

Министерство образования Республики Беларусь
Учреждение образования «Белорусский государственный университет
информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра программного обеспечения информационных технологий

Дисциплина: «Конструирование программного обеспечения»

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
к курсовому проекту
на тему

ВЕБ-СЕРВЕР ДЛЯ ОБРАЗОВАТЕЛЬНОЙ ПЛАТФОРМЫ
ПО РУССКОМУ ЯЗЫКУ

БГУИР КП 1-40 01 01 111 ПЗ

Студент: гр. 351001 Ушаков А.Д.

Руководитель: асс. Шостак Е.В.

Минск 2024

Учреждение образования
«Белорусский государственный университет информатики и
радиоэлектроники»

УТВЕРЖДАЮ
Заведующий кафедрой ПОИТ

(подпись)

2024 г.

ЗАДАНИЕ
по курсовому проектированию

Студенту Ушакову Александру Дмитриевичу

1. Тема работы: «Веб-сервер для образовательной платформы по русскому языку»
2. Срок сдачи студентом законченной работы 17.12.2024 г.
3. Исходные данные к работе: разработка программного средства производится в операционной системе Windows с использованием фреймворка Crow для языка программирования C++; создание графического интерфейса предполагает язык разметки HTML, язык стилей CSS, библиотеки nlohmann/json и httpplib, HTML-парсер Gumbo под C++; хранение материалов реализовано в виде JSON-файлов. Функциональность: программа должна создавать сервер, позволяющий пользователю вводить с веб-страницы ответы на вопросы, получать задания и рекомендации, проверять корректность выполнения, сверяя ответы с возвращаемыми данными. Обработка данных: при выполнении определённых действий на странице с помощью различных запросов данные отправляются на сервер, проходя заданные логические блоки, возвращаются обратно; часть данных без обработки берётся на сервере – из JSON-файлов.
4. Содержание расчётно-пояснительной записки (перечень вопросов, которые подлежат разработке):
Введение. 1. Средства разработки и функциональные требования. 2. Проекти-

рование и разработка. 3. Тестирование и проверка работоспособности. 4. Руководство по использованию. Заключение. Приложения.

5. Перечень графического материала (с точным обозначением обязательных чертежей и графиков)

1. Схема алгоритма настройки и работы системы сервер-клиент

6. Консультант по курсовой работе

Шостак Е.В.

7. Дата выдачи задания 7.09.2024 г.

8. Календарный график работы над проектом на весь период проектирования (с обозначением сроков выполнения и процентом от общего объёма работы):

раздел 1,2 к 15.09.2024 – 15 % готовности работы;

разделы 3, 4 к 15.10.2024 – 30 % готовности работы;

разделы 5, 6 к 15.11.2024 – 60 % готовности работы;

раздел 7, 8, 9 к 15.12.2024 – 90 % готовности работы;

оформление пояснительной записки и графического материала к 17.12.2024 – 100 % готовности работы.

Защита курсового проекта с 13 по 17 декабря 2023 г.

РУКОВОДИТЕЛЬ _____ Е. В. Шостак
(подпись)

Задание принял к исполнению А. Д. Ушаков 7.09.2024 г.
(дата и подпись студента)

СОДЕРЖАНИЕ

Введение.....	6
1 Средства разработки и функциональные требования	7
1.1 Описание средств разработки	7
1.1.1 Обработка запросов с использованием ООП.....	7
1.1.2 Хранение в формате JSON.....	8
1.1.3 Crow	9
1.1.4 Библиотека nlohmann/json	10
1.1.5 Библиотека httpplib.....	11
1.1.6 HTML-парсер Gumbo	12
1.1.7 DOM-дерево	13
1.2 Спецификация функциональных требований.....	14
1.2.1 Комментарий к ФТ-1	14
1.2.2 Комментарий к ФТ-2.....	14
1.2.3 Комментарий к ФТ-3.....	15
1.2.4 Комментарий к ФТ-4.....	15
1.2.5 Комментарий к ФТ-5.....	15
1.2.6 Комментарий к ФТ-6.....	16
1.2.7 Комментарий к ФТ-7	16
1.2.8 Комментарий к ФТ-8.....	16
2 Проектирование и разработка	17
2.1 Обзор существующих аналогов	17
2.1.1 Лекторий «Достоевский».....	17
2.1.2 Нетология	19
2.2 Структура проекта.....	22
2.3 Использованные сущности.....	23
2.3.1 Пользовательские типы данных.....	23
2.3.2 Функции	24
2.3.3 Классы	25
2.3.4 Методы	25
2.3.5 Структуры данных.....	26
2.4 Графический интерфейс.....	26
2.5 Схема программного средства	29
3 Тестирование и проверка работоспособности.....	32
3.1 Загрузка страницы	32
3.2 Ввод ответа.....	33
3.3 Отправка ответа	34
3.4 Неверный ответ.....	34
3.5 Правильный ответ.....	35
3.6 Завершение теста	36
4 Руководство по использованию	38
4.1 Запуск сервера.....	38
4.2 Открытие страницы в браузере	38
Заключение.....	39
Список использованных источников.....	40
Приложение А.....	41
Приложение Б	42
Приложение В.....	44
Приложение Г	45
Приложение Д.....	48

Приложение Е	51
Приложение Ж.....	53

ВВЕДЕНИЕ

В современном мире информационные технологии играют важную роль в образовательном процессе. Разработка веб-приложений для обучения становится одним из ключевых направлений в сфере IT. Одним из эффективных инструментов является создание веб-сервера, обеспечивающего взаимодействие между пользователями и образовательной платформой.

Курсовой проект посвящён разработке веб-сервера для образовательной платформы по русскому языку. Сервер разработан с использованием фреймворка Crow, который обеспечивает быструю разработку приложений на языке C++.

Основной целью проекта является создание системы тестирования, позволяющей пользователям выполнять задания по русскому языку в интерактивном режиме. Веб-сервер принимает ответы пользователей, обрабатывает их и предоставляет обратную связь в виде результатов проверки.

Реализация проекта включает разработку серверной части, обеспечивающей приём и обработку данных, а также простого веб-интерфейса для пользователей. На текущем этапе сервер поддерживает функционал отправки ответов на тесты и их автоматическую проверку с обратной связью.

Актуальность данного проекта обусловлена необходимостью создания современных образовательных платформ, которые обеспечивают доступность обучения для широкого круга пользователей. Интерактивные тесты позволяют повысить уровень усвоения учебного материала благодаря своевременной проверке знаний.

В рамках курсового проекта планируется дальнейшее расширение функционала, включающее регистрацию пользователей, сохранение результатов тестирования, создание системы управления заданиями и разработку удобного интерфейса администратора. Эти улучшения позволят повысить удобство работы с платформой и сделать процесс обучения более эффективным.

1 СРЕДСТВА РАЗРАБОТКИ И ФУНКЦИОНАЛЬНЫЕ ТРЕБОВАНИЯ

1.1 Описание средств разработки

В целях создания программного средства рассматриваются следующие подтемы:

- 1 Объектно-ориентированное программирование: создание классов и методов обработки запросов.
- 2 Использование JSON-файлов для хранения материалов по заданной тематике.
- 3 Фреймворк Crow для создания веб-серверов на C++. Он поддерживает маршрутизацию и создание REST API.
- 4 Библиотека `nlohmann/json` для работы с JSON в C++. Она предоставляет удобный интерфейс для сериализации и десериализации JSON-данных.
- 5 Библиотека `httplib` для выполнения HTTP-запросов (клиент и сервер). Она поддерживает GET, POST, загрузку файлов и работу с SSL (через OpenSSL).
- 6 HTML-парсер Gumbo, разработанный Google. Он разбирает HTML-документы в структурированное дерево, что полезно для веб-скрапинга.
- 7 Работа с деревьями DOM.

1.1.1 Обработка запросов с использованием ООП

Объектно-ориентированное программирование — это методология разработки программного обеспечения, основанная на концепции объектов. Объекты представляют собой сущности, обладающие состоянием (данными) и поведением (методами). Основными принципами ООП являются инкапсуляция, наследование, полиморфизм и абстракция. Эти принципы обеспечивают структурированность, повторное использование кода и легкость его поддержки.

Применение ООП актуально в создании веб-маршрутизаторов — компонентов, управляющих обработкой клиентских запросов. Маршрутизатор сопоставляет URL-адреса с обработчиками, выполняющими соответствующие действия. В этом контексте классы обеспечивают четкую архитектуру системы: каждый маршрут может быть реализован как отдельный класс с собственными методами обработки запросов.

Маршрутизатор, реализованный с использованием ООП, имеет основное хранилище маршрутов, где каждому URL соответствует обработчик. При поступлении запроса маршрутизатор определяет, какой класс-обработчик должен быть вызван, и передает управление соответствующему методу. Если маршрут не найден, возвращается сообщение об ошибке, например, «404 Not Found».

Такой подход обеспечивает гибкость, модульность и масштабируемость системы. Новые маршруты можно добавлять без изменения существующего кода, благодаря чему структура приложения остается понятной и легко поддерживаемой. Использование ООП позволяет разделить логику обработки запросов на независимые модули, обеспечивая удобство тестирования и улучшая читаемость кода.

ООП-маршрутизаторы широко применяются в веб-фреймворках, таких как Crow, Django, Flask, FastAPI и Express.js. В этих фреймворках маршруты создаются на основе классов или объектов, что позволяет реализовывать сложные схемы маршрутизации, поддерживая обработку динамических запросов и параметров. Этот подход является стандартом в разработке веб-приложений благодаря своей гибкости, надёжности и масштабируемости.

1.1.2 Хранение в формате JSON

JSON (JavaScript Object Notation) – это текстовый формат данных, используемый для обмена и хранения информации. В языке C++ работа с JSON требует использования сторонних библиотек, поскольку стандартная библиотека C++ не включает встроенную поддержку этого формата. Популярные библиотеки для работы с JSON на C++ включают `nlohmann/json`, `RapidJSON` и `JSON for Modern C++`.

JSON представляет данные в виде пар «ключ-значение». Ключи всегда являются строками, а значения могут быть строками, числами, булевыми значениями, массивами, объектами или `null`. Такая структура обеспечивает гибкость и позволяет хранить сложные иерархические данные.

При использовании C++ для работы с JSON разработчик может создавать объекты JSON, добавлять в них данные, а затем сериализовать их в строку для сохранения в файл или отправки по сети. Обратный процесс – десериализация – предполагает чтение JSON-строки и преобразование её в объект C++ для дальнейшей обработки.

Например, библиотека `nlohmann/json` предоставляет интуитивно понятный интерфейс, близкий к стандартным контейнерам C++. Данные мож-

но создавать с использованием синтаксиса, похожего на инициализацию `std::map` или `std::vector`. RapidJSON, напротив, ориентирован на высокую производительность, что делает его предпочтительным выбором для приложений с ограниченными ресурсами.

При хранении данных в формате JSON важно соблюдать корректность структуры: пары ключ-значение должны быть заключены в фигурные скобки `{}`, массивы – в квадратные скобки `[]`, а строки – в двойные кавычки `" "`. Любые ошибки в синтаксисе могут привести к сбоям при чтении или записи данных.

Использование JSON на C++ обеспечивает гибкость в работе с конфигурационными файлами, сохранением состояния приложений, передаче данных через сеть и интеграции с веб-сервисами. Благодаря мощным библиотекам и активному сообществу разработчиков работа с JSON в C++ становится простой и удобной даже в сложных проектах.

1.1.3 Crow

Crow – это современный веб-фреймворк на C++, предназначенный для создания веб-сервисов и API. Он предлагает простую и интуитивно понятную модель маршрутизации, поддержку асинхронной обработки запросов и интеграцию с форматами данных, такими как JSON. Благодаря своей легковесности и высокой производительности Crow подходит как для небольших приложений, так и для сложных серверных решений.

Одной из ключевых особенностей Crow является его минималистичный подход к созданию веб-приложений. Фреймворк позволяет легко настраивать маршруты, определять обработчики запросов и возвращать HTTP-ответы. Поддержка асинхронного ввода-вывода обеспечивает высокую пропускную способность при работе с большим числом одновременных соединений.

Crow поддерживает маршрутизацию с параметрами, регулярными выражениями и разными HTTP-методами, такими как GET, POST, PUT и DELETE. Обработчики запросов определяются с помощью удобного синтаксиса, что снижает порог входа для разработчиков. Встроенная поддержка JSON позволяет легко разбирать и формировать ответы, используя совместимую библиотеку `nlohmann/json`.

Сервер, созданный на базе Crow, легко конфигурируется. Разработчик может задавать параметры, такие как номер порта, IP-адрес, уровень логирования и использование HTTPS. Это делает фреймворк универсальным инструментом для создания REST API, микросервисов и полноценных сервер-

ных приложений.

Crow также поддерживает статические файлы, что позволяет обслуживать веб-страницы, стили CSS и скрипты JavaScript без необходимости использования дополнительного веб-сервера. Асинхронная модель на основе библиотеки Boost.Asio обеспечивает высокую производительность в многопоточных средах.

Благодаря своей простоте, высокой скорости работы и поддержке современных стандартов C++ Crow стал популярным выбором среди разработчиков, ищущих легковесный и производительный веб-фреймворк. Он активно развивается и поддерживается сообществом, обеспечивая надежную основу для создания веб-приложений на C++.

1.1.4 Библиотека **nlohmann/json**

Nlohmann/json – это популярная библиотека для работы с форматом JSON на языке C++. Она предоставляет удобный и интуитивно понятный интерфейс, который позволяет работать с JSON так же легко, как с контейнерами стандартной библиотеки C++. Библиотека активно развивается и широко используется в проектах благодаря простоте использования, поддержке современных стандартов C++ и хорошей документации.

Главным преимуществом nlohmann/json является её синтаксис, похожий на стандартные контейнеры `std::map` и `std::vector`. Данные в формате JSON представляются с помощью универсального объекта `json`, который может хранить строки, числа, массивы, объекты, булевы значения и `null`. Это позволяет работать с JSON-структурами как с обычными контейнерами C++.

Библиотека поддерживает автоматическую сериализацию и десериализацию объектов C++. Достаточно использовать встроенные методы `dump()` для преобразования объекта в строку JSON и `parse()` для обратного преобразования. Кроме того, она обеспечивает удобное преобразование пользовательских структур данных с использованием шаблонных функций `to_json()` и `from_json()`.

nlohmann/json также поддерживает различные форматы ввода-вывода, включая файлы, потоки и строки. Это позволяет легко интегрировать её в системы, работающие с конфигурационными файлами, сетевыми сервисами и базами данных. Дополнительные функции включают работу с итераторами, удобные методы доступа к элементам и поддержку исключений для обработки ошибок.

Библиотека совместима с основными компиляторами C++ и поддержи-

вает стандарты C++11 и выше. Она распространяется под либеральной лицензией MIT, что делает её доступной для использования как в коммерческих, так и в открытых проектах.

Благодаря простоте, гибкости и мощным возможностям `nlohmann/json` стала стандартом де-факто для работы с JSON в C++. Она позволяет разработчикам создавать надежные и хорошо структурированные приложения с минимальными затратами времени на обработку данных.

1.1.5 Библиотека `httplib`

`HttpLib` – это легковесная библиотека на C++ для создания HTTP-клиентов и серверов. Она разработана с акцентом на простоту, понятный интерфейс и минимальные зависимости. Библиотека поддерживает основные HTTP-методы, такие как GET, POST, PUT и DELETE, обеспечивая эффективное взаимодействие между клиентами и серверами.

Библиотека `httplib` выделяется своей кроссплатформенностью и автономностью. Для базового использования она не требует сторонних библиотек, кроме стандартной библиотеки C++. Однако для поддержки HTTPS может понадобиться библиотека `OpenSSL`. Благодаря этому `httplib` часто используется в проектах, где важны легкость интеграции и минимальные системные требования.

Создание HTTP-сервера с помощью `httplib` требует минимального объема кода. Сервер настраивается с использованием методов, определяющих обработчики запросов для заданных маршрутов. Обработчики получают информацию о запросах и возвращают HTTP-ответы. Библиотека также поддерживает установку заголовков, работу с параметрами URL и передачу тела запроса.

В режиме клиента `httplib` обеспечивает простой способ отправки HTTP-запросов. Методы `Get()`, `Post()` и другие позволяют выполнять запросы и получать ответы с минимальными настройками. Поддерживаются отправка форм, загрузка файлов, работа с заголовками, настройка тайм-аутов и работа с переадресацией.

Дополнительные возможности библиотеки включают управление многопоточностью, обработку статических файлов на сервере, поддержку JSON-данных и установку собственных обработчиков ошибок. Для улучшения безопасности при работе с HTTPS предусмотрена проверка сертификатов и использование пользовательских настроек SSL.

Библиотека `httplib` имеет открытый исходный код и распространяется по

лицензии MIT. Благодаря простому API, гибкости и низким системным требованиям она широко используется для разработки веб-сервисов, API и приложений IoT, где важны минимализм и высокая производительность.

1.1.6 HTML-парсер Gumbo

Gumbo – это открытая библиотека на C для парсинга HTML-документов. Она разработана компанией Google и предназначена для разбора HTML-кода в соответствии со спецификацией HTML5. Gumbo обеспечивает надежное и корректное извлечение структурированных данных из веб-страниц, даже если HTML-код содержит ошибки или нестандартные конструкции.

Основной целью разработки Gumbo было создание парсера, соответствующего стандартам W3C. Библиотека полностью реализует спецификацию HTML5, обеспечивая точное и предсказуемое поведение при обработке веб-контента. Она автоматически исправляет типичные ошибки разметки, такие как незакрытые теги и неправильные вложенные элементы, что делает её полезной для обработки реальных веб-страниц.

Gumbo использует удобную модель представления данных: HTML-документ преобразуется в дерево DOM (Document Object Model), представленное структурами C. Каждая нода имеет тип (например, элемент, текст или комментарий) и содержит соответствующие атрибуты, такие как имя тега, текстовое содержимое или список дочерних элементов. Это позволяет легко обходить дерево и извлекать необходимые данные.

Библиотека ориентирована на высокую производительность и эффективное управление памятью. Разработчик отвечает за освобождение выделенных ресурсов, что дает контроль над использованием памяти в крупных проектах. При этом интерфейс Gumbo остаётся простым и интуитивно понятным для разработчиков, знакомых с языком C.

Gumbo часто применяется в веб-скрейпинге, индексировании контента, анализе HTML и разработке поисковых систем. Благодаря своей стабильности, совместимости со спецификацией HTML5 и минимальным зависимостям, библиотека стала популярным выбором среди разработчиков, работающих с HTML на уровне системы. Она распространяется под лицензией Apache 2.0, что делает её доступной как для коммерческих, так и для открытых проектов.

1.1.7 DOM-дерево

DOM (Document Object Model) – это иерархическая структура, которая представляет HTML-документ в виде дерева узлов. Каждый элемент HTML, атрибут или текст становятся узлами этого дерева, что позволяет изменять содержимое и структуру веб-страниц с помощью программных средств. Эта модель обеспечивает доступ к элементам и их свойствам через интерфейсы, позволяя манипулировать веб-документами динамически, не перезагружая страницу.

В контексте работы с библиотекой Gumbo на языке C, DOM представляет собой структуру, в которой HTML-документ представлен в виде дерева узлов. Gumbo парсит HTML в соответствии со стандартом HTML5, создавая структуру, которая моделирует сам документ. Каждый узел в дереве может представлять элемент HTML, текстовый блок или атрибут, что дает возможность извлекать и изменять данные, работая с ними программно.

Используя Gumbo, разработчик может получить доступ к различным элементам документа, обходя это дерево узлов. Например, каждый HTML-элемент, такой как тег `<div>`, становится узлом дерева, который может содержать дочерние узлы, такие как другие теги или текстовые блоки. Текстовые узлы содержат текстовое содержимое между тегами, а атрибуты элементов (например, `id` или `class`) представляют собой пары «ключ-значение», которые также можно обрабатывать как узлы дерева.

Библиотека Gumbo строит дерево на основе входного HTML-документа с корневым элементом `GumboOutput`. Это дерево предоставляет все необходимые данные для работы с веб-страницей, позволяя извлекать информацию и манипулировать содержимым. Когда HTML-документ парсится с помощью функции `gumbo_parse()`, создается структура данных, которая точно отражает содержание документа, и это дерево можно использовать для дальнейшего анализа или извлечения данных.

Кроме того, Gumbo автоматически исправляет типичные ошибки HTML, такие как незакрытые теги или неправильная вложенность элементов. Это позволяет работать с реальными веб-страницами, даже если они содержат ошибки в разметке. Работая с деревом DOM через Gumbo, разработчик получает мощный инструмент для анализа и извлечения информации из HTML-документов, что делает библиотеку удобной для использования в различных проектах по парсингу и скрейпингу данных.

1.2 Спецификация функциональных требований

Под функциональными требованиями подразумевается список задач, операций и функций, которые будут выполняться разрабатываемым программным средством.

Функциональные требования к данному программному средству указаны в таблице 1.2.

Таблица 1.1 – Функциональные требования

Идентификатор	Требование
ФТ-1	Загрузка сервером данных на страницу
ФТ-2	Процесс обмена данными между клиентом и сервером
ФТ-3	Обработка ответов на задания. Серверная часть
ФТ-4	Отправка сервером тематических материалов
ФТ-5	Синтаксический анализ и обработка JSON-файлов с тематическими материалами. Серверная часть
ФТ-6	Хранение страницы, на которой находится пользователь. Серверная часть
ФТ-7	Веб-страница для пользователя
ФТ-8	Обработка DOM сервером

1.2.1 Комментарий к ФТ-1

Это требование подразумевает реализацию функционала, который позволяет веб-серверу загружать данные на клиентскую страницу. Для этого сервер должен отвечать на HTTP-запросы от клиента и передавать необходимые данные в виде HTML-страниц, а также других данных, таких как тесты, результаты и материалы. Использование фреймворка Crow позволяет быстро настроить маршруты для обработки этих запросов, обеспечивая удобную и быструю передачу данных в формате HTML. Важно, чтобы процесс загрузки данных был оптимизирован для предотвращения задержек, что достигается с помощью асинхронной обработки запросов в Crow.

1.2.2 Комментарий к ФТ-2

Обмен данными между клиентом и сервером является неотъемлемой частью работы платформы. Сервер должен принимать запросы от клиента (например, ответы на тесты или запросы на получение новых заданий) и отправлять обратно необходимые данные. Для обеспечения быстрого обмена данными можно использовать протокол HTTP, который поддерживается фреймворком Crow. Одним из ключевых аспектов является использование

метода POST для отправки данных от клиента к серверу и метода GET для получения данных.

Кроме того, важно предусмотреть систему обработки ошибок, которая обеспечит корректную работу сервера в случае некорректных или ошибочных данных. Для улучшения взаимодействия с клиентом данные могут быть отправлены в формате JSON, что облегчает их обработку на стороне клиента. С помощью `hhttplib` можно настроить отправку и получение данных с клиента, что позволит эффективно управлять всем процессом обмена.

1.2.3 Комментарий к ФТ-3

Серверная часть платформы должна быть способна принимать ответы пользователей на тесты, проверять их на корректность и предоставлять обратную связь. Сервер должен обрабатывать входящие ответы с учетом логики тестов, например, сравнивая ответы пользователя с правильными, которые хранятся на сервере, и предоставлять пользователю результат проверки. Это может включать подсчёт правильных и неправильных ответов, вычисление процента правильных ответов и вывод комментариев.

1.2.4 Комментарий к ФТ-4

Образовательная платформа должна обеспечивать отправку тематических материалов пользователю. Эти материалы могут быть текстовыми, графическими или в других форматах и должны быть доступны для загрузки по запросу пользователя при определённых действиях на странице

1.2.5 Комментарий к ФТ-5

Для хранения тематических материалов и другой информации, необходимой для платформы, используется формат JSON. Сервер должен обеспечивать синтаксический анализ и обработку этих файлов. Это включает в себя извлечение нужной информации из JSON-документов и передачу её клиенту. Важным аспектом является использование библиотеки `nlohmann/json`, которая позволяет удобно работать с JSON-данными в C++. Она позволяет эффективно парсить, модифицировать и передавать данные между сервером и клиентом.

1.2.6 Комментарий к ФТ-6

Для улучшения взаимодействия с пользователем и удобства работы с платформой сервер должен поддерживать хранение информации о текущей странице, на которой находится пользователь. Эта информация позволяет динамически обновлять интерфейс и предоставлять контекстно-зависимую информацию, например, продолжение теста или переход к следующему заданию.

1.2.7 Комментарий к ФТ-7

Платформа должна предоставлять удобный интерфейс для пользователя, через который он сможет выполнять тесты, получать результаты и взаимодействовать с материалами. Веб-страница будет являться основным каналом для обмена информацией с пользователем. Интерфейс должен быть интуитивно понятным и доступным для широкой аудитории, включая поддержку адаптивного дизайна для различных типов устройств.

1.2.8 Комментарий к ФТ-8

Для корректной работы веб-страниц с динамическим контентом сервер должен быть способен обрабатывать DOM (Document Object Model) на стороне сервера. Это включает в себя создание, изменение и отправку HTML-страниц с учётом действий пользователя. Например, при изменении данных на веб-странице или отправке формы сервер должен правильно обработать и вернуть обновлённую версию страницы.

2 ПРОЕКТИРОВАНИЕ И РАЗРАБОТКА

2.1 Обзор существующих аналогов

При проектировании программного средства стоит обратить внимание на довольно качественные образовательные платформы: лекторий «Достоевский» и Нетология.

2.1.1 Лекторий «Достоевский»

На рисунках 2.1 – 2.3 представлен интерфейс лектория, на котором отлична видна навигация по сайту, содержание разделов и оформление материалов.

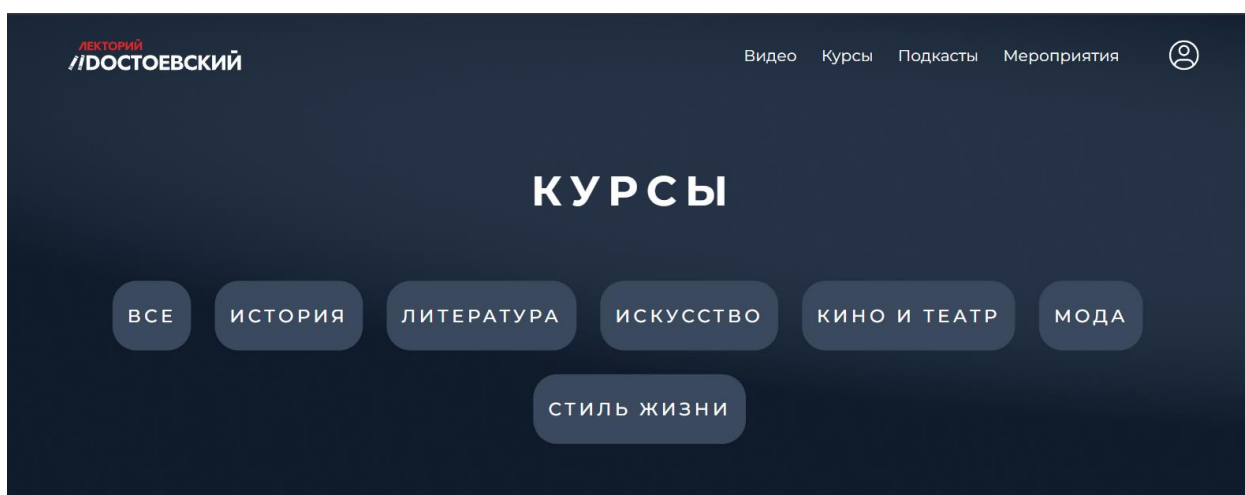


Рисунок 2.1 – Навигация по лекторию

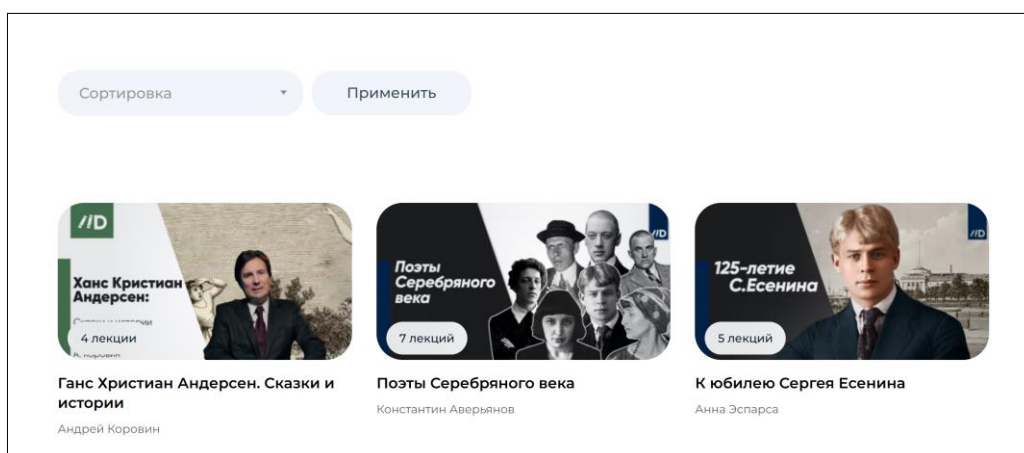
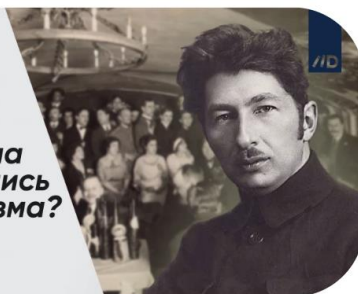


Рисунок 2.2 – Содержимое раздела «Литература»

ЛЕКТОРИЙ
ДОСТОЕВСКИЙ

**Музыка
символизма
или живопись
акмеизма?**



**АКМЕИЗМ КАК МОЛОДЕЖНЫЙ БУНТ: МУЗЫКА
СИМВОЛИЗМА ИЛИ ЖИВОПИСЬ АКМЕИЗМА?**


Автор:

Софья Давыдова

Дата публикации : 22.10.2020

Все в жизни меняется, но молодежный бунт — удел каждой эпохи. Эта лекция о том, как молодое поколение воспротивилось идеям старого, в результате чего общество познакомилось с новым направлением поэзии — акмеизмом. Софья Давыдова расскажет, как благодаря конфликту поколений появился «Цех поэтов» и о том, как же все-таки отличить акмеизм от символизма. P.S. А вы знали, что символисты — немножко спиритисты?

АВТОР



Константин Александрович Аверьянов

Историк, ведущий научный сотрудник ИРИ РАН, д.и.н.

Российский историк, доктор исторических наук, ведущий научный сотрудник Института российской истории РАН. Специализируется по истории XIV—XVII веков, истории Москвы и Подмосковья. Член Городской межведомственной комиссии по наименованию территориальных единиц, улиц, станций метрополитена, организаций и других объектов города Москвы. Выпускник исторического факультета МГПИ им. В.И. Ленина (ныне – МПГУ).

Рисунок 2.3 – Пример оформления одного из тестов

Лекторий «Достоевский» — это современная образовательная платформа, ориентированная на популяризацию сложных культурных, исторических и гуманитарных тем в доступной форме. Проект предлагает лекции, подкасты, курсы и интервью с известными специалистами, что делает его ценным ресурсом для любителей истории, литературы и искусства. Сайт доступен по адресу: dostoverno.ru.

Преимущества платформы:

- 1 Широкий выбор контента: платформа предлагает лекции, подкасты, курсы, интервью и образовательные видео по истории, литературе, искусству и другим гуманитарным наукам.
- 2 Доступность: материалы доступны в видео- и аудиоформате, что позволяет изучать их в удобное время. Многие курсы и мероприятия бесплатны благодаря поддержке Российского фонда культуры.
- 3 Разнообразные форматы обучения: платформа проводит как онлайн-лекции, так и офлайн-встречи с известными деятелями культуры, что повышает уровень вовлечённости аудитории.
- 4 Культурное партнёрство: проект сотрудничает с крупными культурными учреждениями, такими как Третьяковская галерея, Российская

государственная библиотека и телеканал «Звезда», что обеспечивает высокий уровень образовательного контента.

Недостатки платформы:

1 Ограниченная тематика: контент в основном ориентирован на гуманитарные дисциплины, что может быть недостатком для тех, кто интересуется техническими или естественнонаучными направлениями.

2 Развивающийся проект: поскольку платформа относительно новая, база курсов всё ещё расширяется, а некоторые направления пока остаются мало представленными.

3 Технические ограничения: для доступа к онлайн-материалам требуется стабильное интернет-соединение, а участие в офлайн-мероприятиях возможно только в Москве.

4 Языковая ограниченность: все материалы доступны исключительно на русском языке, что сужает международную аудиторию платформы.

Таким образом, лекторий «Достоевский» — это перспективная образовательная платформа, которая объединяет традиционные культурные ценности и современные технологии, делая образование доступным и интересным для широкой аудитории.

2.1.2 Нетология

На рисунках 2.4 – 2.6 изображён интерфейс платформы, на котором представлены навигация по сайту, содержание разделов и оформление материалов.

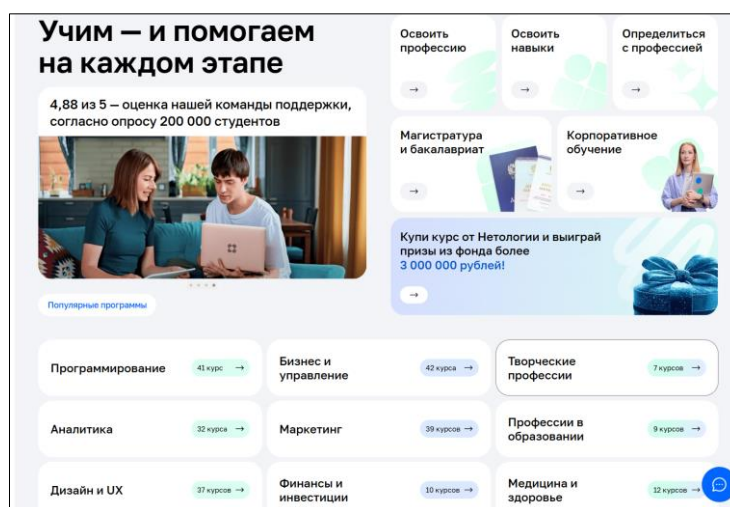
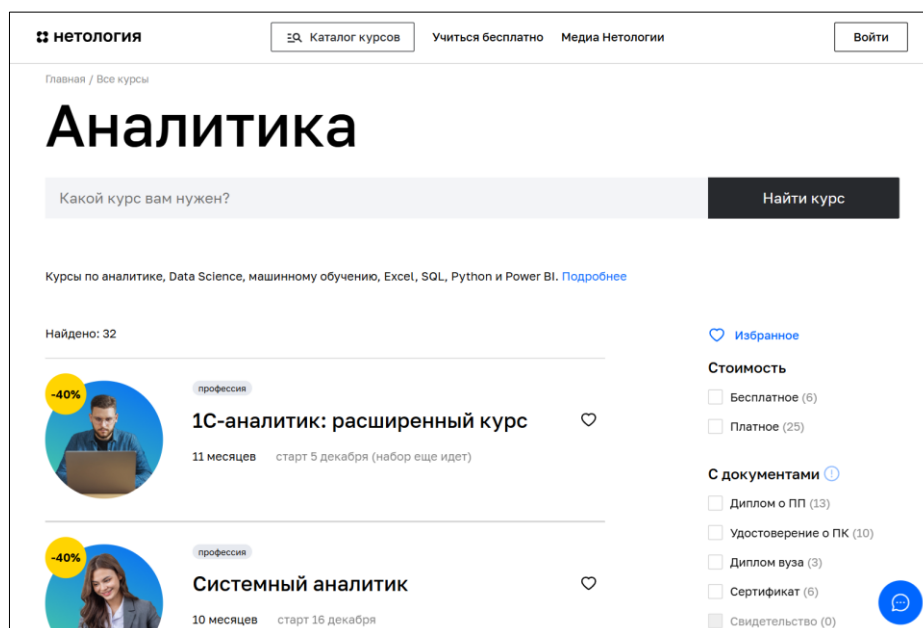


Рисунок 2.4 – Навигация по сайту



2.5 – Внешний вид раздела на сайте

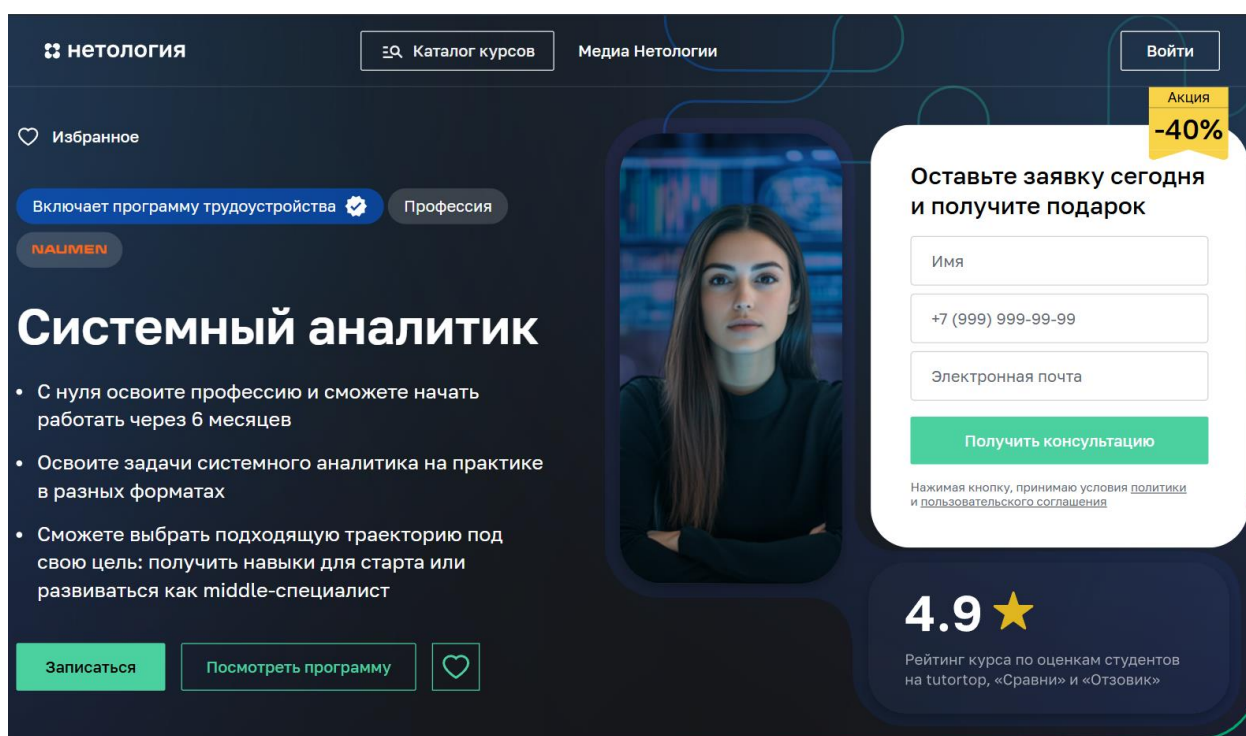


Рисунок 2.6 – Пример курса на сайте

«Нетология» – предлагает курсы по маркетингу, дизайну, управлению и другим направлениям, ориентированным на профессиональное развитие. Подробнее на сайте: netology.ru

Преимущества:

1 Технологическая база: платформа предлагает собственную систему управления обучением (LMS), обеспечивающую удобный доступ к учебным материалам и заданиям.

2 Интерактивное обучение: используются современные технологии, включая вебинары, тестирование, проверку домашних заданий и удалённые экзамены.

3 Поддержка студентов: кураторы и преподаватели взаимодействуют с учащимися через чаты и личные кабинеты, обеспечивая обратную связь.

4 Доступ к материалам: в зависимости от программы, материалы могут открываться сразу или постепенно, что позволяет адаптировать обучение под индивидуальный график.

Недостатки:

1 Отсутствие единого стандарта контента: в отзывах отмечается, что некоторые методики в заданиях не были объяснены в теоретическом материале, что усложняет самостоятельное изучение.

2 Самоорганизация: успешное обучение требует высокой дисциплины, поскольку студентам приходится контролировать свой учебный процесс.

3 Стоимость: обучение на платформе может быть дорогим, особенно на длительных курсах с дипломами или программами профессиональной переподготовки.

4 Зависимость от интернета: полное функционирование платформы требует стабильного интернет-соединения, что может стать препятствием для некоторых пользователей.

Эти особенности делают «Нетологию» подходящей для тех, кто готов активно участвовать в процессе обучения, самостоятельно управлять своим временем и использовать цифровые технологии на высоком уровне.

2.2 Структура проекта

На рисунке 2.2.1 представлена файловая организация проекта.



Рисунок 2.7 – Структура проекта

Файл `Materials.json` содержит материалы, относящиеся к тематике: задания по русскому языку.

В каталоге `cppClient` располагаются файлы, которые отвечают за клиентскую часть, обработку происходящего на веб-странице и отправку запросов на сервер. Файл `checkButtonClick.cpp` обрабатывает действия при нажатии кнопок на странице и отправляет POST-запросы. Файл `firstPageOnLoad.cpp` отправляет на сервер GET-запросы для получения тематических материалов.

В каталоге `cppServer` располагаются файлы, относящиеся к серверной части. Файл `app.cpp` создаёт сервер и задаёт маршруты для принятия запросов. Файл `controller.cpp` создаёт классы для обработки запросов. Файл `materialsParams.cpp` используется для парсинга JSON.

В директории `layout` располагается статический материал, формирующий веб-страницу.

2.3 Используемые сущности

Сущности, такие, как пользовательские типы данных, функции, классы, структуры данных, использованные в проекте, формируют основу для эффективного взаимодействия между серверной и клиентской частями приложения. Они обеспечивают обработку запросов, управление данными, их хранение и передачу между сервером и клиентом, а также обновление пользовательского интерфейса. Использование таких технологий, как JSON для передачи данных, `Crow` для маршрутизации запросов, и `Gumbo` для работы с HTML-страницами, позволяет проекту быть гибким, масштабируемым и удобным в расширении, что критично для обеспечения стабильной и динамичной работы веб-приложения.

2.3.1 Пользовательские типы данных

На рисунке 2.8 представлена таблица, которая содержит настраиваемые типы данных, используемые в проекте. Они представляют объекты, с которыми ведётся работа, включая ответы сервера, структуры JSON и объекты для HTTP-запросов.

Название	Характеристика	Назначение
<code>nlohmann::json</code>	Шаблонный тип из библиотеки nlohmann	Представление данных в формате JSON
<code>crow::response</code>	Тип ответа в фреймворке Crow	Возврат ответов сервера клиенту
<code>httplib::Client</code>	Клиент для HTTP-запросов	Отправка HTTP-запросов на сервер

Рисунок 2.8 – Пользовательские типы данных

2.3.2 Функции

На рисунке 2.9 представлена таблица, которая содержит функции, реализованные в проекте. Они обеспечивают основную бизнес-логику, включая обработку запросов, чтение файлов, взаимодействие с сервером и управление HTML-контентом.

Название	Характеристика	Назначение	Параметры
<code>startServer()</code>	Функция запуска сервера	Запускает сервер	Нет
<code>firstPageOnLoad()</code>	Обработчик GET-запроса	Возвращает данные теста	Нет
<code>checkButtonClick()</code>	Обработчик POST-запроса	Проверяет ответ клиента	<code>const crow::request& req</code>
<code>readJSON()</code>	Читает файл JSON	Возвращает объект JSON	<code>const std::string& fileName</code>
<code>findElementByClass()</code>	Ищет элемент в HTML по классу	Возвращает текст элемента	<code>GumboNode* node, const std::string& className</code>
<code>findElementsByClass()</code>	Ищет все элементы с указанным классом	Возвращает список текстов	<code>GumboNode* node, const std::string& className</code>
<code>onSendButtonClick()</code>	Обрабатывает клик по кнопке	Отправляет POST-запрос	<code>const std::string& serverAddress, std::string& editableInput, int& page</code>
<code>fetchFromServer()</code>	Загружает данные с сервера	Возвращает JSON-объект	<code>const std::string& serverAddress, const std::string& endpoint</code>

Рисунок 2.9 – Задействованные функции

2.3.3 Классы

На рисунке 2.10 представлена таблица, которая включает классы, использованные в проекте для работы с сервером, JSON-данными и HTTP-запросами. Они обеспечивают поддержку серверных маршрутов, работу с данными и клиентскую часть.

Название	Характеристика	Назначение
<code>crow::SimpleApp</code>	Веб-приложение Crow	Определение серверных маршрутов
<code>nlohmann::json</code>	Представление JSON	Работа с данными JSON
<code>httplib::Client</code>	HTTP-клиент	Отправка HTTP-запросов

Рисунок 2.10 – Классы в проекте

2.3.4 Методы

Класс	Название	Характеристика	Назначение	Параметры
<code>crow::SimpleApp</code>	<code>route_dynamic()</code>	Определение маршрута API	Создание маршрутов	<code>const std::string& route</code>
<code>crow::SimpleApp</code>	<code>mount()</code>	Монтирование статического каталога	Сервинг статических файлов	<code>const std::string& url_prefix,</code> <code>crow::mustache::context</code>
<code>crow::SimpleApp</code>	<code>port()</code>	Задаёт порт сервера	Конфигурация сервера	<code>int port</code>
<code>crow::SimpleApp</code>	<code>multithreaded()</code>	Включает многопоточность	Оптимизация сервера	Нет
<code>crow::SimpleApp</code>	<code>run()</code>	Запуск сервера	Запуск серверного приложения	Нет
<code>httplib::Client</code>	<code>Post()</code>	Отправка POST-запроса	Отправляет данные на сервер	<code>const char* path, const</code> <code>std::string& body,</code> <code>const char*</code> <code>content_type</code>
<code>httplib::Client</code>	<code>Get()</code>	Отправка GET-запроса	Получает данные с сервера	<code>const char* path</code>

Рисунок 2.11 – Методы использованных классов

На рисунке 2.11 представлена таблица, которая описывает методы, связанные с функциональностью классов. Они используются для настройки сервера, обработки маршрутов, управления HTTP-запросами и динамического формирования ответов.

2.3.5 Структуры данных

На рисунке 2.12 представлена таблица, которая описывает основные структуры данных, используемые для хранения информации в HTML-дереве и работы с элементами интерфейса.

Название	Характеристика	Назначение
<code>GumboNode</code>	Узел HTML-дерева	Представление HTML-узла
<code>GumboVector</code>	Контейнер для узлов HTML	Хранение дочерних узлов
<code>GumboOutput</code>	Результат парсинга HTML	Результат работы Gumbo
<code>crow::mustache::context</code>	Контекст шаблонов	Генерация HTML-страниц

Рисунок 2.12 – Структуры данных проекта

2.4 Графический интерфейс

На рисунке 2.13 показана основная страница непосредственно с тестами по русскому языку.

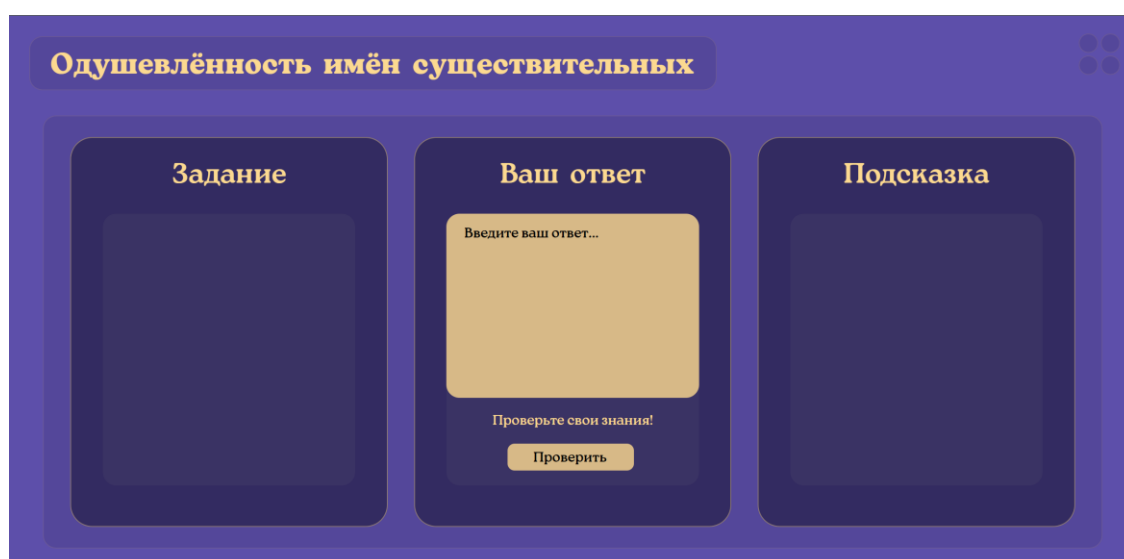


Рисунок 2.13 – Страница с тестами по русскому языку

В таблице 2.1 показана структура HTML-страницы: DOM-элементы, их стили и назначение.

Таблица 2.1 – Элементы графического интерфейса

Название элемента	CSS-стили	Назначение
<body>	background-color: darkslateblue; font-family: 'Roca'; overflow: hidden; display: flex; flex-direction: column; height: 100vh;	Общий фон страницы, организация компоновки
<header>	display: flex; width: 100%; height: 80px;	Заголовок страницы
#legend	background-color: rgba(0, 0, 0, 0.1); color: #d6b674; font-weight: 800; font-size: 2.118rem; border-radius: 15px; padding: 8px 24.5px 8px 22.5px;	Название темы задания
.Dots	display: flex; width: 50px; height: 50px; flex-wrap: wrap; justify-content: space-around; margin-top: 12px; position: absolute; right: 17px;	Бургерная кнопка
.Dot	width: 20px; height: 20px; background-color: rgba(0, 0, 0, 0.1); border-radius: 50%; border: 1px solid rgba(222, 188, 118, 0.1);	Составляющие бургерной кнопки

Продолжение таблицы 2.1

<code><main></code>	<code>display: flex; align-items: center; height: calc(95vh - 110px); background-color: rgba(0, 0, 0, 0.1); width: 95%; margin: auto; border-radius: 15px;</code>	Основной контейнер контента.
<code>.field</code>	<code>background-color: rgba(0, 0, 0, 0.4); height: 90%; width: 30%; border-radius: 25px; border: 1px solid rgba(222, 188, 118, 0.4); display: flex; ..</code>	Блок задания, ответа, подсказки.
<code>.field__name</code>	<code>color: #d6b674; font-size: 2rem; margin-top: 3.8%; font-weight: bold;</code>	Название блока (Задание, Ваш ответ)
<code>.field__block</code>	<code>background-color: #d6b5740a; width: 80%; height: 70%; margin-top: -20px; border-radius: 15px; display: flex; justify-content: center; flex-wrap: wrap;</code>	Контейнер для текста и ввода
<code>.field__block-text</code>	<code>width: 87.5%; margin-top: 5%;</code>	Текст внутри блока задания
<code>.input</code>	<code>background-color: #d6b574cc; width: 100%; height: 65%; border-radius: 15px; padding: 3% 0 0 7%; text-align: left; color: black;</code>	Поле для ввода ответа
<code>.ToF</code>	<code>margin-top: -4.75px; width: 100%; text-align: center;</code>	Сообщение о результате теста

Продолжение таблицы 2.1

<code>.sendButton</code>	<code>background-color: #d6b574cc; width: 50%; height: 10%; border-radius: 8px; text-align: center; color: black; margin-right: 1.5%;</code>	Кнопка отправки ответа
<code>.sendButton__text</code>	<code>padding-right: 1.5%; margin-top: 1.5%;</code>	Текст внутри кнопки отправки

2.5 Схема программного средства

На рисунках 2.14 – 2.16 представлены схемы настройки и работы системы клиент-сервер.

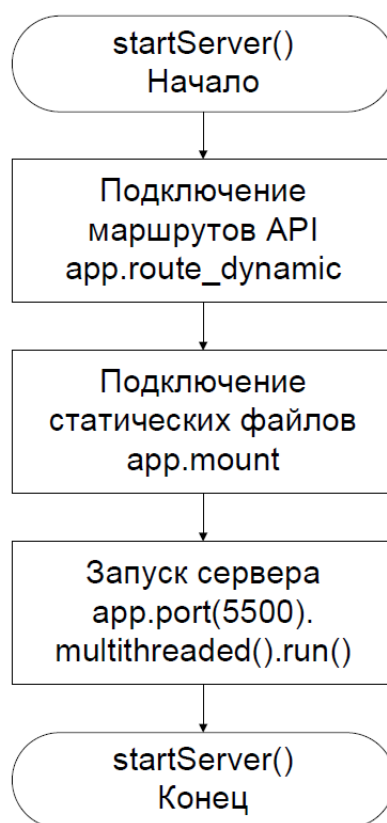


Рисунок 2.14 – Создание сервера

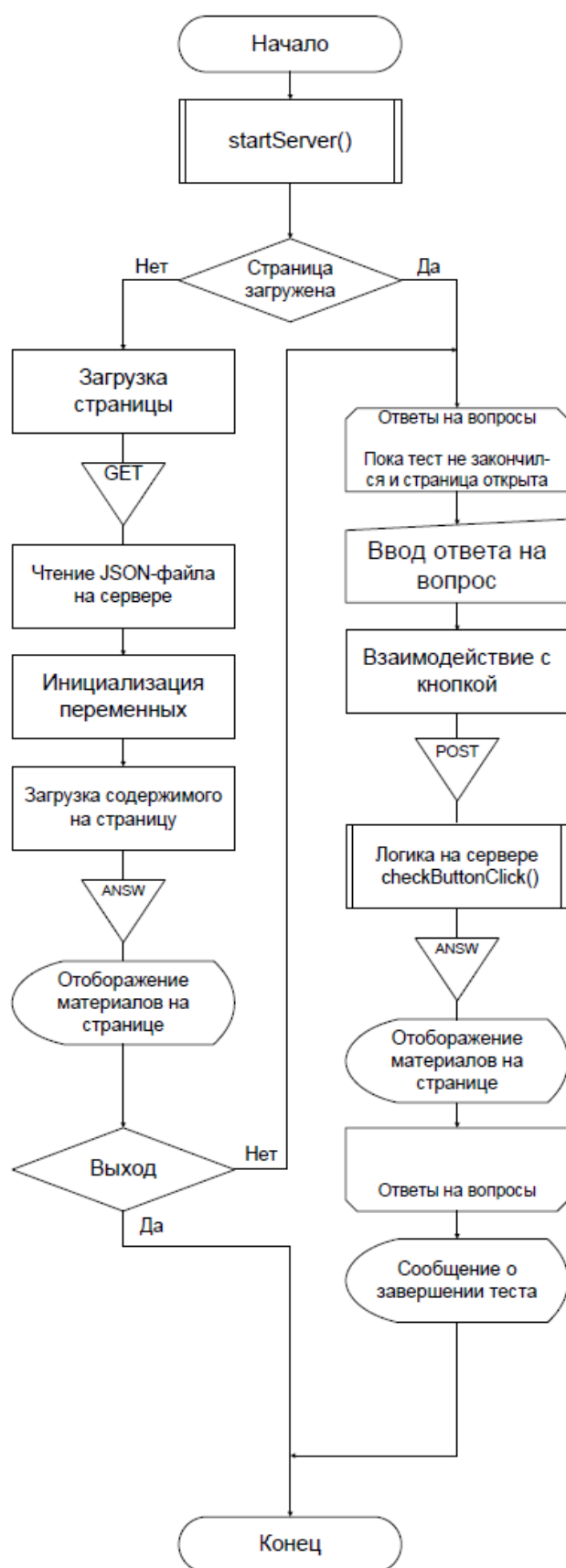


Рисунок 2.15 – Схема работы системы сервер-клиент

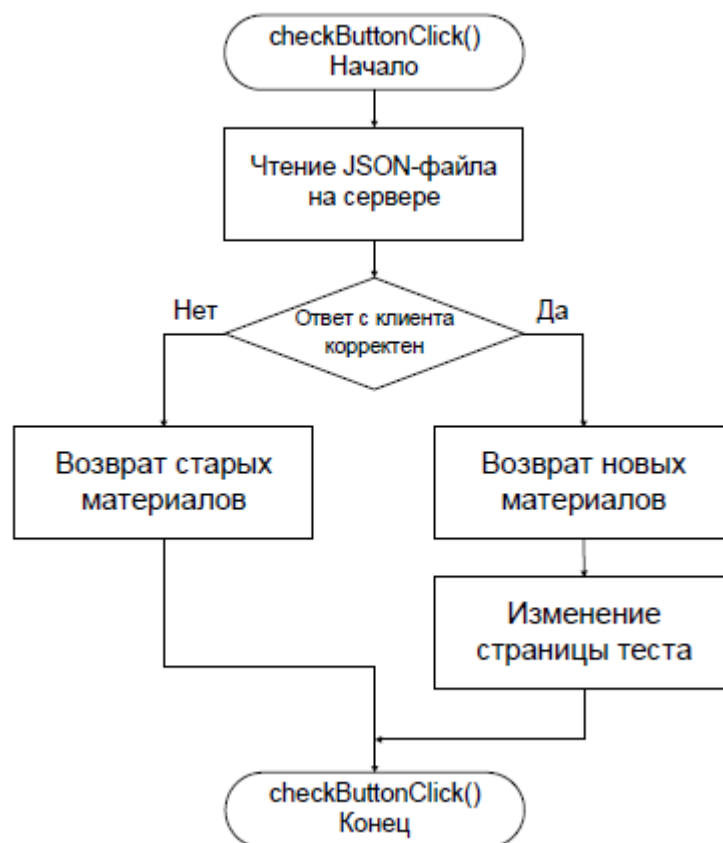


Рисунок 2.16 – Логика обработки на сервере

3 ТЕСТИРОВАНИЕ И ПРОВЕРКА РАБОТОСПОСОБНОСТИ

3.1 Загрузка страницы

При загрузке страницы ожидается, что будет сделан GET-запрос на сервер для получения необходимого материала из JSON-файла. После все материалы должны отобразиться на странице. Как показано на рисунках 3.1 – 3.3, тест пройден успешно.

```
"pages": [  
  {  
    "task": "Слово снеговик является одушевлённым или неодушевлённым? (В ответ запишите прилагательное в форме среднего рода с большой буквы)",  
    "answer": "Одушевлённое",  
    "rule": "Существительные, родительный и винительный падежи которых в форме множественного числа совпадают, являются одушевлёнными"  }  
]
```

Рисунок 3.1 – Материалы в формате JSON

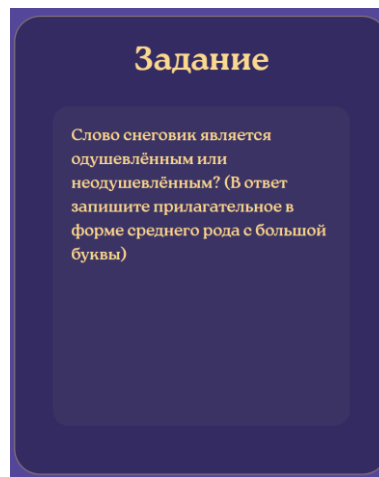


Рисунок 3.2 – Возвращённое с сервера задание теста

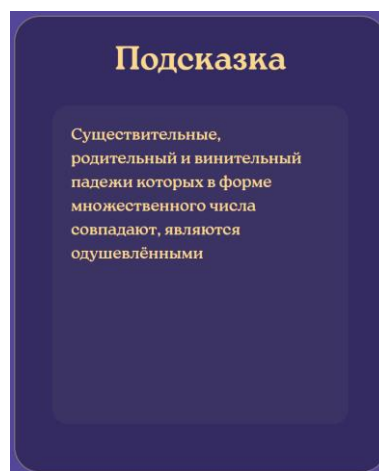


Рисунок 3.3 – Возвращённая с сервера теория к тесту

3.2 Ввод ответа

Ожидается, что, пока поле ввода не затронуто пользователем, в нём будет отображаться фраза «Введите ваш ответ». Как только пользователь осуществляет ввод, фраза должна исчезать, а затем, после расфокусировки, снова появляться, если введённый текст отсутствует (равен «»). На рисунках 3.4 – 3.6 продемонстрировано успешное прохождение теста.

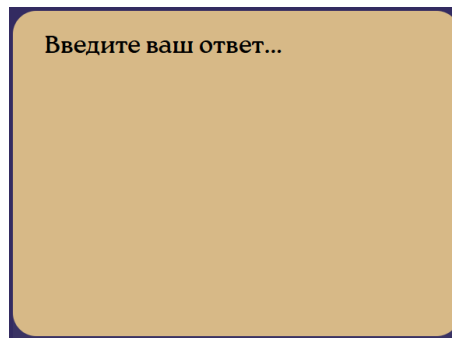


Рисунок 3.4 – Поле ввода до фокусировки или после расфокусировки пользователем



Рисунок 3.5 – Поле во время ввода

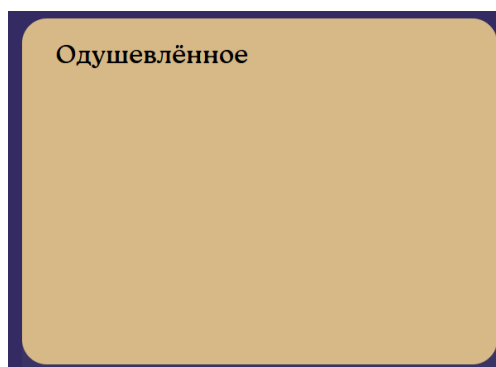


Рисунок 3.6 – Поле ввода при наличии текста после расфокусировки

3.3 Отправка ответа

При нажатии на кнопку «Проверить» делается POST-запрос на сервер для обработки содержимого запроса (см. подразделы 3.4, 3.5), а поле ввода очищается, что показано на рисунке 3.7

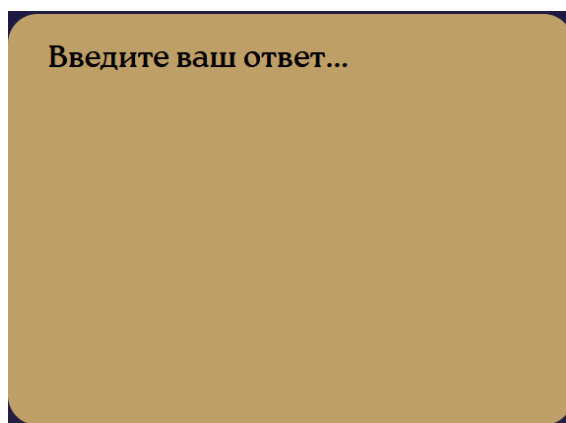


Рисунок 3.7 – Поле ввода после отправки ответа

3.4 Неверный ответ

Если в ходе проверки на сервере оказалось, что ответ, отправленный пользователем, неверный, содержимое задания и теории не меняется, а вместо фразы «Проверьте свои знания!» появляется «Неверно, попробуйте ещё раз». Успешное прохождение теста показано на рисунках 3.8 – 3.10.

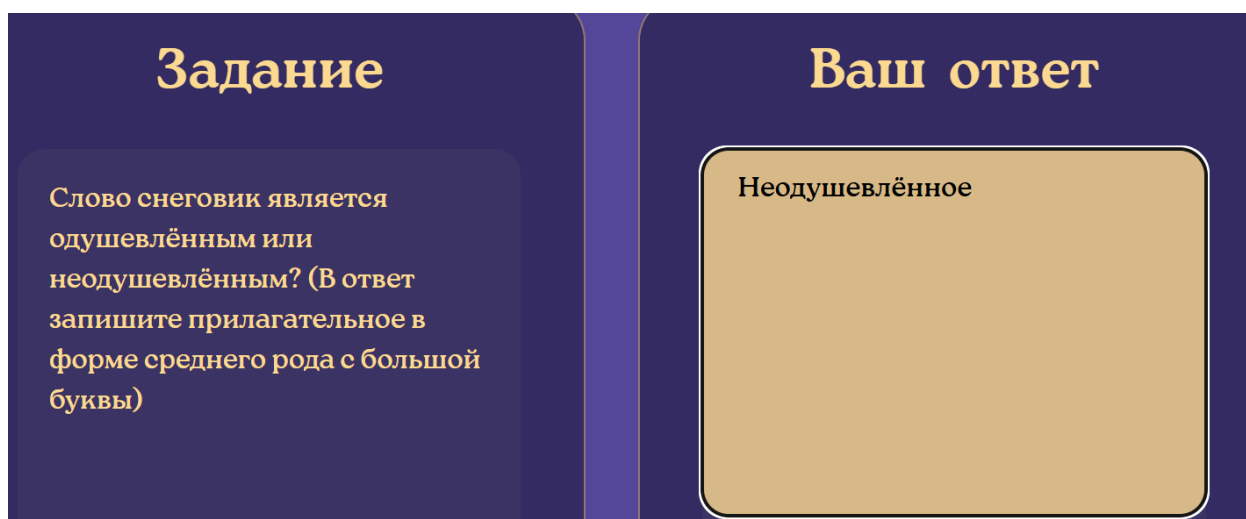


Рисунок 3.8 – Ввода неверного ответа

```
"task": "Слово снеговик является одушевлённым или неодушевлённым? (В ответ запишите прилагательное в форме среднего рода с большой буквы)",  
"answer": "Одушевлённое",
```

Рисунок 3.9 – Корректный ответ в JSON-файле

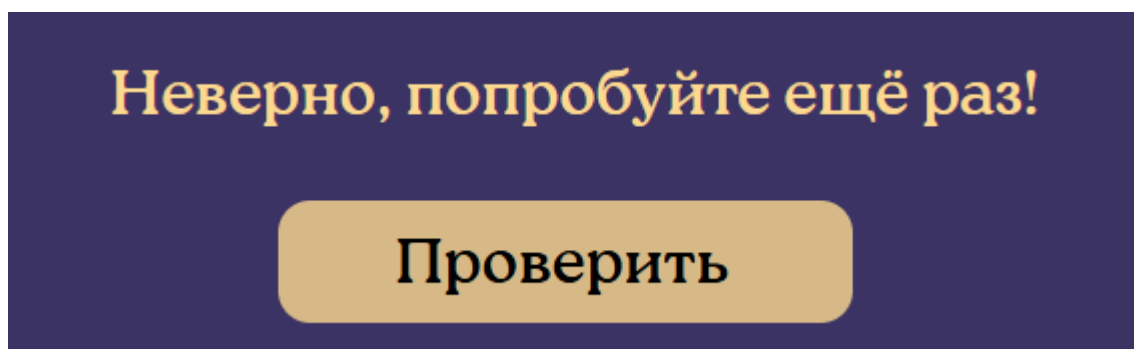


Рисунок 3.10 – Сообщение о неверном ответе

3.5 Правильный ответ

Если в ходе проверки на сервере оказалось, что ответ, отправленный пользователем, корректный, содержимое задания и теории меняется (происходит переход на следующую страницу), а вместо фразы «Проверьте свои знания!» появляется «Правильно! Новое задание!». Успешное прохождение теста показано на рисунках 3.11 – 3.14.

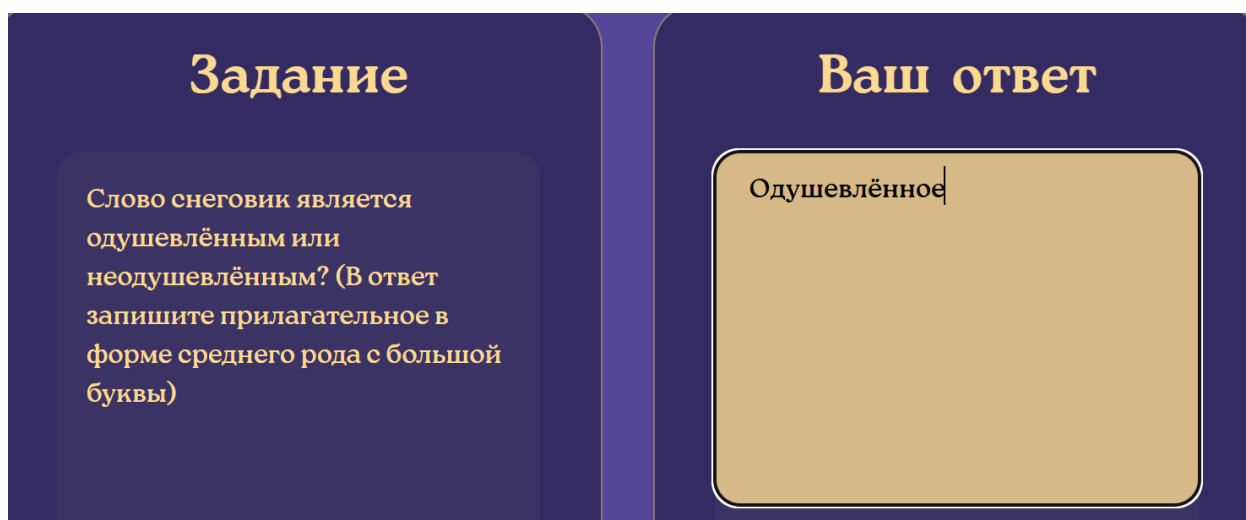


Рисунок 3.11 – Ввод правильного ответа

```

"pages": [
  {
    "task": "Слово снеговик является одушевлённым или неодушевлённым? (В ответ запишите прилагательное в форме среднего рода с большой буквы)",
    "answer": "Одушевлённое",
    "rule": "Существительные, родительный и винительный падежи которых в форме множественного числа совпадают, являются одушевлёнными"
  },
  {
    "task": "Слово труп является одушевлённым или неодушевлённым? (В ответ запишите прилагательное в форме среднего рода с большой буквы)",
    "answer": "Неодушевлённое",
    "rule": "Существительные, именительный и винительный падежи которых в форме множественного числа совпадают, являются неодушевлёнными"
  },
  {
    "task": "Слово валет является одушевлённым или неодушевлённым? (В ответ запишите прилагательное в форме среднего рода с большой буквы)",
    "answer": "Одушевлённое",
    "rule": "Картонные и шахматные фигуры являются одушевлёнными именами существительными"
  }
]

```

Рисунок 3.12 – Содержимое JSON с правильным ответом и следующими страницами

Задание	Ваш ответ	Подсказка
Слово труп является одушевлённым или неодушевлённым? (В ответ запишите прилагательное в форме среднего рода с большой буквы)	Введите ваш ответ...	Существительные, именительный и винительный падежи которых в форме множественного числа совпадают, являются неодушевлёнными

Рисунок 3.13 – Изменение содержимого (переход на следующую страницу)

Правильно! Новое задание!

Проверить

Рисунок 3.14 – Сообщение о правильном ответе

3.6 Завершение теста

После прохождения теста в полях «Задание» и «Подсказка» должно об этом сообщаться. Фраза «Правильно! Новое задание!» должна меняться на «Вы прошли тест!». Сервер не должен прекращать работу при повторных нажатиях кнопки «Проверить», что соответствует действительности.

Результаты успешного прохождения теста можно наблюдать на рисунках 3.15 – 3.16.



Рисунок 3.15 – Вид полей «Задание» и «Подсказка»

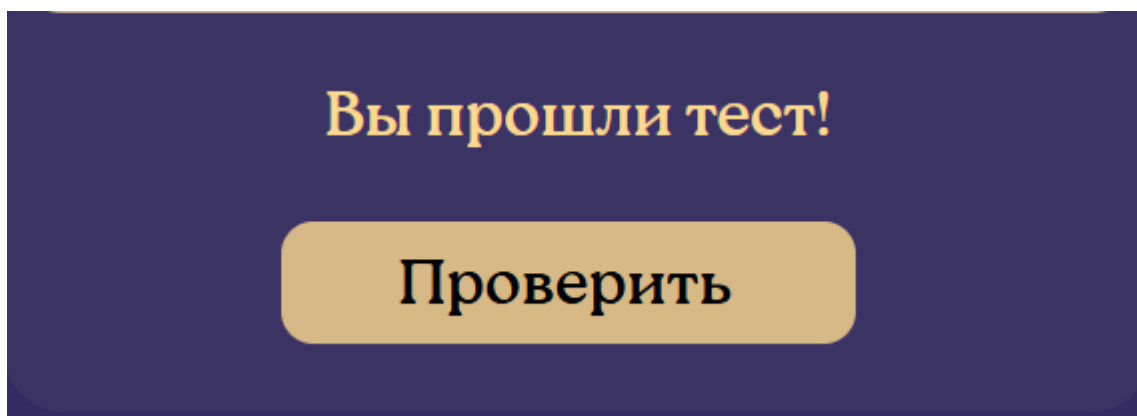


Рисунок 3.16 – Изменение сообщения

4 РУКОВОДСТВО ПО ИСПОЛЬЗОВАНИЮ

4.1 Запуск сервера

Запустить сервер возможно после сборки «С++»-файлов, а затем запуска исполнительного файла (см. рисунки 4.1, 4.2).

```
mkdir build && cd build  
cmake ..  
cmake --build . --config Release
```

Рисунок 4.1 – Сборка «С++»-файлов

```
./server/app.exe
```

Рисунок 4.2 – Запуск сервера

4.2 Открытие страницы в браузере

В поисковой строке браузера необходимо ввести <http://localhost:5500/startPage/russianSections/morphology/noun/nounTask1/nounTask1.html> (см. рисунок 4.3).

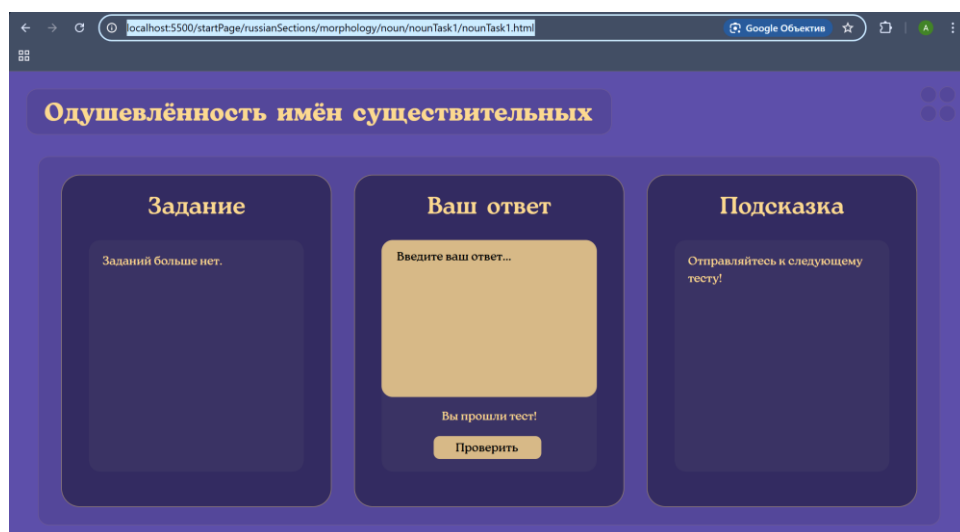


Рисунок 4.3 – Загрузка страницы в браузере

ЗАКЛЮЧЕНИЕ

Разработка веб-сервера для образовательной платформы по русскому языку на базе фреймворка Crow позволила решить задачу создания интерактивной системы тестирования. В процессе выполнения проекта были изучены и применены современные технологии серверной разработки, обработки данных и сетевого взаимодействия.

Ключевыми результатами стали реализация обработки пользовательских ответов на тесты, синтаксический анализ JSON-файлов с учебными материалами и динамическая загрузка контента. Сервер успешно обеспечивает двустороннюю связь с клиентом, корректно обрабатывает запросы и предоставляет обратную связь пользователям в режиме реального времени.

В ходе разработки был сделан акцент на расширяемость системы. Использование таких библиотек, как `nlohmann/json`, `httplib` и `Gumbo`, обеспечило гибкость в работе с различными форматами данных и удобную настройку серверных маршрутов. Это позволило повысить устойчивость системы и упростить её дальнейшую модификацию.

Проведённая работа продемонстрировала важность правильной архитектуры приложения, внимательной проработки логики обработки данных и чёткого взаимодействия между клиентской и серверной частями. Полученный опыт открыл перспективы для улучшения проекта, включая реализацию системы учётных записей, хранения результатов тестирования и управления образовательными материалами.

Таким образом, выполненная работа стала ценным практическим опытом в области веб-разработки и показала потенциал использования C++ для создания эффективных серверных решений. Полученные знания и навыки могут быть применены в будущих проектах с более сложной архитектурой и расширенным функционалом.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

- [1] Орлов, С. А. Технологии разработки программного обеспечения: учеб. Пособие. – СПб, 2003. – 321 с.
- [2] Майкл Д. Изучаем C++ через программирование игр. – СПб.: Питер. 2015. – 450с.
- [3] Лафоре Р. Объектно-ориентированное программирование в C++. 4-е полное изд. – СПб.: Питер, 2018. – 928 с.
- [4] Мартин Р. Чистый код: создание, анализ и рефакторинг. Библиотека программиста. - СПб.: Питер. 2018. – 464 с.
- [5] Уилсон, С. Принципы проектирования и разработки программного обеспечения, учебн. курс. – СПб, 2003. – 570 с.

ПРИЛОЖЕНИЕ А

(обязательное)

Файл app.cpp

```
#include "crow_all.h"
#include <fstream>
#include <nlohmann/json.hpp>

void startServer() {
    crow::SimpleApp app;

    // Маршруты API
    app.route_dynamic("/api/firstPageOnLoad").methods("GET"_method)([]
(const crow::request& req) {
    return crow::response(404); // Заглушка
    });

    app.route_dynamic("/api/checkButtonClick").methods("POST"_method)(
[])(const crow::request& req) {
    return crow::response(404); // Заглушка
    });

    // Статические файлы
    app.mount("/layout", crow::mustache::load_directory("../layout"));
    app.mount("/client", crow::mustache::load_directory("../client"));

    // Запуск сервера
    app.port(5500).multithreaded().run();
}

int main() {
    startServer();
    return 0;
}
```

ПРИЛОЖЕНИЕ Б

(обязательное)

Файл controller.cpp

```
#include <crow_all.h>
#include <nlohmann/json.hpp>
#include "materialsParams.h"

crow::response firstPageOnLoad() {
    try {
        auto materials = readJSON("Materials.json");
        auto nounTests =
materials["russianSections"]["morphology"]["independentParts"]["noun"]
["tests"];
        auto firstTask = nounTests[0]["pages"][0]["task"];
        auto firstRule = nounTests[0]["pages"][0]["rule"];

        nlohmann::json response;
        response["task"] = firstTask;
        response["rule"] = firstRule;

        return crow::response(response.dump());
    } catch (const std::exception& e) {
        return crow::response(500, e.what());
    }
}

crow::response checkButtonClick(const crow::request& req) {
    try {
        auto materials = readJSON("Materials.json");
        auto nounTests =
materials["russianSections"]["morphology"]["independentParts"]["noun"]
["tests"];
        auto test = nounTests[0];

        auto body = nlohmann::json::parse(req.body);
        int page = body["page"];
        std::string editableInput = body["editableInput"];

        auto isTestFinish = [&](int page, const nlohmann::json& test)
-> bool {
            if (test["pages"].contains(page)) {
                return false;
            }
            nlohmann::json updatedData = {
                {"ToF", "Вы прошли тест!"},
                {"task", "Заданий больше нет."},
                {"rule", "Отправляйтесь к следующему тесту!"},
            };
```

```

        {"page", page}
    };
    return true;
};

auto isCorrectAnswer = [&](int& page, const nlohmann::json&
test) {
    if (editableInput == test["pages"][page]["answer"]) {
        ++page;
        if (!isTestFinish(page, test)) {
            return crow::response(200);
        }
    } else {
        return crow::response(400);
    }
};

    return crow::response(200);
} catch (const std::exception& e) {
    return crow::response(500, e.what());
}
}

```

ПРИЛОЖЕНИЕ В

(обязательное)

Файл materialsParams.cpp

```
#include <iostream>
#include <fstream>
#include <nlohmann/json.hpp>

nlohmann::json readJSON(const std::string& fileName) {
    std::ifstream file(fileName);
    if (!file) {
        throw std::runtime_error("No chance to read " + fileName + "
file");
    }

    nlohmann::json jsonData;
    file >> jsonData;
    return jsonData;
}
```

ПРИЛОЖЕНИЕ Г (обязательное)

Файл firstPageOnLoad.cpp

```
#include <iostream>
#include <fstream>
#include <sstream>
#include <string>
#include <vector>
#include <stdexcept>
#include <nlohmann/json.hpp>
#include <httpplib.h>
#include <gumbo.h>

// Функция для поиска элементов по классу
std::vector<std::string> findElementsByClass(GumboNode* node, const
std::string& className) {
    std::vector<std::string> elements;

    if (node->type != GUMBO_NODE_ELEMENT) {
        return elements;
    }

    GumboAttribute* classAttr = gumbo_get_attribute(&node-
>v.element.attributes, "class");
    if (classAttr && className == classAttr->value) {
        // Получаем текстовое содержимое элемента
        if (node->v.element.children.length > 0) {
            GumboNode* child = static_cast<GumboNode*>(node-
>v.element.children.data[0]);
            if (child->type == GUMBO_NODE_TEXT) {
                elements.push_back(child->v.text.text);
            }
        }
    }

    // Рекурсивный обход дочерних элементов
    const GumboVector* children = &node->v.element.children;
    for (unsigned int i = 0; i < children->length; ++i) {
        auto childElements =
findElementsByClass(static_cast<GumboNode*>(children->data[i]),
className);
        elements.insert(elements.end(), childElements.begin(),
childElements.end());
    }

    return elements;
}
```

```

// Функция для загрузки данных с сервера
nlohmann::json fetchFromServer(const std::string& serverAddress, const
std::string& endpoint) {
    httpplib::Client client(serverAddress.c_str());
    auto res = client.Get(endpoint.c_str());

    if (!res || res->status != 200) {
        throw std::runtime_error("Ошибка загрузки данных");
    }

    return nlohmann::json::parse(res->body);
}

// Основная функция firstPageOnLoad
void firstPageOnLoad(const std::string& serverAddress, GumboNode*
htmlRoot) {
    try {
        // Запрос к серверу
        auto jsonData = fetchFromServer(serverAddress,
"/api/firstPageOnLoad");

        // Обновляем HTML-страницу
        auto fieldTextElements = findElementsByClass(htmlRoot,
"field__block-text");
        if (fieldTextElements.size() >= 2) {
            fieldTextElements[0] = jsonData["task"];
            fieldTextElements[1] = jsonData["rule"];
        }

        // Вывод для проверки
        std::cout << "Task: " << jsonData["task"] << std::endl;
        std::cout << "Rule: " << jsonData["rule"] << std::endl;

    } catch (const std::exception& e) {
        std::cerr << "Ошибка: " << e.what() << std::endl;
    }
}

int main() {
    std::string serverAddress = "http://localhost:5500";

    // Загрузка HTML-документа
    std::ifstream htmlFile("example.html");
    if (!htmlFile.is_open()) {
        std::cerr << "Не удалось открыть HTML-файл" << std::endl;
        return 1;
    }

    std::stringstream buffer;
    buffer << htmlFile.rdbuf();

```

```
std::string htmlContent = buffer.str();

// Парсинг HTML с помощью Gumbo
GumboOutput* output = gumbo_parse(htmlContent.c_str());

// Выполнение функции firstPageOnLoad
firstPageOnLoad(serverAddress, output->root);

// Освобождение ресурсов
gumbo_destroy_output(&kGumboDefaultOptions, output);

return 0;
}
```

ПРИЛОЖЕНИЕ Д

(обязательное)

Файл checkButtonClick.cpp

```
#include <iostream>
#include <string>
#include <vector>
#include <nlohmann/json.hpp>
#include <httpplib.h>
#include <gumbo.h>

// Функция для поиска элемента по классу
std::string findElementByClass(GumboNode* node, const std::string&
className) {
    if (node->type != GUMBO_NODE_ELEMENT) {
        return "";
    }

    GumboAttribute* classAttr = gumbo_get_attribute(&node-
>v.element.attributes, "class");
    if (classAttr && className == classAttr->value) {
        // Извлечение текстового содержимого
        if (node->v.element.children.length > 0) {
            GumboNode* child = static_cast<GumboNode*>(node-
>v.element.children.data[0]);
            if (child->type == GUMBO_NODE_TEXT) {
                return child->v.text.text;
            }
        }
    }

    // Рекурсивный поиск
    const GumboVector* children = &node->v.element.children;
    for (unsigned int i = 0; i < children->length; ++i) {
        std::string result =
findElementByClass(static_cast<GumboNode*>(children->data[i]),
className);
        if (!result.empty()) {
            return result;
        }
    }

    return "";
}

// Функция для обработки "click" (отправка запроса)
void onSendButtonClick(const std::string& serverAddress, std::string&
editableInput, int& page) {
```



```

httplib::Client client(serverAddress.c_str());

// Подготовка данных
nlohmann::json dataToSend = {
    {"editableInput", editableInput},
    {"page", page}
};

auto res = client.Post("/api/checkButtonClick",
    dataToSend.dump(),
    "application/json");

if (res && res->status == 200) {
    auto jsonData = nlohmann::json::parse(res->body);

    std::cout << "Ответ сервера:" << std::endl;
    std::cout << "ToF: " << jsonData["ToF"] << std::endl;
    std::cout << "Task: " << jsonData["task"] << std::endl;
    std::cout << "Rule: " << jsonData["rule"] << std::endl;

    // Обновление данных
    editableInput.clear();
    page = jsonData["page"];
} else {
    std::cerr << "Ошибка при отправке данных" << std::endl;
}
}

int main() {
    std::string serverAddress = "http://localhost:5500";

    // Загружаем HTML (симуляция)
    std::ifstream htmlFile("example.html");
    if (!htmlFile.is_open()) {
        std::cerr << "Не удалось открыть HTML-файл" << std::endl;
        return 1;
    }

    std::stringstream buffer;
    buffer << htmlFile.rdbuf();
    std::string htmlContent = buffer.str();
    GumboOutput* output = gumbo_parse(htmlContent.c_str());

    // Ищем элементы
    std::string editableInput = findElementByClass(output->root,
"editableInput");
    std::string sendButton = findElementByClass(output->root,
"sendButton");
    std::string task = findElementByClass(output->root, "field__block-
text");

```

```
        std::string rule = findElementByClass(output->root, "field__block-  
text");  
        int page = 0;  
  
        // Обработка события "click"  
        onSendButtonClick(serverAddress, editableInput, page);  
  
        // Освобождение ресурсов  
        gumbo_destroy_output(&kGumboDefaultOptions, output);  
  
        return 0;  
    }
```

ПРИЛОЖЕНИЕ Е (обязательное)

Файл nounTask1.html

```
<!DOCTYPE html>

<html>

  <head>

    <meta charset="utf-8"/>
    <title>RuL</title>
    <link rel="stylesheet" href="nounTask1.css" type="text/css" />

  </head>

  <body>

    <header>

      <div
id="legend">Одушевлённость&nbsp;&nbsp;&nbsp;имён&nbsp;&nbsp;&nbsp;существительных<
/div>

      <div class="Dots">

        <div class="Dot"></div>
        <div class="Dot"></div>
        <div class="Dot"></div>
        <div class="Dot"></div>

      </div>

    </header>

    <main>

      <div id="question" class="field">

        <div class="field__name">Задание</div>
        <div class="field__block"><div class="field__block-
text"></div></div>

      </div>

      <div id="answer" class="field">

        <div class="field__name">Ваш&nbsp;&nbsp;&nbsp;ответ</div>
```

```

        <div class="field__block">
            <div class="input" id="editableInput"
contenteditable="true" placeholder="Введите ваш ответ...">
                Введите ваш ответ...
            </div>
            <div class="ToF">Проверьте свои знания!</div>
            <div class="sendButton"><div
class="sendButton__text">Проверить</div></div>
        </div>

        <div id="rule" class="field">

            <div class="field__name">Подсказка</div>
            <div class="field__block"><div class="field__block-
text"></div></div>

        </div>

    </main>

    <script src="/firstPageOnLoad.js"></script>
    <script src="/checkButtonClick.js"></script>

</body>

</html>

```

ПРИЛОЖЕНИЕ Ж (справочное)

Файл nounTask1.css

```
@import url("../../../../../../fonts/fonts.css");

body {
    background-color: darkslateblue;
    font-family: 'Roca';
    overflow: hidden;
    display: flex;
    flex-direction: column;
    height: 100vh;
}

header {
    display: flex;
    width: 100%;
    height: 80px;
}

main {
    display: flex;
    align-items: center;
    height: calc(95vh - 110px);
    background-color: rgba(0, 0, 0, 0.1);
    width: 95%;
    margin: auto;
    margin-top: 25px;
    border-radius: 15px;
    border-color: rgba(222, 188, 118, 0.1);
    border-style: solid;
    border-width: 1px;
}

#legend {
    background-color: rgba(0, 0, 0, 0.1);
    color: #d6b674;
    font-weight: 800;
    font-size: 2.118rem;
    border-radius: 15px;
    padding-left: 22.5px;
    padding-right: 24.5px;
    padding-bottom: 8px;
    margin-top: 15px;
    margin-left: 15px;
    position: absolute;
    border-color: rgba(222, 188, 118, 0.1);
```

```

    border-style: solid;
    border-width: 1px;
}

.Dots {
    display: flex;
    width: 50px;
    height: 50px;
    flex-wrap: wrap;
    justify-content: space-around;
    margin-top: 12px;
    position: absolute;
    right: 17px;
}

.Dot {
    width: 20px;
    height: 20px;
    background-color: rgba(0, 0, 0, 0.1);
    border-radius: 50%;
    border-color: rgba(222, 188, 118, 0.1);
    border-style: solid;
    border-width: 1px;
}

.field {
    background-color: rgba(0, 0, 0, 0.4);
    height: 90%;
    width: 30%;
    border-radius: 25px;
    border-color: rgba(222, 188, 118, 0.4);
    border-style: solid;
    border-width: 1px;
}

#question {
    margin-left: 2.5%;
    margin-right: 0;
}

#answer {
    margin-left: 2.5%;
    margin-right: 2.5%;
}

#rule {
    margin-left: 0;
    margin-right: 2.5%;
}

.field {

```

```

        display: flex;
        flex-wrap: wrap;
        justify-content: center;
        align-items: space-between;
    }

    .field__name {
        color: #d6b674;
        font-size: 2rem;
        margin-top: 3.8%;
        font-weight: bold;
    }

    .field__block {
        background-color: #d6b5740a;
        width: 80%;
        height: 70%;
        margin-top: -20px;
        border-radius: 15px;
        display: flex;
        justify-content: center;
        flex-wrap: wrap;
        color: #d6b674;
        text-align: left;
    }

    .input {
        background-color: #d6b574cc;
        width: 100%;
        height: 65%;
        border-radius: 15px;
        padding-top: 3%;
        text-align: left;
        padding-left: 7%;
        color: black;
    }

    .ToF {
        margin-top: -4.75px;
        width: 100%;
        text-align: center;
    }

    .sendButton {
        background-color: #d6b574cc;
        width: 50%;
        height: 10%;
        border-radius: 8px;
        text-align: center;
        /*color: rgb(68, 52, 170);*/
        color: black;
    }

```

```
        margin-right: 1.5%;
    }

    .sendButton__text {
        padding-right: 1.5%;
        margin-top: 1.5%;
    }

    .field__block-text {
        width: 87.5%;
        margin-top: 5%;
    }

    #editableInput:focus .placeholder,
    #editableInput.typing .placeholder {
        visibility: hidden; /* Скрываем текст, когда ввод активен */
        margin-left: 6%;
        margin-top: 2%;
    }
}
```