и содержимого страниц
- Управление структурой сайта (страницы, навигация)
- Работа с историей изменений (undo/redo)
- Работа с админ-панелью

Рассмотрим более подробно процесс создания редактора.

1. Аналитическая часть

1.1 Выбор стека разработки

В процессе подготовки к реализации курсового проекта передо мной встал важный выбор технологического стека. Существовали две основные альтернативы: классический набор HTML, CSS, JavaScript в сочетании с Flask на бэкенде, либо современный стек на базе React , TypeScript и Next.js

После длительного и тщательного анализа требований к проекту, а также скрупулёзной оценки своих навыков и возможностей, я принял решение использовать React и TypeScript. Несмотря на то, что этот выбор предполагает более крутую кривую обучения и требует значительных усилий для освоения, он обусловлен множеством ключевых факторов.

Во-первых, React предоставляет мощный инструментарий для создания интерактивных пользовательских интерфейсов, что особенно важно для современного веб-приложения. Этот фреймворк позволяет создавать компоненты, которые легко переиспользовать и модифицировать, что значительно ускоряет процесс разработки и поддержки кода. Более того, компонентный подход способствует созданию чистого и структурированного кода, что особенно важно при работе в команде.

Во-вторых, TypeScript добавляет статическую типизацию в JavaScript, что значительно упрощает разработку и поддержку кода, особенно в условиях ограниченного времени на выполнение проекта. Строгая типизация позволяет выявлять ошибки на этапе компиляции, что существенно снижает количество багов и делает код более понятным и структурированным. Это также способствует более быстрой отладке и улучшению качества кода.

Этот стек также предоставляет широкие возможности для масштабирования проекта в будущем. React позволяет создавать сложные пользовательские интерфейсы с высокой производительностью благодаря использованию Virtual DOM, что обеспечивает быструю и эффективную работу приложения даже при больших объёмах данных. TypeScript, в свою

очередь, обеспечивает строгую типизацию и удобство работы с кодом, что делает его идеальным выбором для крупных и сложных проектов.

Выбрав React и TypeScript, я также учитывал долгосрочные перспективы. Этот технологический стек позволяет легко находить документацию и готовые решения, что значительно упрощает процесс разработки. Кроме того, опыт работы с современными технологиями повышает мою квалификацию и делает моё портфолио более актуальным на рынке труда. Это особенно важно в условиях стремительного развития технологий и постоянного появления новых решений.

Для успешной реализации проекта на выбранном стеке я планирую не только изучить основы React и TypeScript, но и глубоко освоить современные инструменты разработки, такие как Webpack, Babel и другие. Я намерен создать масштабируемую архитектуру проекта, которая будет учитывать все современные требования и лучшие практики разработки. Особое внимание будет уделено внедрению системы управления состоянием приложения и оптимизации производительности.

В дальнейшем, я планирую активно использовать систему тестирования для обеспечения качества кода и надёжности приложения. Это позволит минимизировать количество ошибок и обеспечить стабильную работу проекта в различных условиях.

Несмотря на то, что классический стек мог бы обеспечить более быстрое начало работы, я решил инвестировать время в изучение более современного и перспективного технологического стека. Это решение обусловлено не только стремлением создать качественный продукт, соответствующий современным стандартам разработки, но и желанием получить ценный опыт работы с технологиями, которые востребованы на рынке труда и активно используются в современных проектах.

Прежде чем окончательно остановиться на стеке React и TypeScript, я решил попробовать себя в реализации проекта с использованием классического набора технологий: HTML, CSS и JavaScript в связке с Flask.

Несмотря на кажущуюся простоту и доступность этого стека, я столкнулся с рядом существенных проблем.

Во-первых, при попытке реализовать современный интерфейс с использованием только HTML и CSS, я столкнулся с ограничениями в плане гибкости и масштабируемости. Создание сложных интерактивных элементов требовало написания огромного количества кода, который было сложно поддерживать и модифицировать. JavaScript в этом стеке часто оказывался «за бортом», выполняя лишь вспомогательные функции, что существенно усложняло разработку клиентской части приложения. Более того, при попытке интеграции с бэкендом через Flask я столкнулся с проблемой отсутствия чёткой структуры и организации кода, что приводило к путанице и ошибкам.

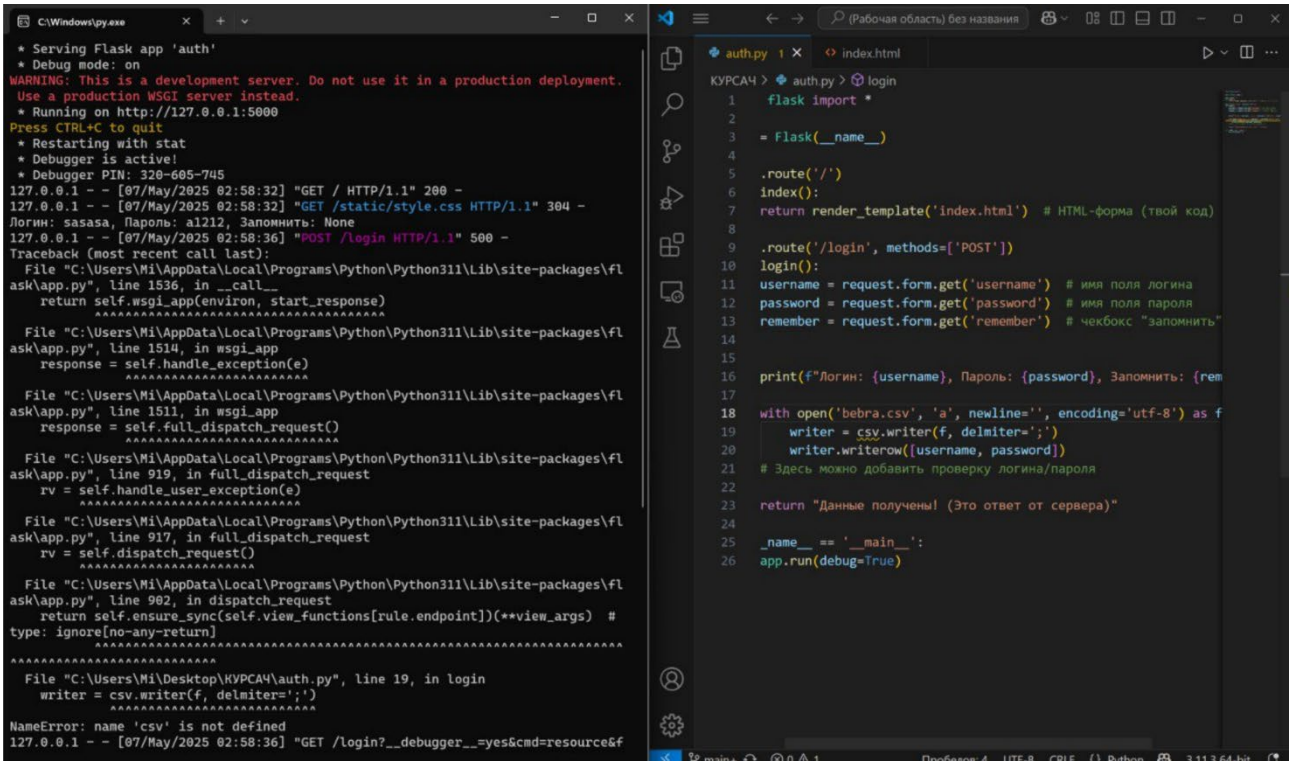


Рисунок 1. При попытке усложнения кода с использованием Flask, появлялись ошибки со стороны сервера.

В процессе работы с этим стеком я всё больше убеждался, что для реализации полноценного современного веб-приложения он не подходит. Отсутствие строгой типизации и необходимость постоянного ручного управления состоянием приложения делали процесс разработки трудоёмким и неэффективным. Каждый новый функционал требовал значительных временных затрат на реализацию и отладку, а поддержка кода становилась всё более сложной.

Именно поэтому я принял решение перейти на стек React и TypeScript. Этот выбор позволил мне не только значительно упростить процесс разработки, но и получить более структурированный и поддерживаемый код. React с его компонентным подходом и TypeScript со строгой типизацией дали мне тот уровень гибкости и контроля, который был недостижим при использовании классического стека. Это позволило мне сосредоточиться на создании качественного продукта, не отвлекаясь на решение технических проблем, связанных с организацией кода и управлением состоянием приложения.

1.2 Исследование аналогичных сервисов

После выбора стека, передо мной возник важный вопрос: существуют ли специализированные WYSIWYG-редакторы для создания сайтов? Если да, то какие они имеют преимущества и недостатки? Данный вопрос очень важен, так как создать качественный продукт можно, только проанализировав данную информацию.

После проведённого исследования было выяснено, что такие инструменты действительно существуют и активно используются как профессионалами, так и новичками. Среди наиболее популярных WYSIWYG-редакторов можно отметить Tilda который предлагает мощный визуальный конструктор для создания интерактивных макетов. Также стоит упомянуть Froala Editor — многофункциональный инструмент с поддержкой

HTML и возможностью создания адаптивных дизайнов. Не менее известен Turbologo — онлайн-сервис, позволяющий быстро создавать сайты с помощью визуального конструктора.

Изучив преимущества и недостатки этих инструментов, было выделено несколько ключевых моментов. WYSIWYG-редакторы обладают интуитивно понятным интерфейсом, что позволяет создавать сайты без глубоких знаний программирования. Они предоставляют возможность визуального редактирования в реальном времени, что значительно ускоряет процесс разработки. Кроме того, такие редакторы обычно включают готовые шаблоны и стили, что существенно упрощает процесс создания дизайна.

Однако у WYSIWYG-редакторов есть и свои недостатки. Один из главных — ограниченная кастомизация: сложно реализовать уникальные дизайнерские решения, которые не предусмотрены в интерфейсе редактора.

Также существует зависимость от конкретного инструмента: при переходе на другую платформу могут возникнуть сложности с переносом проекта. Некоторые редакторы могут работать медленнее при создании сложных проектов, а также имеют ограничения в поддержке интерактивных элементов, которые проще реализовать через код.

На этом этапе разработки было осознано ключевое преимущество моего сервиса — возможность экспорта проекта в готовый HTML-код. Это решение имеет несколько важных преимуществ.

Во-первых, это доступность для начинающих разработчиков, которые смогут легко интегрировать созданный макет в свой проект.

Во-вторых, универсальность: HTML-код можно использовать на любой платформе и в любом проекте.

В-третьих, гибкость доработки: возможность дальнейшей оптимизации и адаптации кода под конкретные требования проекта.

Таким образом, мой сервис будет сочетать простоту визуального создания сайтов. Проанализировав всю информацию, было принято решение приступить к созданию прототипа.

1.3 Создание прототипа в Figma

Этот этап считается критически важным, поскольку он позволяет иметь представление о конечном результате и избежать возможных недоразумений в процессе работы.

Для реализации этой задачи был выбран мощный инструмент в области дизайна интерфейсов — редактор Figma. Решение использовать именно этот инструмент было обусловлено его широкими возможностями для создания прототипов и макетов различной сложности.

Опираясь исключительно на собственное творческое видение, я приступил к созданию дизайна без использования каких-либо референсов или готовых шаблонов. Отсутствие внешних ориентиров и строгих ограничений позволило максимально реализовать свой творческий потенциал.

Несмотря на отсутствие чётких ориентиров и предварительных референсов, мне удалось создать прототип, который продемонстрировал необходимую функциональность и визуальную целостность. В процессе работы возникали определённые моменты неопределённости, требовавшие корректировок, однако именно этот экспериментальный подход позволил найти оригинальные визуальные решения.

В результате был разработан дизайн, который, сохраняя необходимую эргономичность и функциональность, отличается высокой степенью оригинальности. Этот опыт наглядно продемонстрировал, что творческий подход и готовность к экспериментам могут привести к созданию уникальных визуальных решений даже при отсутствии готовых примеров и шаблонов

Хотя данный прототип дизайна не является полностью завершённым, он отражает приблизительное расположение ключевых элементов, которые будут включены в финальную версию сайта. На этом этапе основная цель —

продемонстрировать общую структуру и концепцию расположения основных компонентов интерфейса.

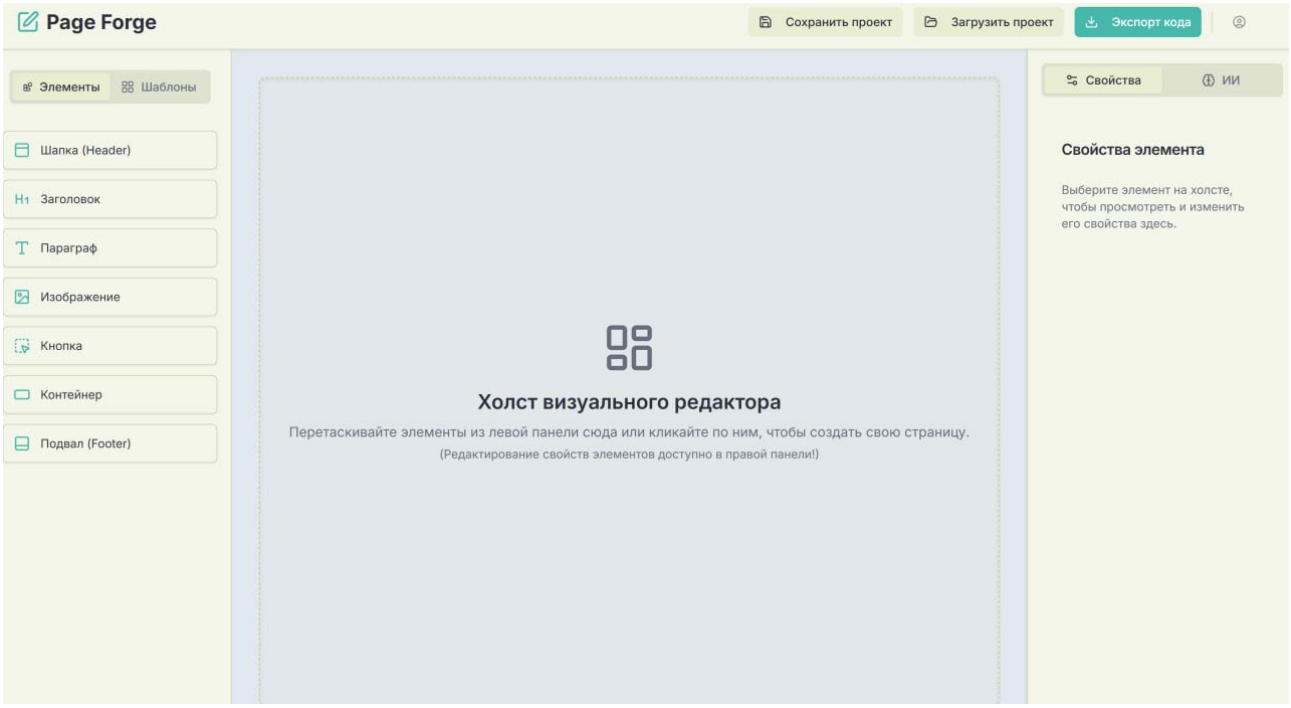


Рисунок 2. Прототип Figma

После проработки идеи проекта и создания прототипа, мною было принято решение проанализировать всю теоретическую информацию по выбранному стеку разработки.

2. Теоретическая часть

2.1 Особенности работы на React

Прежде чем говорить об особенностях библиотеки React, следует разобраться: в чем её основное отличие от других библиотек?

React — это библиотека JavaScript для создания пользовательских интерфейсов с помощью модульных и повторно используемых компонентов.

Компонентный подход — фундамент React. Интерфейс разбивается на независимые переиспользуемые компоненты, каждый из которых управляет собственным состоянием, логикой и отображением. Это ускоряет разработку, упрощает тестирование и поддержку кода.

React использует специальный синтаксис JSX (JavaScript XML), который позволяет описывать структуру интерфейса в виде, приближённом к HTML. Это делает разработку более интуитивно понятной. Одной из главных архитектурных особенностей React является использование Virtual DOM — виртуального дерева элементов в памяти. Вместо непосредственных изменений реального DOM, React сначала сравнивает старую и новую версии виртуального дерева, после чего минимально обновляет реальный DOM. Это повышает производительность, особенно при частых обновлениях интерфейса.

Кроме того, React использует однонаправленный поток данных, при котором информация передаётся от родителя к дочернему компоненту через свойства (props). Это делает структуру приложения предсказуемой и логически целостной.

Современные React-приложения в основном строятся на функциональных компонентах, в которых активно используются хуки (hooks) — специальные функции, позволяющие работать с состоянием и жизненным циклом компонентов. Наиболее часто применяются useState (для хранения состояния) и useEffect (для выполнения побочных эффектов, таких как HTTP-запросы или подписки).

Также в React активно используется собственное состояние компонентов (state) — это данные, которые определяют текущее поведение и отображение компонента. При изменении состояния React автоматически обновляет интерфейс. В отличие от props, которые передаются "снаружи", state управляется внутри самого компонента.

Ещё одной важной частью React является жизненный цикл компонентов. Ранее в классовых компонентах использовались специальные методы (componentDidMount, componentDidUpdate, componentWillUnmount), которые реагировали на различные стадии жизни компонента. В функциональных компонентах с появлением хуков эти методы заменяются вызовами useEffect.

Возникает закономерный вопрос: зачем подключать TypeScript, если уже есть React? Разве нельзя обойтись только библиотекой React? Прежде чем дать ответ на этот вопрос, важно понять, что такое TypeScript и какие задачи он решает.

2.2 TypeScript и его преимущества

TypeScript — это строго типизированное надмножество JavaScript, разработанное компанией Microsoft. Оно компилируется в чистый JavaScript, но добавляет мощную систему типов, позволяющую разработчикам явно указывать, с какими данными работает код. Это даёт значительные преимущества, среди которых:

- Раннее выявление ошибок — уже на этапе написания кода среда разработки указывает на типовые несоответствия;
- Автодополнение и подсказки — редакторы (например, VS Code) могут подсказывать свойства и методы;
- Документирование типов — структура данных описывается явно в коде, упрощая понимание и поддержку;
- Типобезопасность — снижает риск логических ошибок при передаче данных между компонентами;

В рамках курсового проекта я использовал React в связке с TypeScript. Каждый компонент написан с поддержкой JSX-разметки и типовой аннотацией. Например, в файле pages/index.tsx применена типизация props, что помогает избежать некорректного использования данных, передаваемых в компоненты.

Также в структуре проекта используются файлы с расширением .ts, например, для описания интерфейсов и утилит. Это позволяет вынести типы в отдельные модули и использовать их повторно — один из принципов чистой архитектуры.

К примеру, в моём редакторе сайтов используется файл export.ts, отвечающий за экспорт HTML кода в архив, который содержащий все файлы сайта, который пользователь создаёт через мой сервис. Листинг кода export.ts указан в приложении 1.

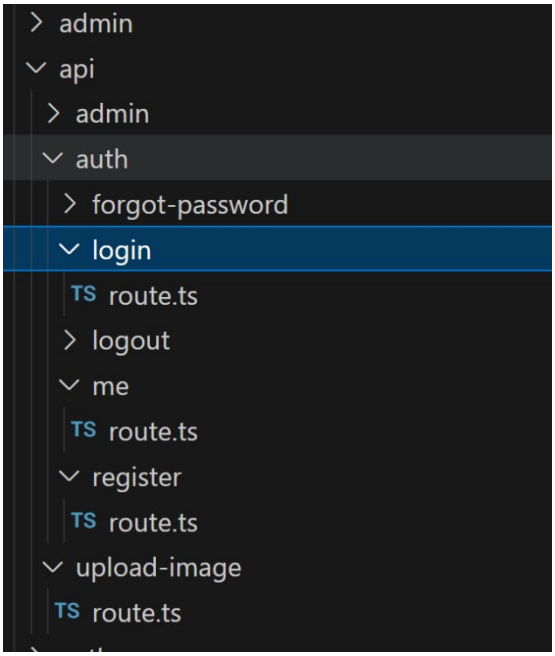


Рисунок 3. Пример файлов .ts в исходном коде проекта

Можно ли обойтись без TypeScript? Да, можно. React сам по себе не требует TypeScript и прекрасно работает с обычным JavaScript. Однако по мере развития моего проекта сложность взаимодействия между компонентами возросла. Именно поэтому все операции были задействованы с использованием TypeScript

Помимо этого, коды TypeScript могут использоваться для написания серверной части. Так, в моём проекте данные коды используются для авторизации, регистрации и прочих API-маршрутов, которые необходимы для моего редактора веб-страниц. Ссылка на репозиторий моего проекта указана в приложении 2.

2.3 Next.js и серверная часть

Next.js — это фреймворк для React, созданный компанией Vercel, который добавляет в экосистему React поддержку серверного рендеринга (SSR — Server-Side Rendering), статической генерации страниц (SSG — Static Site Generation), маршрутизации на основе файловой системы, а также возможность писать backend-логику непосредственно в React-проекте.

К ключевым возможностям Next.js относят:

- Файловая маршрутизация — каждый .tsx-файл в папке pages/ автоматически становится маршрутом сайта;
- SSR и SSG — страницы могут генерироваться заранее или при каждом запросе на сервере, что улучшает производительность и SEO;
- API Routes — можно создавать файлы в папке pages/api/, которые становятся полноценными HTTP эндпоинтами, исполняемыми на сервере;
- Среда выполнения Node.js — позволяет обрабатывать запросы к базе данных, авторизацию, отправку писем и др.;
- Поддержка middlewares и серверной логики — можно встроить полноценную бизнес-логику прямо в Next.js-приложение.

В моем курсовом проекте предусмотрена серверная часть, реализованная средствами Next.js. Это достигается за счет файлов с префиксом getServerSideProps() — они позволяют загружать данные на сервере при каждом запросе и передавать их в компонент в виде props.

Помимо этого, я использую API-обработчики — например, в директории pages/api/ расписаны серверные функции для обработки POST/GET-запросов, что особенно удобно при создании форм, авторизации или интеграции с внешними сервисами. Такой подход позволяет создавать полноценные веб-приложения, где одна часть отвечает за отображение интерфейса, а другая — за обработку данных, и всё это — в рамках одной кодовой базы.

Таким образом, использование Next.js в проекте позволило объединить клиентскую и серверную части веб-приложения в единую архитектуру, упростив разработку и поддержку кода. Благодаря встроенной маршрутизации, возможностям серверного рендеринга и поддержки API-обработчиков, Next.js обеспечил высокую производительность, удобную организацию страниц и лёгкую интеграцию с внешними сервисами.

Рисунок 4. Запуск серверной части Next.js через консоль

3. Практическая часть

3.1 Разработка первой версии конструктора

В пункте 1.3 данной работы я уже указывал, что на самом начальном этапе разработки мной был создан прототип в редакторе дизайнов Figma.

Проанализировав всю необходимую информацию, я приступил к верстке своего проекта используя технологии библиотеки React. В отношении прототипа я внёс некоторые изменения, а именно: убрал кнопку ИИ (так как я не смог реализовать данную возможность, однако оставил задел на будущее обновление в файлах проекта), а также изменил название с Page Force на PagesMi, так как проект Page Force уже существовал на Github у другого пользователя.

Для систематизации разработки UI я использовал принцип атомарного дизайна, разделив компоненты на 5 уровней:

1. Атомы (Atoms) - Базовые элементы:
   - Кнопки: Button, IconButton
   - Поля ввода: Input, Textarea
   - Текстовые элементы: Text, Heading
2. Молекулы (Molecules) - Простые комбинации атомов:
   - Поисковая строка: SearchBar (Input + Button)
   - Карточка элемента: ElementCard (Heading + Text + IconButton)
3. Организмы (Organisms) - Сложные блоки интерфейса:
   - Панель инструментов: ToolbarPanel (ElementCard × N + SearchBar)
   - Инспектор свойств: PropertyInspector (Input, ColorPicker, Select)
4. Шаблоны (Templates) - Макеты страниц:
   - Шаблон редактора: EditorLayout (ToolbarPanel + CanvasArea)
5. Страницы (Pages) - Финальные страницы.

В процессе разработки были внедрены следующие ключевые улучшения по сравнению с Figma-прототипом. Во-первых, реализована динамическая система компонентов, где статичные контейнеры заменены на динамические элементы с возможностью модификации в реальном времени. Во-вторых, добавлены интерактивные состояния для всех элементов интерфейса, включая hover и activation. Также был осуществлен переход на адаптивную верстку с использованием Flexbox и Grid вместо фиксированных размеров. Кроме того, внедрена централизованная система управления стилями через кастомные классы Tailwind, что позволило унифицировать цветовую палитру.

Проект построен на принципах централизованного управления состоянием с четким разделением ответственности между компонентами. Ключевым элементом архитектуры является главный контроллер, который координирует работу всех модулей системы.

Центральный контроллер, расположенный в файле page.tsx (директория src/app/), выступает в роли управляющего модуля приложения. Он хранит полное состояние проекта в объекте siteData, содержащем иерархическую структуру всех страниц сайта, деревья элементов для каждой страницы, стили и контент всех элементов. Контроллер также управляет историей действий через массив history с поддержкой функций undo/redo и реализует более 15 специализированных функций-обработчиков для работы с элементами интерфейса.

Такая архитектура обеспечивает высокую предсказуемость поведения системы и упрощает дальнейшее масштабирование функционала (например для добавления нейросетей).

3.2 Разработка скрипта для экспорта HTML кода

После разработки основной части самого редактора было принято решение приступить к созданию механизма экспорта. Основная задача моего редактора заключалась в том, чтобы преобразовать структуру созданной страницы, заданную в виде компонентов (CanvasElement), в валидный HTML-код, пригодный для экспорта и дальнейшего использования в ZIP-архиве.

Казалось, что задача нереализуема. Однако внимательно изучив структуру кода, я заметил некоторые закономерности между файлами .tsx и .html.

Каждый CanvasElement содержал данные, описывающие его тип, стили и вложенные элементы. Сопоставив это с финальным DOM-деревом, стало ясно, что можно построить алгоритм, который будет проходить по всей структуре страницы, рекурсивно обрабатывать компоненты, и на выходе формировать валидную HTML-разметку.

Таким образом, я расписал, что должен делать мой код:

- Обрабатывать каждый CanvasElement;
- Преобразовавать его в HTML-тег с соответствующими атрибутами и стилями;
- Рекурсивно обходить вложенные элементы;
- Оборачивать всю разметку в базовую HTML-структуру (doctype, <html>, <head>, <body> и т.д.).

Данная технология не нова, и уже применяется в редакторе Pixso. Однако, он не выдавал полноценный код на Html. После всех доработок, код экспорта начал работать полноценно и выдавать практически идентичную картину.

3.3 Добавление кнопок отмены действий

Весьма длительное время (около двух недель) я пробовал способ создания кнопок возврата действий, как это выполнено в любых текстовых и графических редакторах.

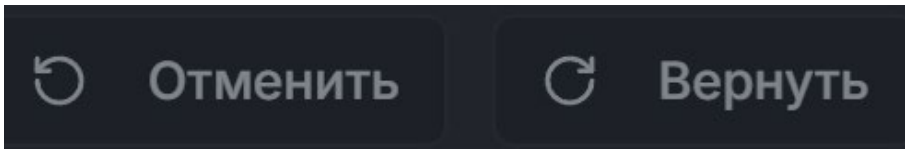


Рисунок 4. Так стали выглядеть кнопки после их создания

Система отмены и повтора действий (Undo/Redo) была реализована с использованием концепции "слепков состояния". Каждое значимое действие пользователя автоматически сохраняет полную копию текущего состояния проекта в специальном хранилище истории. Это позволяет в любой момент вернуться к предыдущим версиям проекта.

В основе системы лежит механизм, который создает новый "слепок" состояния после каждого изменения. Эти слепки хранятся в массиве, где указатель текущей позиции отмечает актуальное состояние проекта. Когда пользователь нажимает кнопку "Отменить", система смещает указатель на одну позицию назад и восстанавливает предыдущее состояние. Кнопка "Повторить" работает аналогично, но двигает указатель вперед по истории действий. Для предотвращения чрезмерного использования памяти установлен лимит в 100 последних действий - при его превышении самые старые записи автоматически удаляются.

Кнопки управления расположены на верхней панели инструментов и визуально оформлены для интуитивного понимания. Кнопка "Отменить" отображается со стрелкой влево, а "Повторить" - со стрелкой вправо. В неактивном состоянии (когда нет действий для отмены/повтора) кнопки становятся полупрозрачными.

3.4 Добавление панели администратора

Спустя некоторое время мне в голову пришла идея создания панели администратора, которая будет активироваться при нажатии секретной комбинации. Данный подход способен практически исключить возможность взлома панели, а также упростит её реализацию.

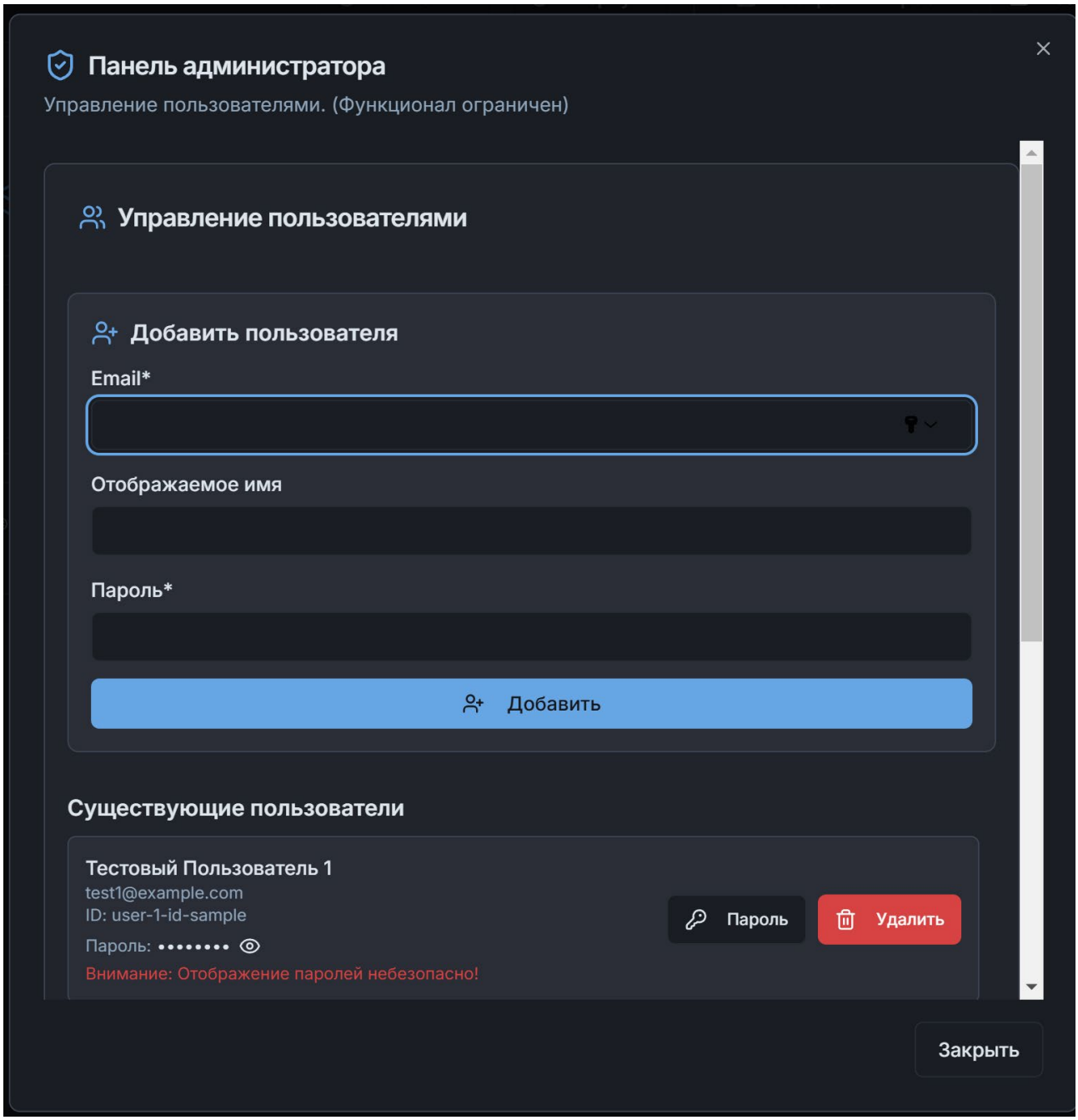


Рисунок 5. Панель администратора

В модальном окне реализован комплексный инструментарий для управления платформой. Функционал управления пользователями позволяет просматривать полный список зарегистрированных аккаунтов, фильтровать, выполнять добавление или удаление пользователей и сбрасывать их пароли.

Интеграция с бэкендом реализована через RESTful API с единой системой обработки ошибок и кэшированием повторяющихся запросов. Архитектура модульная - каждый компонент работает автономно, но интегрирован в общую систему через централизованное хранилище состояний и единую службу аутентификации.

Хочется отметить, что это только первая версия панели администратора. В случае дальнейшего развития проекта, в ней будет возможность реализовать функции редактирования сохраненных проектов у каждого пользователя.

ЗАКЛЮЧЕНИЕ

В рамках курсового проекта была разработана система визуального редактирования веб-страниц, представляющая собой интерактивный редактор с возможностью добавления, изменения и экспорта компонентов в виде валидного HTML-кода. Проект охватывает полный цикл создания редактора: от проектирования интерфейса до реализации экспортного механизма с упаковкой результата в ZIP-архив.

Главной особенностью проекта стала реализация пользовательского интерфейса с возможностью динамического взаимодействия с элементами страницы. Компонентный подход и использование React/TypeScript позволили эффективно структурировать код и добиться масштабируемости. Особое внимание было уделено преобразованию структуры компонентов (CanvasElement) в финальный HTML-документ, пригодный для последующего использования.

Проект подтвердил важность модульного подхода: логика компонентов, панели инструментов, системы экспорта и визуализации были реализованы независимо, что значительно упростило разработку и тестирование. Благодаря использованию современных веб-технологий проект продемонстрировал устойчивость, расширяемость и готовность к дальнейшему развитию — например, добавлению шаблонов, сохранению состояния, подключению CSS-фреймворков и т. д.

Таким образом, цели и задачи курсовой работы были успешно достигнуты. Полученный результат может быть использован как основа для более крупного конструктора сайтов или образовательного инструмента для начинающих веб-разработчиков.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Документация React — URL: https://reactjs.org (дата обращения: 11.06.2025)
2. Хавратова А.Ю., «Основы веб-разработки» — М.: БХВ-Петербург, 2021. (дата обращения: 04.05.2025)
3. Флэнаган Д. «JavaScript. Подробное руководство» — СПб.: Питер, 2022. (дата обращения: 01.06.2025)
4. Меган Дэй, «Современная веб-разработка с React и Redux», Хьюстон, 2020. (дата обращения: 16.06.2025)
5. Next.js Documentation - URL: https://nextjs.org/docs (дата обращения: 11.06.2025).
6. •TypeScript Handbook - URL: https://www.typescriptlang.org/docs/ (дата обращения: 17.05.2025)
7. Pixso – URL: https://pixso.net/ (дата обращения: 4.05.2025)

ПРИЛОЖЕНИЯ

```
import type { SiteData, SitePage, CanvasElement, ElementType } from '@/types/canvas-element';
import JSZip from 'jszip';
import type React from 'react';

function toKebabCase(str: string): string {
  if (str.includes('-')) {
      if (str.startsWith('--') || str === str.toLowerCase()) return str;
  }
  return str.replace(/([A-Z])/g, (match) => `-${match.toLowerCase()}`).replace(/^-/, '');
}


function cssPropertiesToInlineStyle(styles: React.CSSProperties | undefined): string {
  if (!styles) return '';
  return Object.entries(styles)
    .map(([prop, value]) => {
      if (value === undefined || value === null || value === '') return '';

      let kebabProp = prop.startsWith('--') ? prop : toKebabCase(prop);

      if (typeof value === 'string' && (value.includes('var(') || value.includes('hsl(var('))) {
        return `${kebabProp}: ${value.replace(/"/g, "'")};`;
      }

      if (typeof value === 'number' &&
          !['opacity', 'zIndex', 'fontWeight', 'lineHeight', 'flex', 'flexGrow', 'flexShrink', 'order'].includes(prop) &&
          !kebabProp.startsWith('--')
         ) {
        return `${kebabProp}: ${value}px;`;
      }
      return `${kebabProp}: ${String(value).replace(/"/g, "'")};`;
    })
    .join(' ');
}

function getFilenameFromLocalPath(localPath: string): string {
  return localPath.substring(localPath.lastIndexOf('/') + 1);
}

function getFilenameFromLocalPath(localPath: string): string {
  return localPath.substring(localPath.lastIndexOf('/') + 1);
}

function renderElementToHtml(element: CanvasElement, isExporting: boolean): string {
  const styleString = cssPropertiesToInlineStyle(element.styles);
  const attributes = element.props ? Object.entries(element.props)
    .map(([key, value]) => {
        if (typeof value === 'boolean') {
            return value ? toKebabCase(key) : '';
        }
        if (isExporting && key === 'data-ai-hint') return '';
        if (isExporting && key === 'data-is-child-block') return '';
        if (isExporting && key === 'data-layout-type' && (element.type !== 'Container' || !value)) return '';

        return `${toKebabCase(key)}="${String(value).replace(/"/g, '&quot;')}"`;
```

```
    })
    .filter(Boolean)
    .join(' ') : '';

  let childrenHtml = '';
  if (element.type === 'Container' && element.children && element.children.length > 0) {
    childrenHtml = element.children.map(child => renderElementToHtml(child, isExporting)).join('\n');
  } else if (element.type === 'Container' && (!element.children || element.children.length === 0)) {
    // childrenHtml = '<!-- Empty Container -->';
  }


  switch (element.type) {
    case 'Header':
      return `<header style="${styleString}" ${attributes}>${element.content || ''}</header>`;
    case 'Footer':
      return `<footer style="${styleString}" ${attributes}>${element.content || ''}</footer>`;
    case 'Heading1':
      return `<h1 style="${styleString}" ${attributes}>${element.content || ''}</h1>`;
    case 'Heading2':
      return `<h2 style="${styleString}" ${attributes}>${element.content || ''}</h2>`;
    case 'Heading3':
      return `<h3 style="${styleString}" ${attributes}>${element.content || ''}</h3>`;
    case 'Paragraph':
      return `<p style="${styleString}" ${attributes}>${element.content || ''}</p>`;
    case 'Button':
      return `<button style="${styleString}" ${attributes}>${element.content || ''}</button>`;
    case 'Image':
      let imageSrc = element.src || '';
      if (isExporting && imageSrc.startsWith('/secret1/')) {
        const filename = getFilenameFromLocalPath(imageSrc);
        imageSrc = `images/${filename}`;
      }
      return `<img src="${imageSrc}" alt="${element.alt || ''}" style="${styleString}" ${attributes} />`;
    case 'Container':
      return `<div style="${styleString}" ${attributes}>${childrenHtml}</div>`;
    default:
      return `<!-- Unknown element type: ${element.type} -->`;
  }
}
```

```
  const elementsHtml = elementsForExport.map(el => renderElementToHtml(el, true)).join('\n        ');
  const globalFontFamily = page.canvasStyles?.fontFamily || "Inter, sans-serif";

  const embeddedStyles = `
        @import url('https://fonts.googleapis.com/css2?family=Inter:wght@400;500;600;700&display=swap');
        @import url('https://fonts.googleapis.com/css2?family=Roboto:wght@400;500;700&display=swap');
        @import url('https://fonts.googleapis.com/css2?family=Open+Sans:wght@400;600;700&display=swap');
        @import url('https://fonts.googleapis.com/css2?family=Montserrat:wght@400;500;600;700&display=swap');
        @import url('https://fonts.googleapis.com/css2?family=Lato:wght@400;700&display=swap');
        @import url('https://fonts.googleapis.com/css2?family=Poppins:wght@400;500;600;700&display=swap');
        @import url('https://fonts.googleapis.com/css2?family=Nunito:wght@400;600;700&display=swap');

        /* Dark Theme Variables from globals.css */
        :root {
```

Приложение 1. Листинги скрипта export.ts

Приложение 2. [Ссылка на рабочий проект в Github (кликабельно)]