Государственное бюджетное общеобразовательное учреждение

лицей № 369 имени Героя Российской Федерации А.Н.Жихарева

Красносельского района Санкт-Петербурга

Индивидуальный проект по теме

**«Разработка веб-движка по созданию сайтов»**

Тип проекта: информационно-прикладной

Учебная дисциплина: информатика

Автор работы: Лакотко Иван Романович

Класс: 10 А

Руководитель работы: Ильина Ксения Алексеевна

Должность: учитель информатики

Работа допущена к защите «\_\_\_»\_\_\_\_\_\_\_2025 г

Подпись руководителя проекта\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Санкт-Петербург

2025

Оглавление

[Введение 3](#_Toc192441836)

[Глава 1 4](#_Toc192441837)

[1.1 Анализ похожих проектов и выявление их недостатков 4](#_Toc192441838)

[1.2 Системы управления содержимым 5](#_Toc192441839)

[1.3 Архитектура проекта 6](#_Toc192441840)

[1.4 Выбор инструментов для разработки 7](#_Toc192441841)

[Глава 2 10](#_Toc192441842)

[2.1 Клиентская часть 10](#_Toc192441843)

[2.1.1 Структура HTML страниц сайта 10](#_Toc192441844)

[2.1.2 Дизайн 10](#_Toc192441845)

[2.1.3 CSS cтилизация 11](#_Toc192441846)

[2.1.4 JavaScript программирование 11](#_Toc192441847)

[2.2 Серверная часть 12](#_Toc192441848)

[2.2.1 FastAPI 12](#_Toc192441849)

[2.2.2 База данных 13](#_Toc192441850)

[2.2.3 Создание, хранение и изменение пользовательских сайтов. 13](#_Toc192441851)

[Заключение 15](#_Toc192441852)

[Литература 16](#_Toc192441853)

[Приложение 17](#_Toc192441854)

# Введение

**Актуальность.** В современном мире Интернет все активнее внедряется в сферу деятельности нашего общества, он стал важным и незаменимым аспектом жизни каждого. Его появление вывело все сферы нашей жизни на абсолютно новый уровень. Но проблема в том, что не каждый может в полной мере использовать возможности интернета, например создать собственный сайт (портфолио или сайт для предложения собственных услуг). Ведь это требует большого количества времени и навыка в некоторых языках программирования. Решая эту проблему, я поставил себе цель - разработать сервис, позволяющий обычным пользователям Интернета создавать собственные сайты быстро и без программирования. Продуктом будет непосредственно сам веб-сервис, размещенный в интернете в свободном доступе.

**Цель.** Создать веб-сервис, с помощью которого можно будет создавать сайты быстро и без программирования.

**Задачи:**

1. Изучение и анализ похожих сервисов, выявление их недостатков.
2. Изучение архитектуры веб-движков (CMS).
3. Изучение необходимых фреймворков для написания собственного сервиса и выбор инструментов для разработки.
4. Изучение взаимодействий языков программирования и API.
5. Изучение работы с базами данных.
6. Создание веб-движка по созданию сайтов.

# Глава 1

## 1.1 Анализ похожих проектов и выявление их недостатков

На начальной стадии разработки проекта я обратился к существующим аналогичным приложениям. Самые известные из них:

* Тильда // Tilda. URL: [https://tilda.cc](https://tilda.cc/)
* Викс // Wix URL: https://www.wix.com
* Вордпресс (CMS) // WordPress. URL: [https://ru.wordpress.org](https://ru.wordpress.org/)
* Джумла (CMS) // Joomla. URL: [https://www.joomla.org](https://www.joomla.org/)

Изучив эти инструменты, я смог выявить следующие недостатков, которые бы мог устранить в собственном проекте:

* **Сложность интерфейса** (*см Иллюстрации 1, 2*)*.* Я понял, что для обычного пользователя интерфейс каждого сервиса весьма непонятен и отпугивает своей сложностью. Поэтому я решил сделать акцент в своей разработке именно на простоте и удобстве в использовании.
* **Платный контент** (*см Иллюстрации 3, 4*). Большинство полезных функций в данных приложениях являются платными. В тильде бесплатно можно создать только один сайт. А использование всех функций и возможностей вордпресса обойдется вам в 25.000 $ в год. В моем же проекте будут только самые полезные функции, которыми можно будет пользоваться бесплатно.

## 1.2 Системы управления содержимым

Результатом моего проекта будет веб-движок по созданию сайтов (иначе ­система управления содержимым). Существует два вида систем управления содержимым:

1. CMS ([англ.](https://ru.wikipedia.org/wiki/Английский_язык) Content management system, CMS, система управления контентом) — [информационная система](https://ru.wikipedia.org/wiki/Информационная_система) или [компьютерная программа](https://ru.wikipedia.org/wiki/Компьютерная_программа), используемая для обеспечения и организации совместного процесса создания, редактирования и управления содержимым (контентом).
2. Конструктор сайтов (англ. site builder) — программно реализованная сложная система для создания веб-страниц без знания языков программирования.

Разница между CMS и конструктором сайтов заключается в том, что первое — это программа, которая предоставляет инструменты для создания собственного проекта, а второе — сервис, сдающий в аренду свои шаблоны и функции. Конструкторы хорошо подходят для создания лендингов (простых одностраничных сайтов), визиток, презентаций услуг и разовых акций. Их отличает быстрота и легкость. CMS же является серьезной платформой и, в отличие от первых, дает большое пространство для творчества. Интернет-магазины, новостные порталы, блоги и социальные сети — все это можно создать на CMS.

Так как создание собственной полноценной CM системы является сложной задачей, я решил разработать лишь простую версию конструктора сайтов. И для начала нужно разобраться как функционирует данный сервис, как он устроен, иначе говоря, разобраться с архитектурой.  Архитектура программной системы - это метафора, аналогичная архитектуре здания. Она функционирует как план для системы и проекта разработки. Несмотря на некоторые различия между CMS и конструктором сайтов, все же их архитектуры очень похожи.

## 1.3 Архитектура проекта

Архитектура создаваемой системы основана на модульном принципе. CMS полностью строится из модулей, как детский домик из кубиков, по принципу модульного программирования. Модульное программирование — организация программы как совокупности небольших независимых блоков, называемых модулями, структура и поведение которых подчиняются определённым правилам.

Модуль — последовательность логически связанных фрагментов, оформленных как отдельная часть программы. При построении модуля используется концепция: «один модуль — одна функция», то есть модуль — элемент программы, решающий одну самостоятельную задачу.

Основные модули CMS:

1. Системный модуль — подключение остальных модулей, ядро системы.
2. Модуль данных - самый важный модуль системы, именно от его возможностей зависит гибкость и функциональная мощь системы.
3. Модуль файлов и фотографий - дополняют действия модуля данных, масштабирование фотографии.
4. Модуль страниц - используя шаблоны оформления, формирует html страницу, которую потом отправляет пользователю.
5. Модуль запросов - собирает все поступившие данные от пользователя в «контейнер», из которого они будут доступны любому модулю системы.
6. Модуль событий — обработка и генерация событий.

Свой конструктор сайтов я решил строить по такому же модульному принципу, адаптируя систему модулей под свои нужды (многие второстепенные модули убраны или объединены в один).

## 1.4 Выбор инструментов для разработки

После осознания архитектуры проекта, остается очень важный вопрос: какие инструменты использовать для разработки, какие языки программирования? Для начала нужно еще немного вникнуть в структуру проекта. Так как конструктор сайтов — это веб приложение, то есть взаимодействие с пользователем осуществляется с помощью браузера, то существует две стороны взаимодействия:

1. Фронтэнд (клиентская часть). Это пользовательский интерфейс, к которому можно отнести всё, что пользователи видят на сайте или в приложении, и с чем можно взаимодействовать: картинки, выпадающие списки, меню, анимация, карточки товаров, кнопки, чекбоксы, интерактивные элементы.
2. Бэкэнд (серверная часть). Это логика работы сайта, скрытая от пользователя. Именно там происходит то, что можно назвать работой сайта.

Пользовательский интерфейс в браузере обычно разрабатывается на языке разметки HTML (от англ. HyperText Markup Language — «язык гипертекстовой разметки») . Для стилизации интерфейса существует формальный язык декорирования и описания внешнего вида документа (веб-страницы) — CSS (англ. Cascading Style Sheets «каскадные таблицы стилей»). А для придания интерактивности приложению принято использовать язык программирование JavaScript.

Серверная часть пишется в основном на PHP, Ruby, Java, Python, Kotlin или Go. Каждый язык программирования имеет свои плюсы и минусы, в подробности я вдаваться не буду, но очень важно то, что сервер должен взаимодействовать с базой данных. Базы данных - это структурированная информация, которая хранится в связанных электронных таблицах. Базы данных способны обеспечивать быстродействие, стандартизацию данных, их безопасность и интегрированность. Выделяют два основных вида БД:

1. Реляционные БД (SQL DB). Записи и связи между ними организованы при помощи таблиц. В таблицах есть поле для внешнего ключа со ссылками на другие таблицы. Благодаря высокой организации и гибкости структуры реляционные БД применяются для многих типов данных.
2. Нереляционные БД (NoSQL DB). В таких базах данные сохраняются под ключами. Если хотите получить объект, например, изображение или текст, нужно ввести ключ. Таким образом часто хранят информацию о состоянии объектов, представленную различными типами данных. Каждому хранилищу разрабатывают свою схему именования ключей в зависимости от форматов значений.

Объединяющим элементон серверной и клиентской части приложения является API (от англ. application programming interface, дословно интерфейс программирования приложения) — программный интерфейс, то есть описание способов взаимодействия одной компьютерной программы с другими. Проще говоря, это способ взаимодействия какого-то программного кода с набором каких-то программных компонентов, с помощью которых одна компьютерная программа может использовать другую программу.

# Глава 2

Итак, одно замечание - очень важно понять, что я буду разрабатывать лишь локальную версию конструктора сайтов (из-за ограниченного количества времени).

## 2.1 Клиентская часть

### 2.1.1 Структура HTML страниц сайта

Как я уже говорил, клиентская часть — это интерфейс взаимодействия с пользователем, в моем случае это сайт, а точнее многостраничный сайт, который будет содержать в себе следующие разделы:

1. **Главная страница сайта. Э**то основная страница, с которой пользователь может:

* зарегистрироваться на сайте / войти в свой аккаунт
* кратко ознакомиться с моим проектом
* понять как он может создать свой сайт
* связаться со мной при наличие вопросов

1. **Страница аккаунта пользователя.** Здесь пользователь может начать создавать свои сайты. Один пользователь может создать десять сайтов.
2. **Страница создания и настройки пользовательского сайта.** На этом этапе пользователь придумывает названию создаваемому сайту, выбирает его шаблон .
3. **Страница редактирования пользовательского сайта.** Это собственно сама страница для создания сайта, тут пользователь создает свой сайт из шаблонных блоков, настраивая их полностью под себя.

### 2.1.2 Дизайн

Чтобы сайт удобно было верстать на HTML, я сначала продумал и создал его дизайн. В этом мне помогла программа Figma (сервис для разработки интерфейсов). Пришлось продумывать дизайн не только основных страниц, но и все окошки регистрации, входа, настройки, все кнопки, подбирать цвета и шрифты для текста и многое другое (*Иллюстрации 5 — 8*).

### 2.1.3 CSS cтилизация

Имея дизайн и структуру сайта, верстать сайт было гораздо удобнее. Для каждой отдельной страницы создается свой файл с расширением .html и затем - структура данной страницы, после этого, подключая файл стилей CSS, странице придается задуманный дизайном вид. У меня был один основной файл стилей, там содержалось оформление всех страниц. Но, чтобы не захламлять основной файл, стили всех всплывающих окон (pop-up), подключение шрифтов, глобальных переменных, стили шаблонных элементов (кнопки) я вынес в отдельные файлы, которые и подключил к основномму (*Иллюстрации 9 — 10*).

### 2.1.4 JavaScript программирование

После создание статической части сайта, пришла пора «оживить» его с помощью JavaScript. К каждому файлу .html подключается свой скрипт js. Скрипты отвечают за интерактивное взаимодействие с пользователем: обработка нажатий на кнопки сайта, открытие pop-up окон регистрации, окон успешной авторизации, обеспечивают навигацию по страницам всего сайта и многое другое, но самое главное — это взаимодействие с внешним API. Что это значит? Все просто. Например, при регистрации или входе в аккаунт js собирает данные с заполненной пользователем формы и отправляет POST запрос на определенный адрес API. После чего сервер отправляет ответ, который опять же принимает наш js и делает определенные действия с полученными данными, будь то успешная регистрация, изменения названия проекта или что-либо еще.

Отдельно надо сказать про процесс создание пользовательского сайта. Изначально у пользователя есть возможность добавить какой либо шаблонный блок на сайт (текстовый блок, блок заголовка, подзаголовка, блок колонок с текстом). При добавление блока, появляется возможность редактирования контента блока (его содержимого) и стилей блока (цвет и размер текста, отступы блока, высота, ширина), а также позиционирование блока относительно других блоков. Каждое действие пользователя сохраняется в истории, за которую отвечает глобальная переменная PAGE\_DATA. Данная переменная представляет собой массив объектов блоков (*Иллюстрация 11*). Данные объекты состоят из следующих полей:

1. **site\_name.** Поскольку сайтов у пользователя может быть несколько, то данное поле отвечает за принадлежность истории изменений к определенному сайту.
2. **block\_id**. У каждого блока на странице есть свой идентификатор, который и указывается в данном поле, чтобы было понятно: с каким блоком взаимодействовал пользователь.
3. **сontent**. Это поле представляет объект изменений.
   * 1. **type**. Статус изменения (add – добавление блока, style – стилизация блока, moving – позиционирование блока)
     2. **what**. Это поле отвечает за HTML содержимое блока, то есть сама структурная разметка блока, с учетом изменений.

Данные в переменной PAGE\_DATA хранятся до тех пор, пока пользователь не нажмет кнопку сохранить изменения. После нажатия данные будут переданы на сервер.

## 2.2 Серверная часть

### 2.2.1 FastAPI

Серверная часть выполняет роль обработки запросов, поступаемых с фронтенда. Бэкенд я решил писать на python, а использовал для этого один из самых быстрых и популярных веб-фреймворков, написанных на python - FastAPI. Выбрал я именно этот фреймворк в силу его простоты. Идея создания API совдится к тому, чтобы создать функции, которые будут обрабатывать поступающие на определенные адреса запросы. Например, чтобы обрабатывать запросы на создание нового пользователя, поступающие с фронтэнда (*Иллюстрация 13*), я написал функцию add\_user (*Иллюстрация 12*). Чтобы понять как работает данная функция, надо немного сказать об организации базы данных в моем проекте.

### 2.2.2 База данных

Для работы с БД я использовал SQLAlchemy, программную библиотеку, написанную на python, для работы с реляционными СУБД (системами управления базами данных) с применением технологии ORM. ORM (англ. Object-Relational Mapping, рус. объектно-реляционное отображение, или преобразование)  – это возможность работать с «виртуальными объектными базами данных». Это обеспечивает работу с данными в терминах классов, а не таблиц данных, что позволяет избавиться от необходимости писать SQL-код для взаимодействия с СУБД. В моем проекте мне понадобится база данных, состоящая из двух таблиц (или классов в системе ORM) (*Иллюстрации 14, 15*):

1. **users**. Это таблица, хранящая в себе всех пользователей и их информацию, а именно: идентификатор, имя, почта, пароль.
2. **sites.** Эта таблица сайтов пользователей, она содержит в себе: собственный идентификатор, идентификатор пользователя (владельца данного сайта), название, ссылку и еще немного технической информации.

Возвращаясь к функции add\_user, теперь понятно, что она просто создает объект пользователя — экземпляр класса User, заполняет его полученными с фронтенда данными пользователя, добавляет его в БД, и обновляет базу.

### 2.2.3 Создание, хранение и изменение пользовательских сайтов.

Отдельного внимания заслуживает процесс создания конкретно пользовательских сайтов. Сначала пользователь посылает POST запрос (очень похожий на запрос с регистрацией, но лишь по другому адресу и с другими данными). Затем, сервер проверяет наличие данного пользователя в БД, после чего в специальной директории user\_sites создает файл .html (файл сайта) с уникальным названием. Уникальное название формируется на основе пользовательского идентификатора, закодированного в base-64 пользовательского имени и идентификатора сайта, поскольку идентификаторы у каждого пользователя уникальны, создается уникальное название файла. После создания основного файла, сервер ждет запроса на изменение файла (кнопка сохранить у пользователя, после нажатия на которую данные из переменной PAGE\_DATA отправляются серверу).

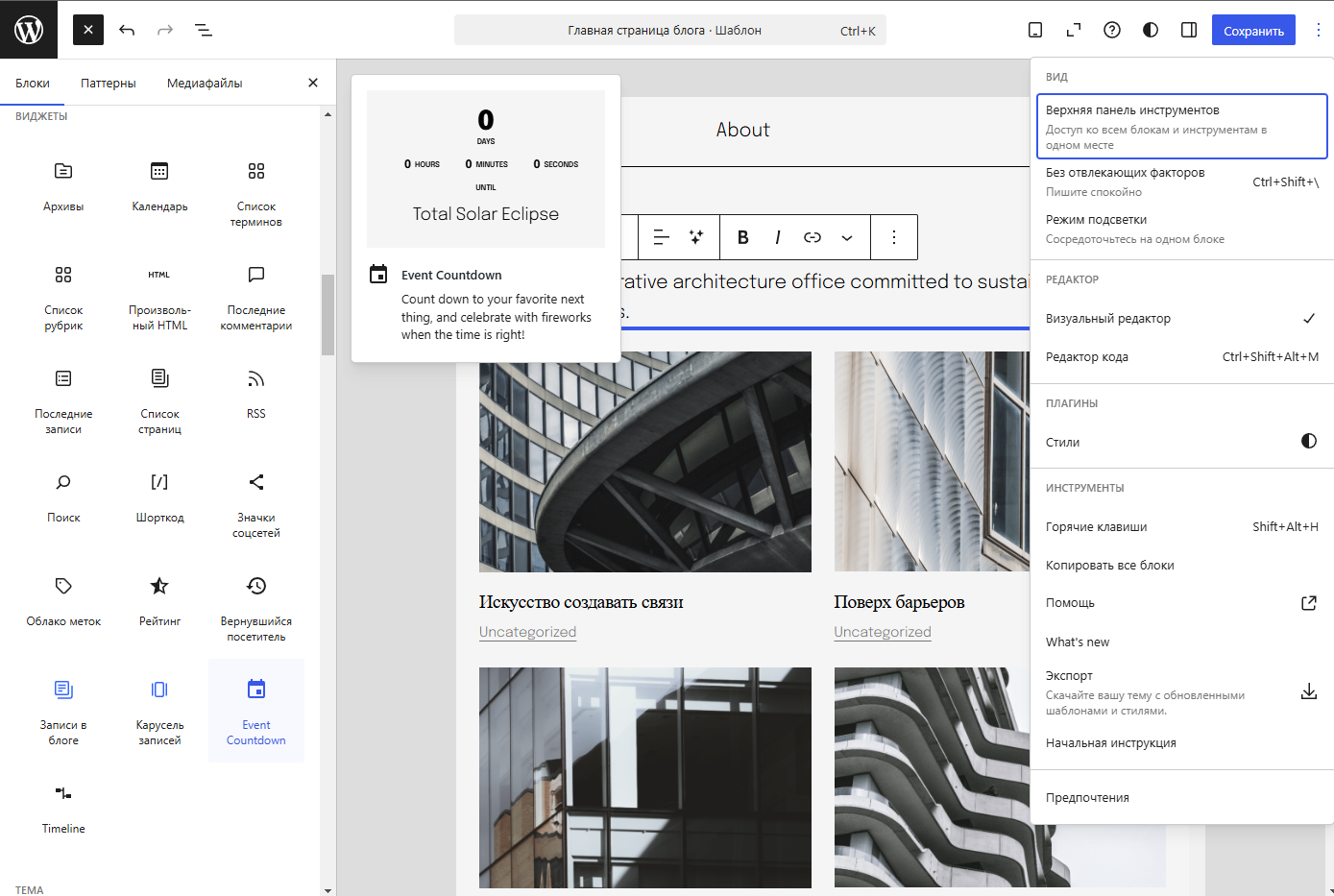
# Заключение

В итоге, изучив существующие аналоги конструкторов сайтов, разобравшись с архитектурами CM систем, с помощью HTML, CSS, JavaScript, FastAPI (python), я создал бесплатный, простой и удобный локальный веб-сервис, позволяющий создавать сайты быстро и без программирования.

# Литература

1. Архитектура CMS / [Электронный ресурс] // URL: <https://habr.com/ru/articles/51152> (Дата обращения: 03.03.2025)
2. Архитектура CMS. Модель данных / [Электронный ресурс] // URL: <https://habr.com/ru/articles/53388> (Дата обращения: 03.03.2025)
3. Модульное программирование / [Электронный ресурс] // URL: <https://en.wikipedia.org/wiki/Modular_programming> (Дата обращения: 05.03.2025)
4. Фронтенд и бэкенд / [Электронный ресурс] // URL: <https://practicum.yandex.ru/blog/chem-otlichaetsya-backend-i-frontend> (Дата обращения: 05.03.2025)
5. Базы данных / [Электронный ресурс] // URL: <https://practicum.yandex.ru/blog/chto-takoe-bazy-dannyh> Дата обращения: 05.03.2025)
6. Виды баз данных / [Электронный ресурс] // URL: <https://habr.com/ru/companies/amvera/articles/754702> (Дата обращения: 05.03.2025)
7. API / [Электронный ресурс] // URL: <https://ru.wikipedia.org/wiki/API> (Дата обращения: 05.03.2025)
8. FastAPI / [Электронный ресурс] // URL: [https://fastapi.tiangolo.com](https://fastapi.tiangolo.com/) (Дата обращения: 01.11.2024 – 09.03.2025)
9. Основы FastAPI / [Электронный ресурс] // URL: <https://metanit.com/python/fastapi/1.1.php> (Дата обращения: 01.11.2024 – 09.03.2025)

# Приложение

Иллюстрация 1

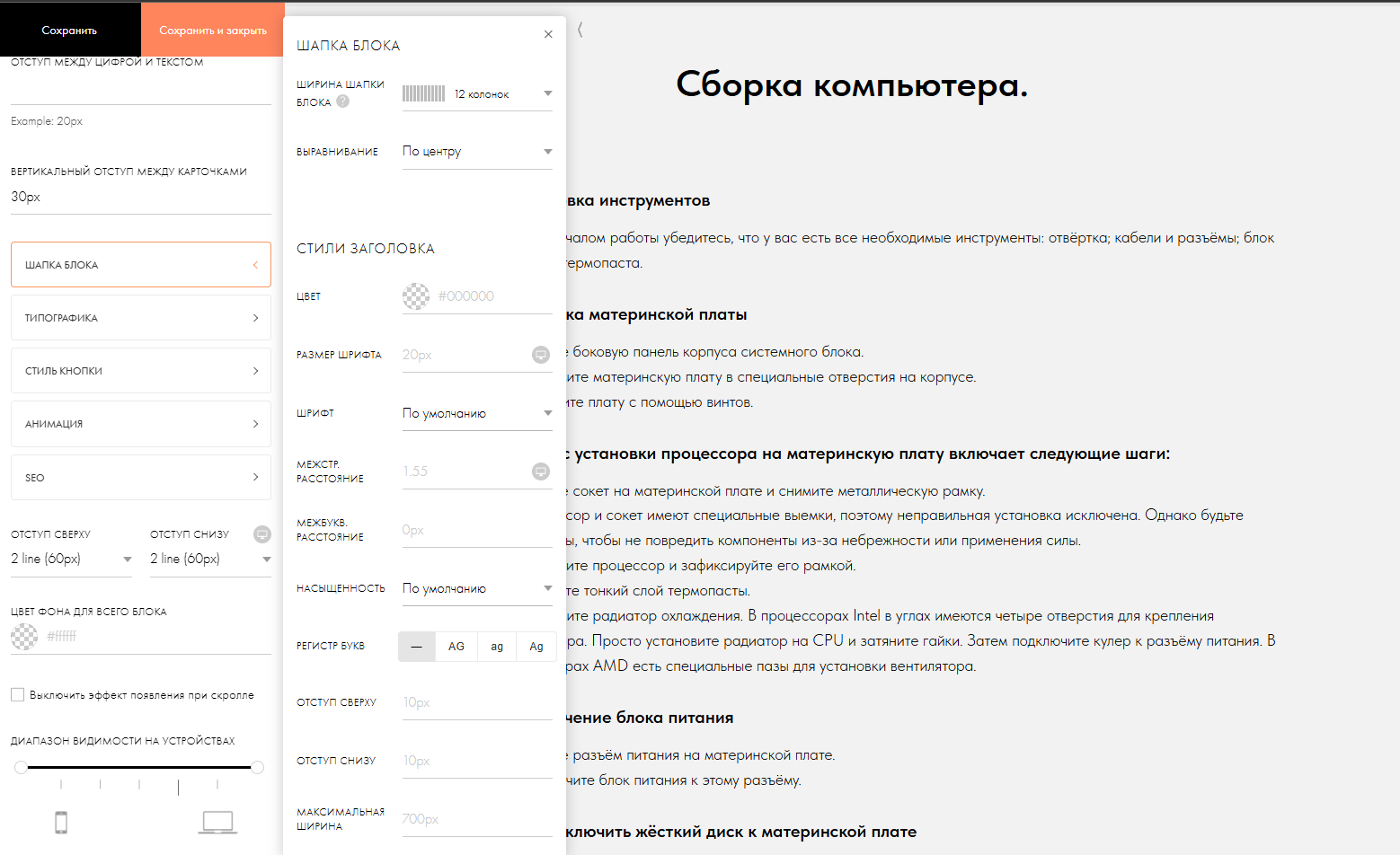
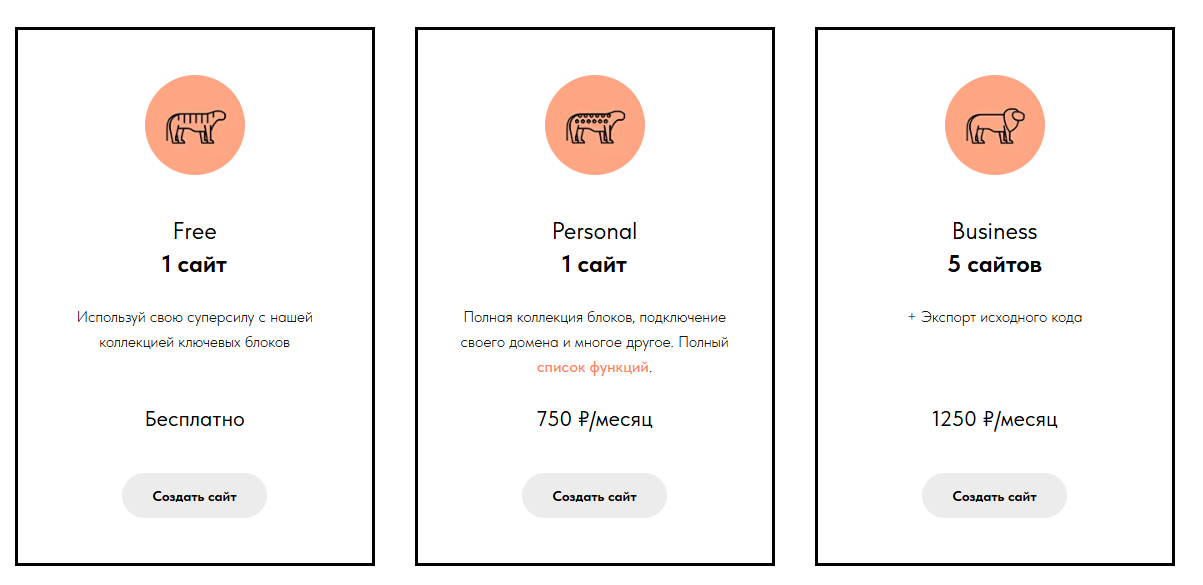
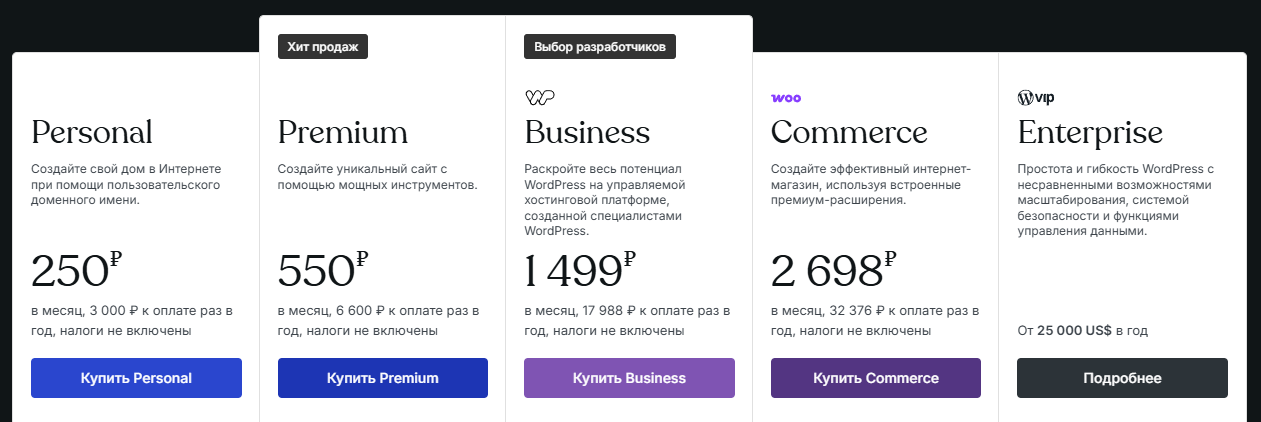
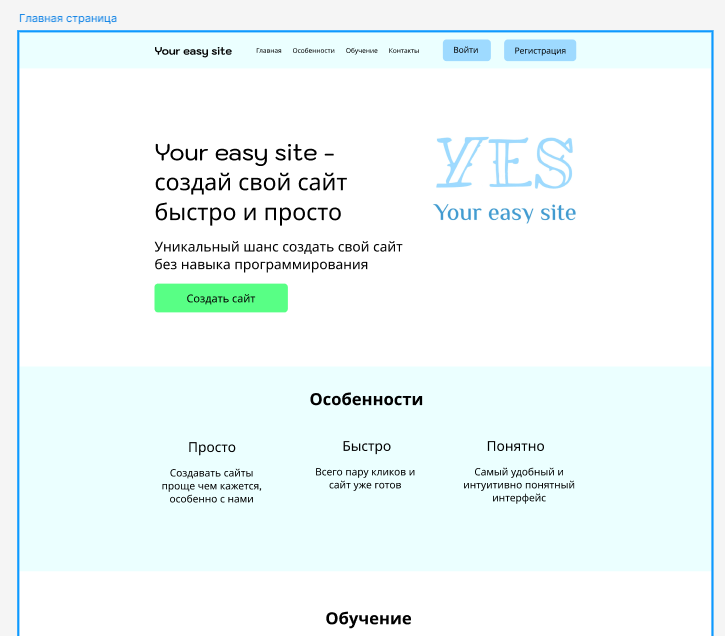
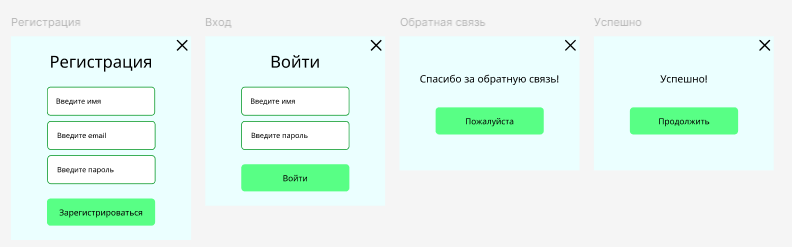


Иллюстрация 2

Иллюстрация 4

Иллюстрация 3

Иллюстрация 6

Иллюстрация 5

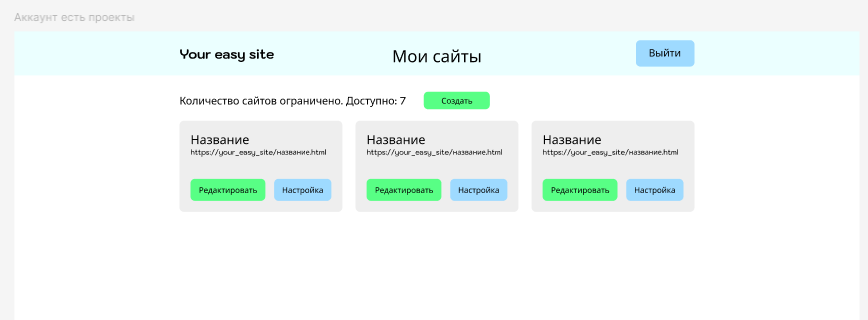
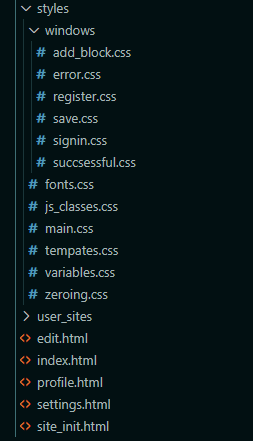
Иллюстрация 7

Иллюстрация 10

Иллюстрация 9

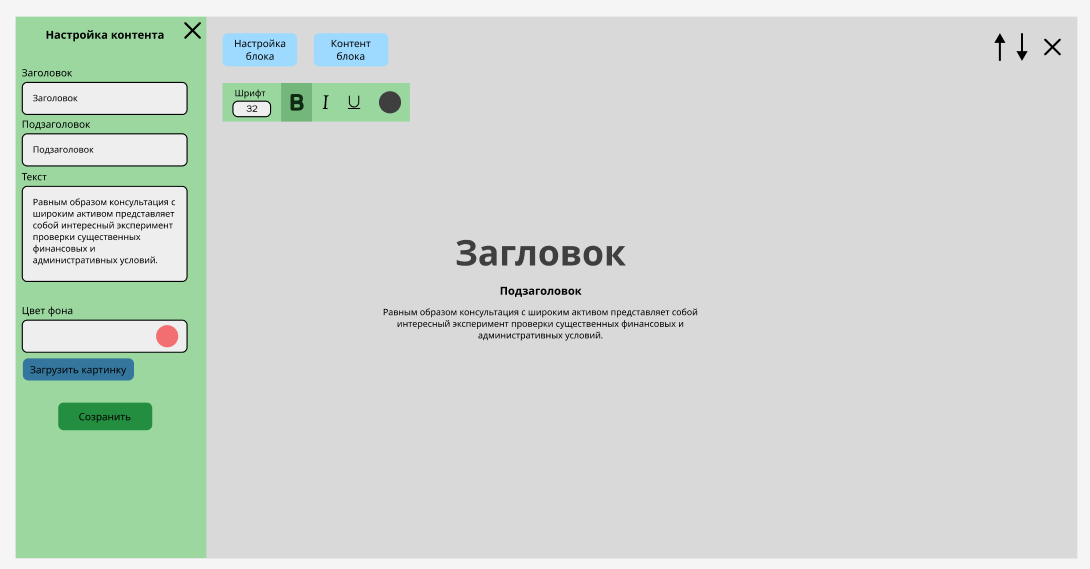
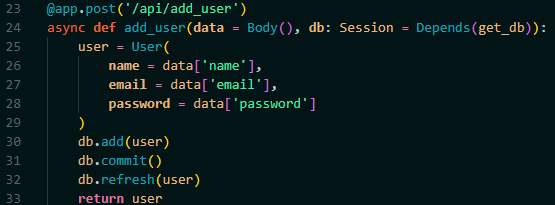
Иллюстрация 8

Иллюстрация 11

Иллюстрация 12

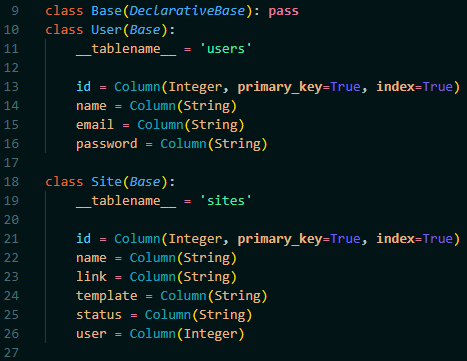
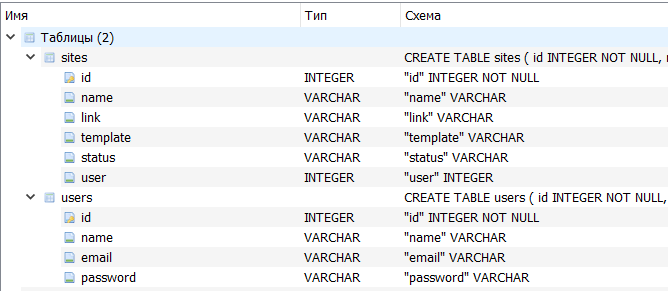
Иллюстрация 14

Иллюстрация 13

Иллюстрация 16

Иллюстрация 15