**МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ**

федеральное государственное бюджетное образовательное учреждение   
высшего образования

**«УЛЬЯНОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»**

Факультет *ИСТ* Кафедра *ИВК*

К ЗАЩИТЕ ДОПУСТИТЬ

Зав. кафедрой

/ С.К. Киселев /

подпись инициалы, фамилия

« » 20 24 г.

**ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА**

Тема *Интеллектуальный помощник для работы с электронной библиотекой*

Обучающийся \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/ *Х.А. Нгуен*

подпись инициалы, фамилия

Обозначение работы *ВКР-УлГТУ-09.03.02-17/1175-2024* Группа *ЦИСТбв-51*

Направление подготовки *09.03.02 «Информационные системы и технологии»*

код, наименование

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Руководитель ВКР / *В.М. Кандаулов* /

подпись, дата инициалы, фамилия

Консультанты:

*экономический раздел* / *М.В. Рыбкина* /

наименование раздела подпись, дата инициалы, фамилия

Ульяновск, 2024

**МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ**

федеральное государственное бюджетное образовательное учреждение   
высшего профессионального образования

**«УЛЬЯНОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»**

Факультет  *ИСТ*  Кафедра  *ИВК* .

Направление подготовки *Информационные системы и технологии* .

|  |  |
| --- | --- |
|  | УТВЕРЖДАЮ |
|  | Зав. кафедрой |
|  | \_\_\_\_\_\_\_\_\_\_/\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ / |
|  | подпись инициалы, фамилия |
|  | «\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2024 г. |

ЗАДАНИЕ

**на выпускную квалификационную работу**

обучающемуся  *Нгуену Хыу Ану* курса  *5*  группы  *ЦИСТбв-51*

фамилия, имя, отчество

1. Тема работы *Интеллектуальный помощник для работы с электронной библиотекой*утверждена приказом по университету №\_\_\_\_\_\_\_ от «\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2024 г.
2. Срок сдачи обучающимся законченной ВКР «*22*» *июня* 2024 г.
3. Исходные данные к работе *разработать систему* *персональной электронной библиотеки работающей в локальной сети*
4. Содержание пояснительной записки *задание на создание системы; информационное, алгоритмическое, программное обеспечение системы; тестирование системы*
5. Перечень графического материала (с точным указанием обязательных чертежей) *схема алгоритма обмена информацией между серверной частью и API ИИ, схема алгоритма добавления электронной книги в электронную библиотеку.*
6. Календарный график работы над ВКР на весь период (с указанием сроков выполнения и содержания отдельных этапов)

|  |  |  |
| --- | --- | --- |
| № этапа | Содержание этапа | Срок выполнения |
| 1 |  |  |
| 2 |  |  |
| 3 |  |  |
| 4 |  |  |
|  |  |  |

1. Консультанты

|  |  |  |  |
| --- | --- | --- | --- |
| Раздел | Ф.И.О. консультанта | Подпись, дата | |
| Задание выдал | Задание принял |
| *Экономический* |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

1. Дата выдачи задания «\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2024 г.

Руководитель\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/\_\_\_\_\_\_\_\_\_/ *В.М. Кандаулов* /

должность, учёная степень, ученое звание подпись инициалы, фамилия

Задание принял к исполнению \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ / *Х.А. Нгуен* /

подпись обучающегосяинициалы, фамилия

Содержание

[CПИСОК ИСПОЛЬЗОВАННЫХ ОБОЗНАЧЕНИЙ, СОКРАЩЕНИЙ И ТЕРМИНОВ 7](#_Toc165719405)

[ВВЕДЕНИЕ 8](#_Toc165719406)

[1 ТЕХНИЧЕСКОЕ ЗАДАНИЕ НА СОЗДАНИЕ СИСТЕМЫ 10](#_Toc165719407)

[1.1 Назначение и цели создания системы 10](#_Toc165719408)

[1.2 Характеристика объекта автоматизации 10](#_Toc165719409)

[1.2.1 Общее описание 10](#_Toc165719410)

[1.2.2 Состав и порядок функционирования 10](#_Toc165719411)

[1.3 Общие требования к системе 11](#_Toc165719412)

[1.3.1 Требования к структуре и функционированию системы 11](#_Toc165719413)

[1.3.2 Дополнительные требования 12](#_Toc165719414)

[1.4 Требования к функциям, выполняемым системой 12](#_Toc165719415)

[1.5 Требования к видам обеспечения 12](#_Toc165719416)

[1.5.1 Требования к информационному обеспечению 12](#_Toc165719417)

[1.5.2 Требования к алгоритмическому обеспечению 12](#_Toc165719418)

[1.5.3 Требования к программному обеспечению 13](#_Toc165719419)

[1.6 Анализ аналогичных разработок 14](#_Toc165719420)

[1.6.1 Calibre 14](#_Toc165719421)

[1.6.2 Kavita 15](#_Toc165719422)

[1.6.3 Koodo 16](#_Toc165719423)

[2 Информационное обеспечение системы 17](#_Toc165719424)

[2.1 Выбор средств управления данными 17](#_Toc165719425)

[2.2 Проектирование базы данных 17](#_Toc165719426)

[2.2.1 Концептуальная схема базы данных 18](#_Toc165719427)

[2.2.2 Внутренняя схема базы данных 18](#_Toc165719428)

[2.3 Проектирование файлов данных 18](#_Toc165719429)

[2.4 Организация сбора, передачи, обработки и выдачи информации 18](#_Toc165719430)

[3 Программное обеспечение системы 19](#_Toc165719431)

[3.1 Структура программного обеспечения и функции его компонентов 19](#_Toc165719432)

[?3.1.1 Серверная часть (backend) 19](#_Toc165719433)

[?3.1.2 Клиентская часть (frontend) 24](#_Toc165719434)

[3.2 Выбор компонентов программного обеспечения 32](#_Toc165719435)

[3.2.1 Операционная система 32](#_Toc165719436)

[3.2.2 Инструментальное средство разработки и язык программирования 32](#_Toc165719437)

[3.2.3 Вспомогательное программное обеспечение 32](#_Toc165719438)

[3.3 Разработка прикладного программного обеспечения 32](#_Toc165719439)

[3.3.1 Структура прикладного программного обеспечения 33](#_Toc165719440)

[3.3 Инсталляция и настройка 34](#_Toc165719441)

[4 ТЕСТИРОВАНИЕ СИСТЕМЫ 36](#_Toc165719442)

[4.1 Условия и порядок тестирования 36](#_Toc165719443)

[4.2 Исходные данные для контрольных примеров 36](#_Toc165719444)

[4.3 Результаты тестирования 36](#_Toc165719445)

[5 ЭКОНОМИЧЕСКАЯ ЧАСТЬ 37](#_Toc165719446)

[ЗАКЛЮЧЕНИЕ 38](#_Toc165719447)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 39](#_Toc165719448)

[ПРИЛОЖЕНИЕ А 40](#_Toc165719449)

# CПИСОК ИСПОЛЬЗОВАННЫХ ОБОЗНАЧЕНИЙ, СОКРАЩЕНИЙ И ТЕРМИНОВ

СУБД – система управления базами данных.

ПО – программное обеспечение.

Фреймворк (Software Framework) – программная платформа, определяющая структуры программного обеспечения.

API – интерфейс программирования приложения (Application Programming Interface).

.NET Core – модульная платформа для разработки программного обеспечения с открытым исходным кодом.

С# – объектно-ориентированный язык программирования общего назначения, разработанный компанией Microsoft.

UML – аббревиатура от Unified Modeling Language, язык графический описания для объектного моделирования в области разработки программного обеспечения, моделирования бизнес процессов, системного проектирования и отображения организационных структур.

# ВВЕДЕНИЕ

В современном мире набирает всю большую популярность искусственный интеллект. Его практически можно встретить во многих отраслях деятельности, таких как финансы, промышленность, торговле, образование, системы безопасности, робототехника и т.п. В быту можно встретить в обработке фотоизображений, распознаваний образов на фото, улучшение и восстановление качества фотоизображений, и транскрипция или перевод с одного естественного языка на другой. Всё благодаря развитию мощностей компьютерной техники, увеличение количества транзисторов на кристаллах интегральных схем, а также появлении искусственных нейронных сетей и генеративных ИИ. Также популярности ИИ способствовало появление чат-бота на базе больших языковых моделей LLM ChatGPT от компании OpenAI.

На данном этапе развития ИИ может использоваться в качестве инструмента автоматизации некоторых процессов. В некоторых сферах деятельности она может полностью заменить человека, в основном это сфера предоставления услуг конечным потребителям, такие как кассиры, банковские служащие, официанты и т.п. Также ИИ может использоваться в качестве автопилота транспортных средств, и самый известный пример — это электромобили от американской компаний Tesla. В военном деле ИИ используется для управления дронами, роботами-сапёрами и автономными комплексами вооружения.

В сфере разработки программного обеспечения искусственный интеллект, в основном на базе нейронных сетей, помогают в написании кода и быстрого поиска нужной информации. Такие ИИ-ассистенты не могут полностью заменить программиста человека, так как код всё также нужно анализировать на корректность работы логики в предметной области и на ошибки, особенно связанные с кибербезопасностью. В основном публичные ИИ-инструменты, такие как GitHub Copilot, обучающиеся на коде опубликованные в открытом доступе, у которых в большинстве случаев не было проведено ревью кода (code review), может генерировать устаревший код, так как код используемых библиотек может измениться со временем и/или генерировать код с неявными уязвимостями, связанные с безопасностью.

Из основных недостатков искусственного интеллекта являются

* Безопасность и конфиденциальность данных: использование ИИ может создать уязвимости в системах безопасности и привести к утечкам конфиденциальной информации.
* Потеря рабочих мест: автоматизация процессов с использованием ИИ может привести к сокращению рабочих мест, особенно в тех сферах, где задачи могут быть легко автоматизированы.
* Предвзятость и дискриминация: ИИ может унаследовать предвзятость из данных, на которых он обучается, что может привести к несправедливым или дискриминационным решениям.
* Ответственность и прозрачность: в некоторых случаях сложно объяснить, почему ИИ принял определенное решение, что затрудняет определение ответственности в случае ошибок или несправедливых действий.
* Этические вопросы: использование ИИ вызывает много этических вопросов, включая проблемы конфиденциальности, принципа справедливости и вопросы о том, кто несет ответственность за действия ИИ.
* Угроза безопасности: развитие ИИ также может создать новые угрозы для безопасности, такие как создание автономных систем, способных принимать вредные решения.

Цель данной дипломной работы является получение навыков разработки прикладного обеспечения, навыков работы с современными инструментами разработки, интеграции искусственного интеллекта в информационную систему.

# 1 ТЕХНИЧЕСКОЕ ЗАДАНИЕ НА СОЗДАНИЕ СИСТЕМЫ

## 1.1 Назначение и цели создания системы

Назначением системы:

Система предназначена для хранения электронных книг, в основном технической литературы, а также других документов в едином хранилище с возможностью их просмотра, редактирования описания книг, отслеживания активности пользователя при работе с документами.

Местом внедрения является внутренняя локальная сеть пользователя.

Цель создания системы:

* уменьшение времени доступа к работе с электронными книгами и документами;
* снижение нагрузки на дисковые системы устройств пользователя.

## 1.2 Характеристика объекта автоматизации

Объектом автоматизации является неавтоматизированное рабочее место читателя в локальной сети.

### 1.2.1 Общее описание

Объектом автоматизации нынешней системы является процесс работы с файлами книг и документов. Доступ к файлам можно получить только внутри локальной сети на сетевом хранилище по протоколу SMB, организованный с помощью программного обеспечения роутера.

### 1.2.2 Состав и порядок функционирования

Функционирование системы включает в себя набор, следующий процессов:

* реализация общего доступа к хранилищу с книгами;
* отслеживание прогресса чтения для каждой книги;
* ведения словаря слов и фраз.

Алгоритм работы с файлами электронных книг следующий:

* открываем сетевой общий диск через любой файловый менеджер поддерживающий протокол обмена данными SMB;
* находим определенный файл книги;
* читаем его, находя неизвестные слова записываем их и по необходимости переводим, если пользователю это потребуется, через любые сторонние сервисы через веб-браузер.

## 1.3 Общие требования к системе

### 1.3.1 Требования к структуре и функционированию системы

Система должна представлять собой программный комплекс, в состав которого должны входить следующие подсистемы:

* подсистема сервера;
* подсистема клиента.

Подсистема серверной части должна состоять из следующих компонентов:

учёт списка

Пользовательский интерфейс подсистемы клиентской части программного обеспечения должен состоять из:

* страница подключения к серверу;
* страница списка документов;
* страница просмотра документа.

Требования к режимам функционирования системы предполагают:

* обеспечения корректного подключения к серверу;
* обеспечение корректной работы на нескольких устройствах;
* обеспечение корректной работы отслеживания активности пользователя;
* обеспечение сохранения файлов документов.

Перспективы развития системы предполагают:

* возможность веб-поиска по выбранному тексту;
* возможность вывода обобщения по выбранному тексту или всего документа;
* возможность преобразование книги в другой файловый формат;
* возможность нахождения данных о книге в сети Интернет;
* возможность отображения советов и рекомендации;
* улучшения удобства чтения.

### 1.3.2 Дополнительные требования

Техническое обслуживание и администрирование программного обеспечения должно выполняться специалистами, имеющими соответствующую квалификацию и навыки выполнения работ.

Система является свободно распространяемой и не требует лицензии.

## 1.4 Требования к функциям, выполняемым системой

В системе должны быть реализованы следующие функции:

* функция предоставления списка документов;
* функция просмотра документа;
* функция перевода текста.

Первоочередность реализации функций указана по порядку.

## 1.5 Требования к видам обеспечения

### 1.5.1 Требования к информационному обеспечению

В системе должны быть предусмотрены следующие требования:

* наличие системы управления базами данных PostgreSQL;
* реализация средств контроля вводимых данных;
* требования к архивации и шифровании отсутствуют.

### 1.5.2 Требования к алгоритмическому обеспечению

Необходимо реализовать алгоритм обмена информацией между серверной частью и API искусственного интеллекта.

Необходимо реализовать алгоритм добавление электронной книги в электронную библиотеку.

### 1.5.3 Требования к программному обеспечению

Требования к организации интерфейса:

* пользовательский интерфейс должен быть удобен, интуитивно понятен и прост в использовании;
* на главной странице расположить список недавно открытых электронных книг, кнопка добавления новых книг из системы пользователя, сортировка книг по разным критериям, поиск по разным критериям;
* на странице аутентификации пользователя расположить ввод пользователя и пароля в виде ПИН-кода, кнопка входа в систему, кнопка регистрации нового пользователя;
* на странице просмотра книги должно быть кнопки навигации по разным страницам, управляющие элементы изменения масштаба, возможность выделять текст, копирование в буфер обмена выделенного текста, также возможность нахождения перевода выделенного текста, нахождение определения выделенного текста;
* страница редактирования метаданных книги таких как название, теги, описание.

Требования к написанию программного кода:

* язык программирования C#, JavaScript/TypeScript;
* веб-фреймворк ASP.NET, Angular;
* язык разметки HTML;
* текстовый редактор Visual Studio Code;
* интеграционная среда программирования JetBrains Rider.

Требования к пользователю системы:

* операционная система Windows NT не ниже 10 версии или дистрибутив Linux с любой графической оболочкой.

## 1.6 Анализ аналогичных разработок

Существует некоторое количество систем, выполняющих схожие разрабатываемой задачи. Их основной функционал направлен на предоставление электронной библиотеки и возможности их чтения.

Таблица №1 - Сравнительный анализ аналогичных программных продуктов

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Название программного продукта** | **Calibre** | **Kavita** | **Koodo** | **Разработанный программный продукт** |
| Многопользовательская система | - | + | - | + |
| Интуитивно понятный интерфейс | + | + | + | + |
| Веб-интерфейс | - | + | + | + |
| Настольное приложение | + | - | + | +  (в виде PWA) |
| Трекинг проведенного времени | - | - | - | + |
| Лицензия исходного кода программы | GPL 3.0 | GPL 3.0 | AGPL 3.0 | MIT |

### 1.6.1 Calibre

Calibre – кроссплатформенное, свободное и отрытое программное обеспечение для чтения и хранения электронных книг в электронной библиотеке для настольных систем на базе Windows, MacOS и Linux, разработанное в 2006 году. В данный момент продолжает поддерживаться. Распространяется под лицензией GNU GPL версии 3. Последняя версия программы – 7.9.0 (19 апреля 2024). Обладает довольно большим функционалом: можно конвертировать книги в разные форматы, собирать статьи с новостных сайтов и создавать из них электронную книгу, может искать электронные книги на разных сайтах магазинах и т.п. Большая часть кода программы написана на языке программирования Python и с помощью фреймворка QT. Визуальное представление программы Calibre представлена на рисунке 1.

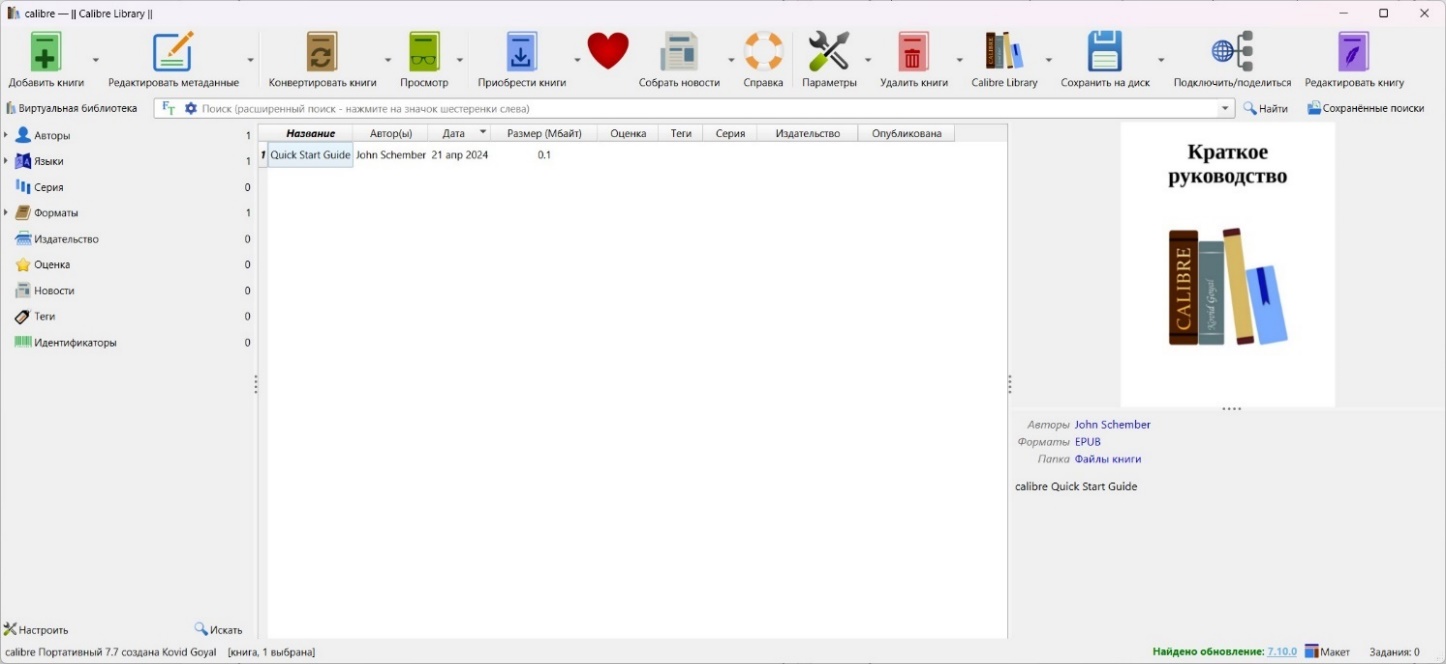


Рисунок 1 – Окно программы Calibre

### 1.6.2 Kavita

Kavita – открытое программное обеспечение для чтения и хранения книг в электронной библиотеке, созданное в 2020 году. Сделана в виде web-приложения. Написана на языках программирования C# и Typescript с помощью фреймворком ASP.NET и Angular. На данный момент ещё находится в разработке. Последняя версия – 0.8.1 (23 апреля 2024). Исходный код программы распространяется под лицензией GNU GPL версии 3. Визуальное представление программы Kavita изображена на рисунке 2.

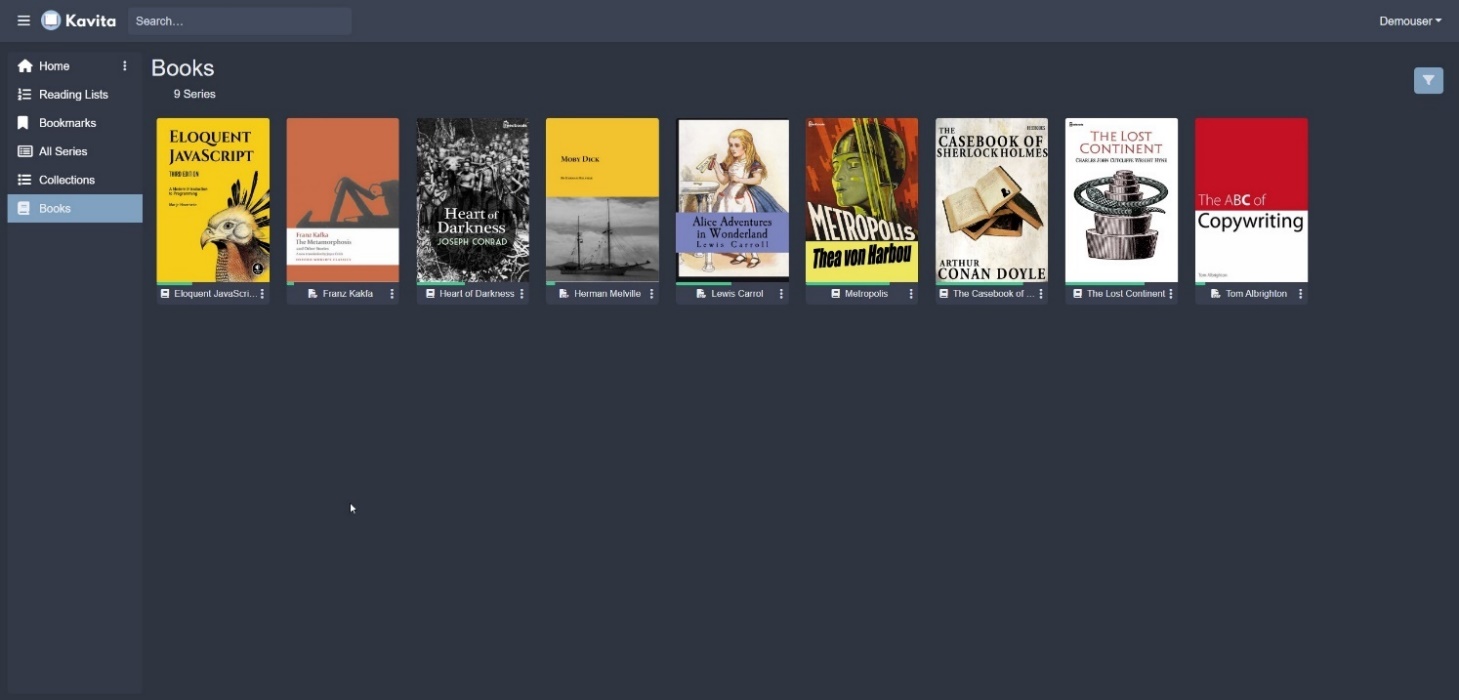


Рисунок 2 – Веб-страница программы Kavita

### 1.6.3 Koodo

Koodo – свободное программное обеспечение для чтения и хранения книг, разработанная в 2020 году. Написана на языке программирования JavaScript с использованием фреймворка React. Распространяется в виде настольного приложения на базе Electron и может работать как веб-приложение. Исходный код распространяется под лицензией AGPL версии 3. Последняя версия – 1.6.6 (6 апреля 2024). Визуальное представление программы Koodo изображена на рисунке 3.

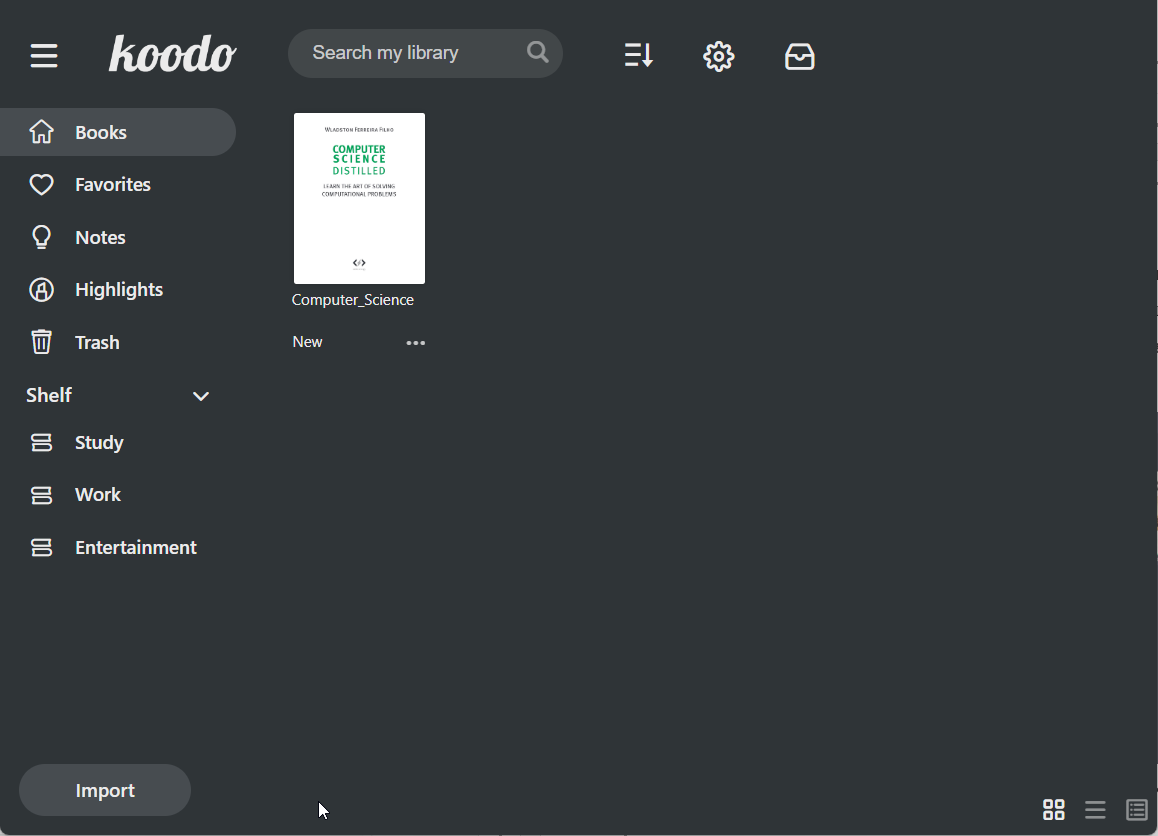


Рисунок 3 – Страница программы Kodoo

# 2 Информационное обеспечение системы

## 2.1 Выбор средств управления данными

На данный момент существуют множество систем управлений баз данных (СУБД). Одними из популярных СУБД на 2023 год считаются исходя опроса 76634 всех участников на сайте StackOverflow:

* PostgreSQL (45,55%).
* MySQL (41,09%).
* SQLite (30,9%).
* MongoDB (25,52%).
* Microsoft SQL Server (25,45%).

Проценты обозначают количество людей желающих пользоваться данными СУБД и тех людей, которые пользовались ими в течении последнего года.

## 2.2 Проектирование базы данных

Схема базы данных BookApi представлена на рисунке №.

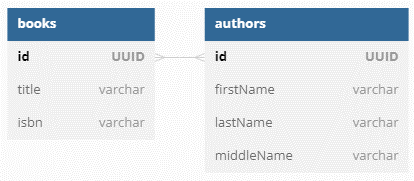


Рисунок № – Схема базы данных BookApi

Классы Book и Author с описанием полей описаны в таблице 1 и 2 соответственно.

Таблица 1 – Класс Book

|  |  |  |
| --- | --- | --- |
| **Название поля** | **Тип данных** | **Описание** |
| id (ИД) | GUID | Уникальный ключ, идентификатор |
| title (Название) | string | Название электронной книги |
| isbn | string? | Код ISBN, необязательное |

Таблица 2 – Класс Author

|  |  |  |
| --- | --- | --- |
| **Название поля** | **Тип данных** | **Описание** |
| id (ИД) | GUID | Уникальный ключ, идентификатор |
| firstName (Имя) | string | Имя автора |
| lastName (Фамилия) | string | Фамилия автора |
| middleName (Отчество) | string? | Отчество автора, необязательное |

### 2.2.1 Концептуальная схема базы данных

-

### 2.2.2 Внутренняя схема базы данных

-

### 2.3 Проектирование файлов данных

-

### 2.4 Организация сбора, передачи, обработки и выдачи информации

-

# 3 Программное обеспечение системы

## 3.1 Структура программного обеспечения и функции его компонентов

### ?3.1.1 Серверная часть (backend)

Для написания сервисов BookApi, FileStorage и Gateway был использован язык программирования C# и фреймворк ASP.NET Core. Для работы сервисов gRPC нужно установить пакет gRPC.AspNetCore. Для работы клиента gRPC нужно ещё установить три пакета – Google.Protobuf, Grpc.Net.Client, Grpc.Tools. Также службы BookApi и FileStorage используют ещё дополнительную библиотеку ORM Entity Framework и SQLite для хранения данных о электронных книгах и их авторах. Проекты с заготовленным шаблоном проекта генерировались посредством консольной утилиты *dotnet* поставляемый с *.NET SDK*. Структура проекта с сервисами представлена на рисунке 3.

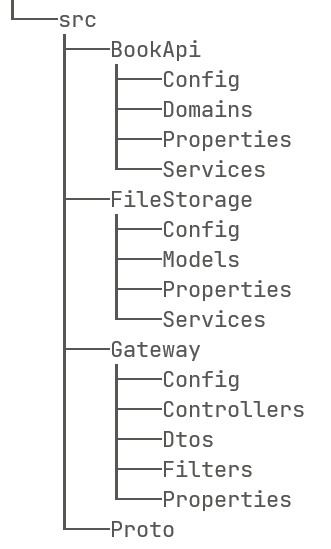


Рисунок 3 – Структура проекта

В директории «src/Proto» хранятся файлы protobuf файлы.

Далее представлены файлы каждого проекта в формате XML, которая показывает версию .NET, список зависимостей и пакетов.

Все gRPC сервисы в BookApi определяются файлами с расширением «.proto». Эти файлы затем генерируются в код на C# с помощью пакета Grpc.Tools при сборке проекта.

Служба Gateway является шлюзом между BookApi и WebClient. Она предоставляет HTTP API для WebClient и при получении запроса от клиента передаёт их BookApi, используя классы BookServiceClient и AuthorServiceClient. Эти классы также автоматически генерируются пакетом Grpc.Tools. Список конечных точек (endpoint) HTTP API приложения Gateway:

* GET /books – список записей книг;
* POST /books – создание записи книги;
* PUT /books – обновление записи книги;
* DELETE /books/{id:Guid} – удаление записи книги;
* POST /books/upload/{id:Guid} – загрузка файла книги;
* POST /books/download/{id:Guid} – скачивание файла книги;
* GET /authors – список записей авторов книг;
* POST /authors – добавление нового автора;
* PUT /authors – обновление параметров существующего автора;
* DELETE /authors/{id:Guid} – удаление автора;
* GET / – переадресует по URL клиентского приложения.

На рисунке 4 API представлено в виде UML диаграммы.

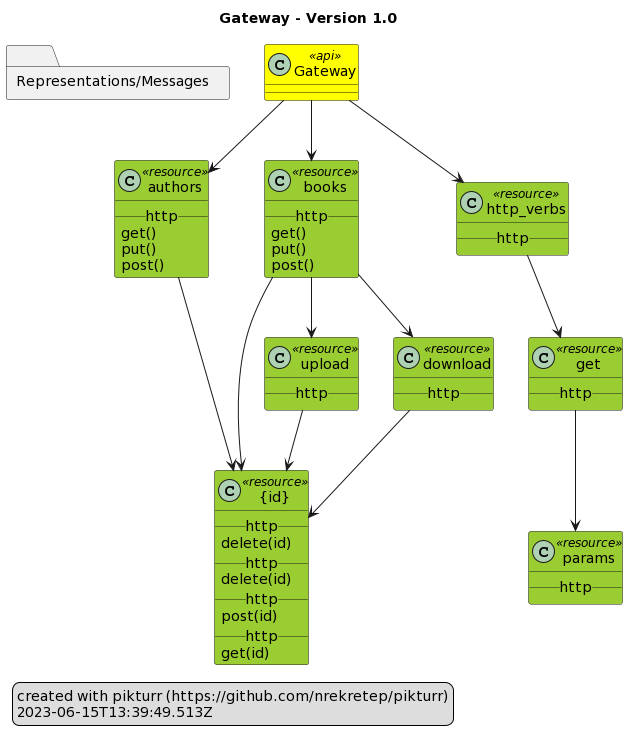


Рисунок 4 – Диаграмма Gateway API

За API отвечают классы-контроллеры BookController и AuthorController.

При переходе на страницу «/», Gateway перенаправляет на страницу веб-интерфейса. Путь до веб-интерфейса, также задаётся через файл конфигурации.

Пути URL к другим сервисам находятся в файле «appsettings.json» в корневых директориях каждого проекта. Его можно в случае необходимости перезаписать через командную строку, либо через переменные окружения.

Диаграмма классов в проекте BookApi представлена на рисунке 5.

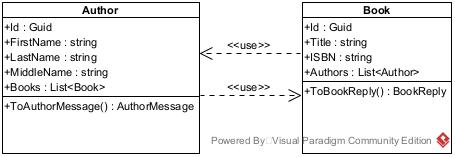


Рисунок 5 – Диаграмма классов BookApi

Классы AuthorMessage и BookReply сгенерированы пакетом Grpc.Tools и используются для передачи сообщений между сервисами.

Сервис BookApi общается со службой FileStorage для хранения файлов книг. Если поступает запрос к BookApi на скачивание или загрузку файла книги, то он обращается к нему.

Загружаемый или скачиваемый файл книги разбивается на коллекцию блоков данных (chunk) и передаётся в виде потока через gRPC. Сервис FileStorage сохраняет файлы на диск по пути, заданный файлом конфигурации. После загрузки файла FileStorage передаёт обратно сообщение со сгенерированным уникальным идентификатором UUID и потом создаёт объект класса FileEntry. Этот объект затем сохраняется в базу данных.

При удалении записи книги в BookApi, удаляется ещё и запись, вместе с файлом, если существует ещё на диске, в сервисе FileStorage. И вызывается методом DeleteEntry из BookApi.

Просмотреть список конечных точек HTTP можно на странице, сгенерированные при запуске проекта Gateway по HTTP-адресу <https://localhost:8443/swagger> (рисунок 6).

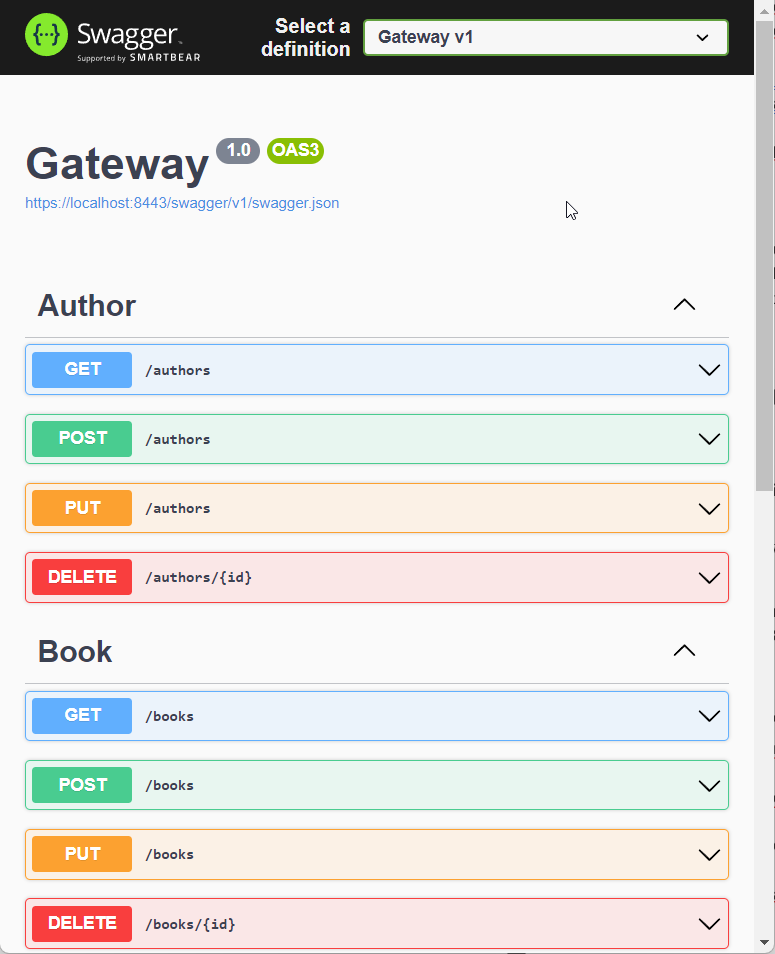


Рисунок 6 – Список конечных точек HTTP

Для сервисов определены следующие номера портов в файле конфигурации каждого проекта «appsettings.json»:

* 8443 для Gateway;
* 12433 для BookApi;
* 11433 для FileStorage.

### ?3.1.2 Клиентская часть (frontend)

Для написания веб-приложения WebClient используется язык программирования TypeScript и фреймворк Angular версии 16. Также в зависимостях этого приложения присутствует Angular Material, который предоставляет компоненты в стиле Material от Google. Фреймворк использует программный паттерн Dependency Injection для управления зависимостями. Классы, от которых будут зависеть другие компоненты, нужно маркировать атрибутом «@Injectable()» перед объявления класса. Они также автоматически добавляются при создании через консольное приложение *ng*. Под компонентами понимается класс представления, имеющий разметку HTML и файл стилей. Их можно задать всё в одном файле или можно разбить на несколько файлов. Компоненты также можно создавать консольной утилитой *ng*.

Далее представлена структура проекта WebClient в виде дерева на   
рисунке 7.

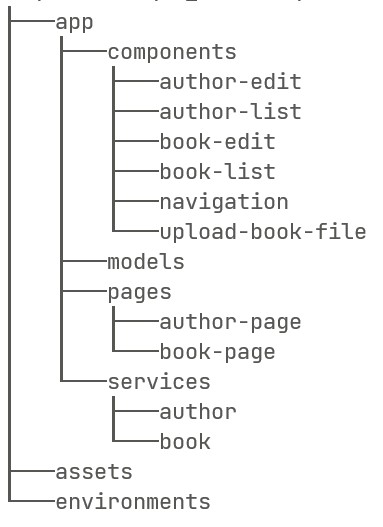


Рисунок 7 – Структура проекта WebClient

Путь к API Gateway сервиса находится в файлах «environment.\*.ts» в директории src/environments. Представлены в виде экспортируемого объекта:

В директории models находятся классы предметной области «book.ts» и «author.ts».

В папке *src/app/services* находятся сервисы для работы с API. Они используются в компонентах из директории *src/app/pages*.

В проекте используется модуль «AppRoutingModule» для возможности переходить на другие страницы, набрав адрес в адресной строке браузера.

В панели навигации только есть две ссылки – на основную страницу (ссылки «Книги») и страницу списков авторов (ссылка «Авторы»). Панель навигации всегда находится в закрытом состояний (рисунок 8).

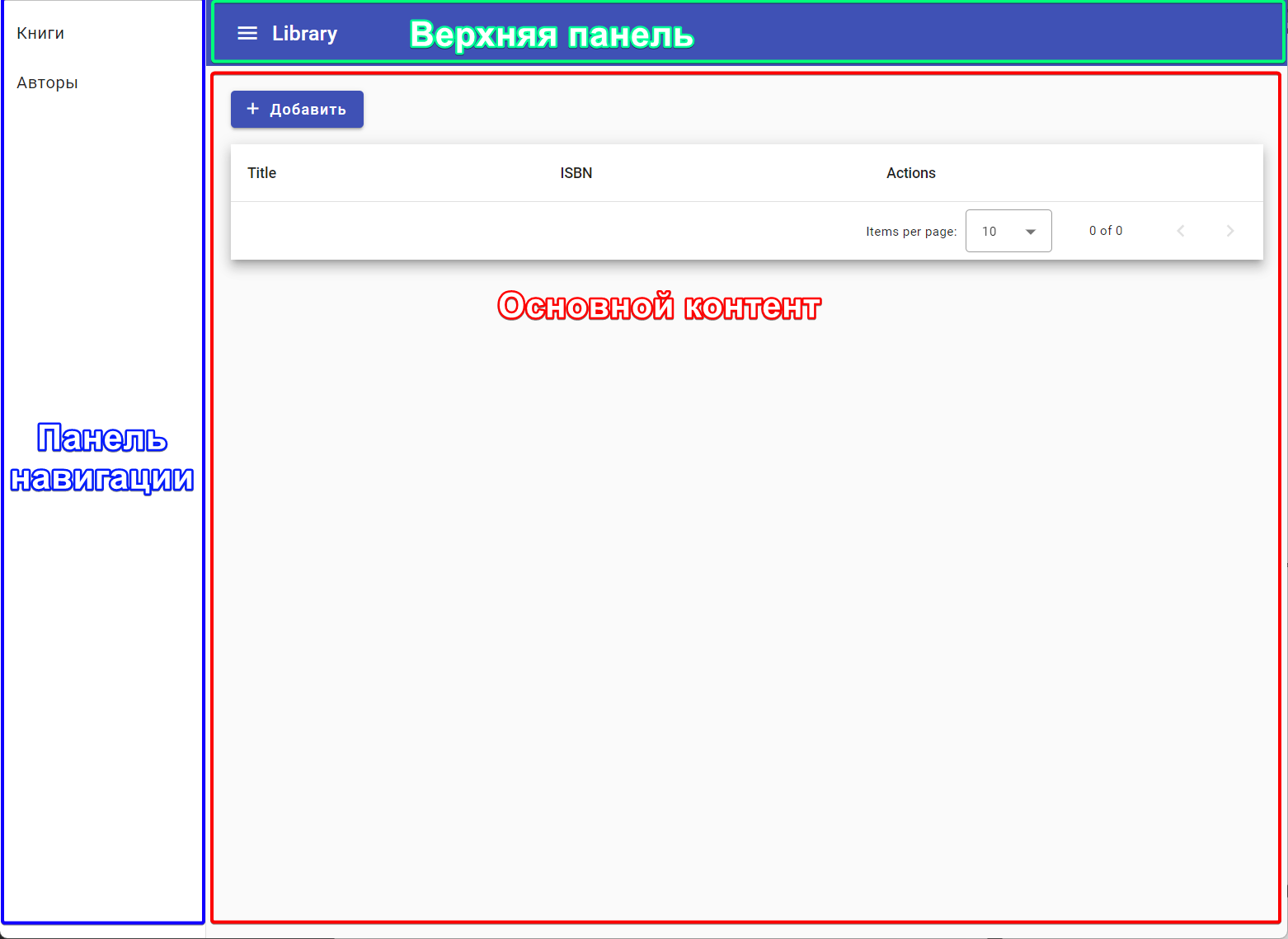


Рисунок 8 – Разметка (layout) интерфейса

В верхней панели отображается кнопка открытия/закрытия панели навигации в виде трёх полосок и название приложения «Library».

На основной странице выводится список книг в виде таблицы (рисунок 9).

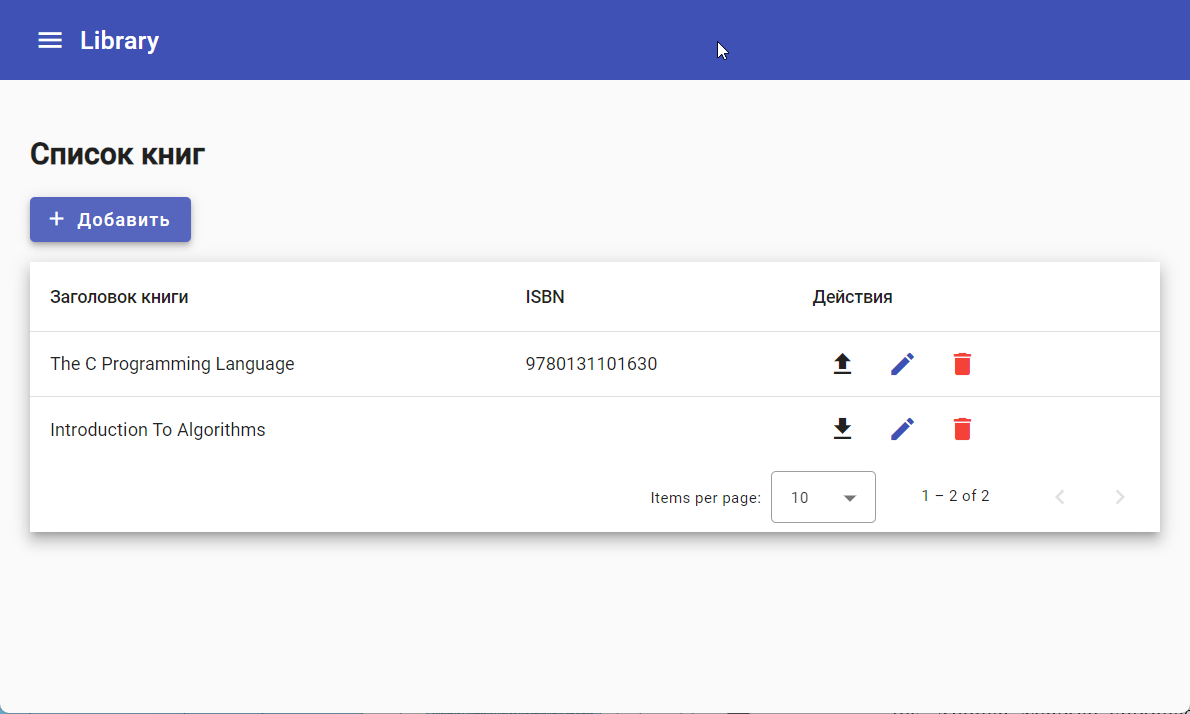


Рисунок 9 – Основная страница клиентской части

Таблица записей книг имеет следующие поля – «Заголовок книги», «ISBN» (Международный стандартный книжный номер) и «Действия». Также есть возможность сортировки по первым двум полям. Это можно сделать, нажав на ячейку в шапке таблицы (рисунок 10).

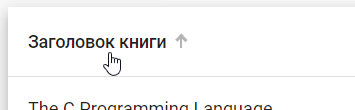
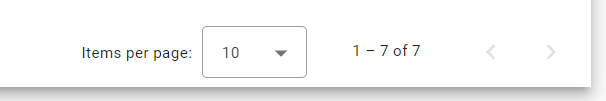


Рисунок 10 – Сортировка по колонкам

У таблицы есть встроенная возможность разбиении строк на страницы (pagination). Количество строк в таблице определена заранее массивом чисел – 10, 20, 50. Она находится под таблицей (рисунок 11).

Рисунок 11 – Панель навигации по страницам

Она состоит из выпадающего меню, в котором можно выбрать отображаемое количество элементов на одной странице, потом идёт диапазон отображаемых элементов от общего количества строк в формате <от> - <до> of <общее\_количество\_строк> и кнопки навигации на следующий и предыдущий.

При нажатии кнопки «Добавить» над таблицей, открывается диалоговое окно для создания новой записи (рисунок 12).

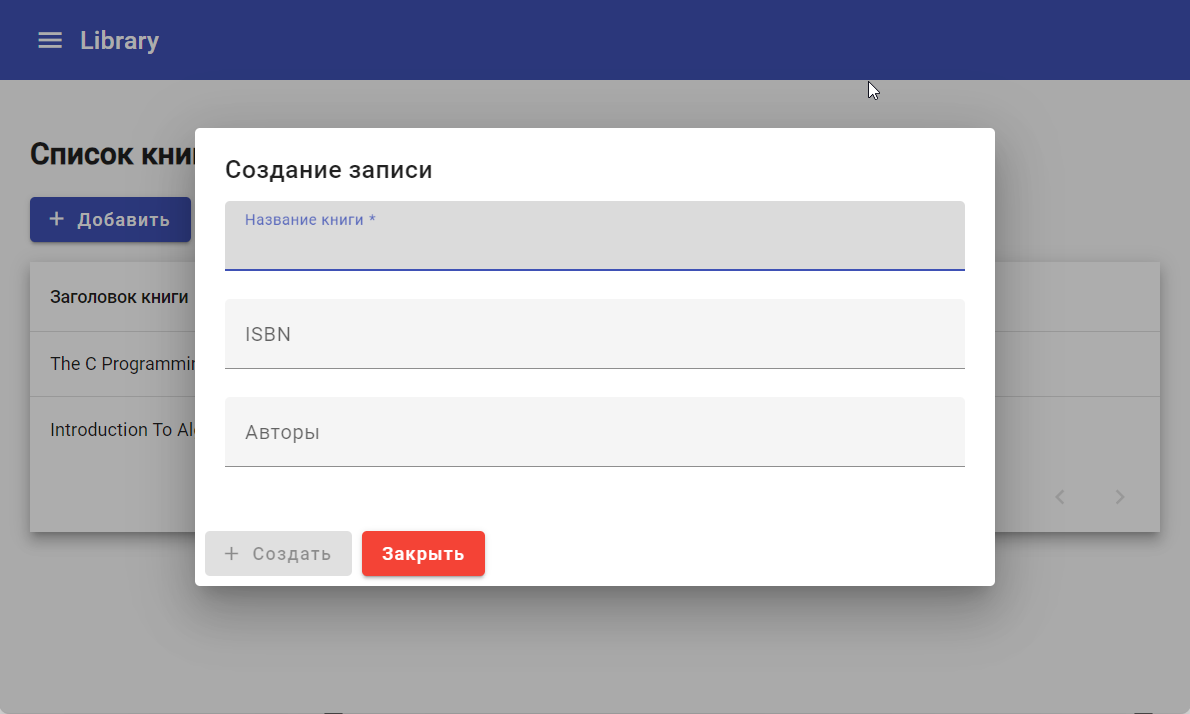
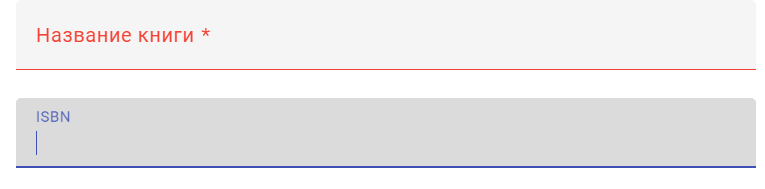


Рисунок 12 – Окно добавления/редактирования записи книги

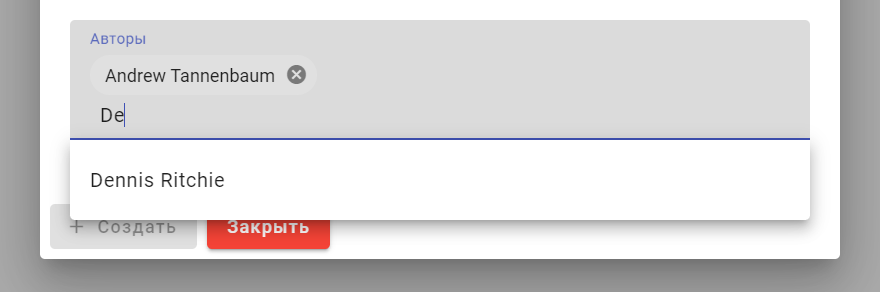
Такое же окно открывается и при редактировании записи нажатием на кнопку с изображением ручки.

Обязательным полем является «Название книги», и она помечена звёздочкой над заголовком поля в верхнем правом углу. Если эта строка будет иметь длину меньше одной символа, то кнопка «Создать» или «Изменить» будет не доступна и не будет окрашена цветом. За валидацию полей в фреймворке Angular отвечают классы валидаторы Validators.

Если пользователь перейдёт к следующему полю без заполненного обязательного поля, то это цвет контура поля и цвет заголовка текстового поля измениться на красный (рисунок 13).

Рисунок 13 – Обязательное поле

Авторов книги можно выбирать несколько. Также можно отфильтровать, набрав в строке нужные символы. Выбранные элементы отображается в виде чипов (chips) (рисунок 14).

Рисунок 14 – Поле «Авторы»

Также из следующих функции есть:

* удаление записи;
* загрузка файлов книг;
* cкачивание файлов книг.

У каждой записи книг есть булево поле «hasFile», которое говорит есть ли файл книги в электронном варианте или нет. Исходя из этого поля клиентское приложение показывает один из двух вариантов кнопок действия – скачивание или загрузка (рисунок 15).

Рисунок 15 – Кнопка скачивания/загрузки

При нажатии на кнопку загрузки открывается диалоговое окно с выбором файла. Для выбора файла нужно нажать на поле и откроется диалоговое окно для выбора файла с вашего компьютера. Без выбранного файла кнопка загрузки файла будет недоступна (рисунок 16).

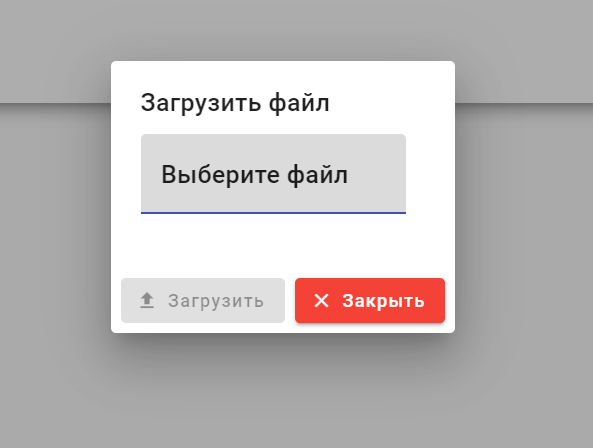


Рисунок 16 – Диалог выбора файла для загрузки

Скачивание файла происходит сразу же при нажатии на кнопку. Для этого используется библиотека file-saver. Она скачивает с API блобы и сохраняет его в виде HTML ссылки и программным способом нажимает на него. И затем открывается диалоговое окно выбора пути для сохранения файла.

Уведомление выпадает либо об успешности операции над какими-либо записи, либо при возникновении ошибки при обращениях к серверной части приложения (рисунок 17).

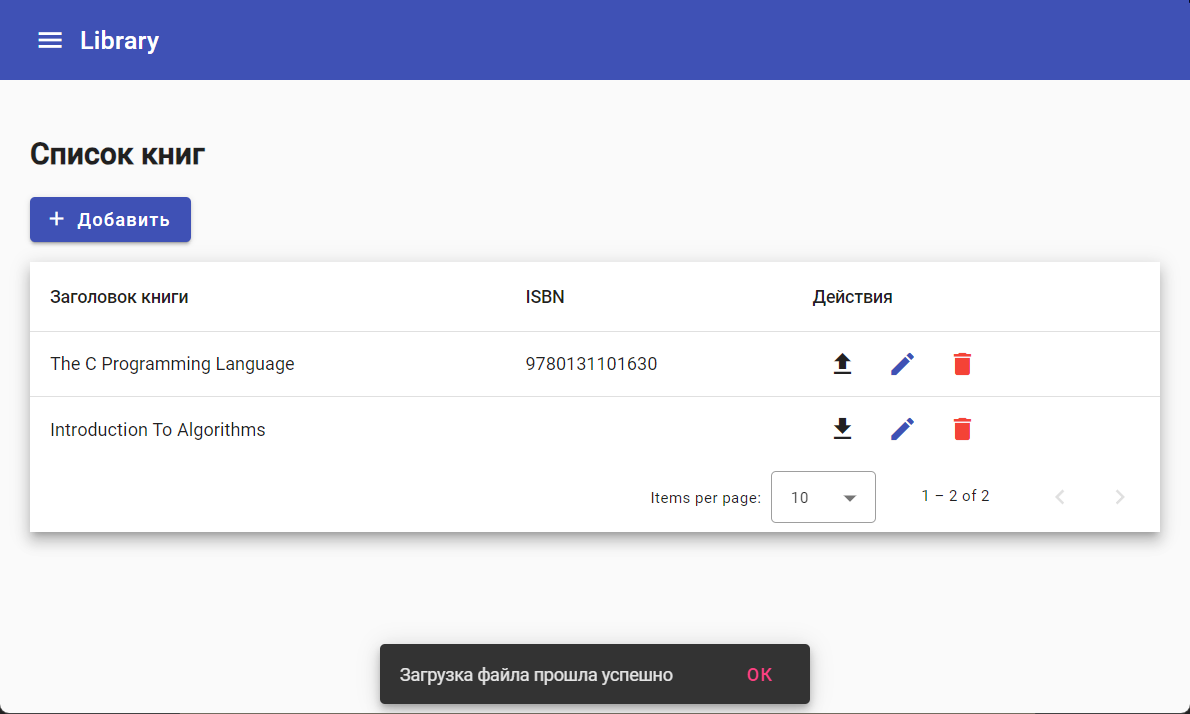


Рисунок 17 – Выпадающее уведомление

Страница со списком авторов книг также представлена в виде таблицы и кнопки добавления новой записи (рисунок 18).

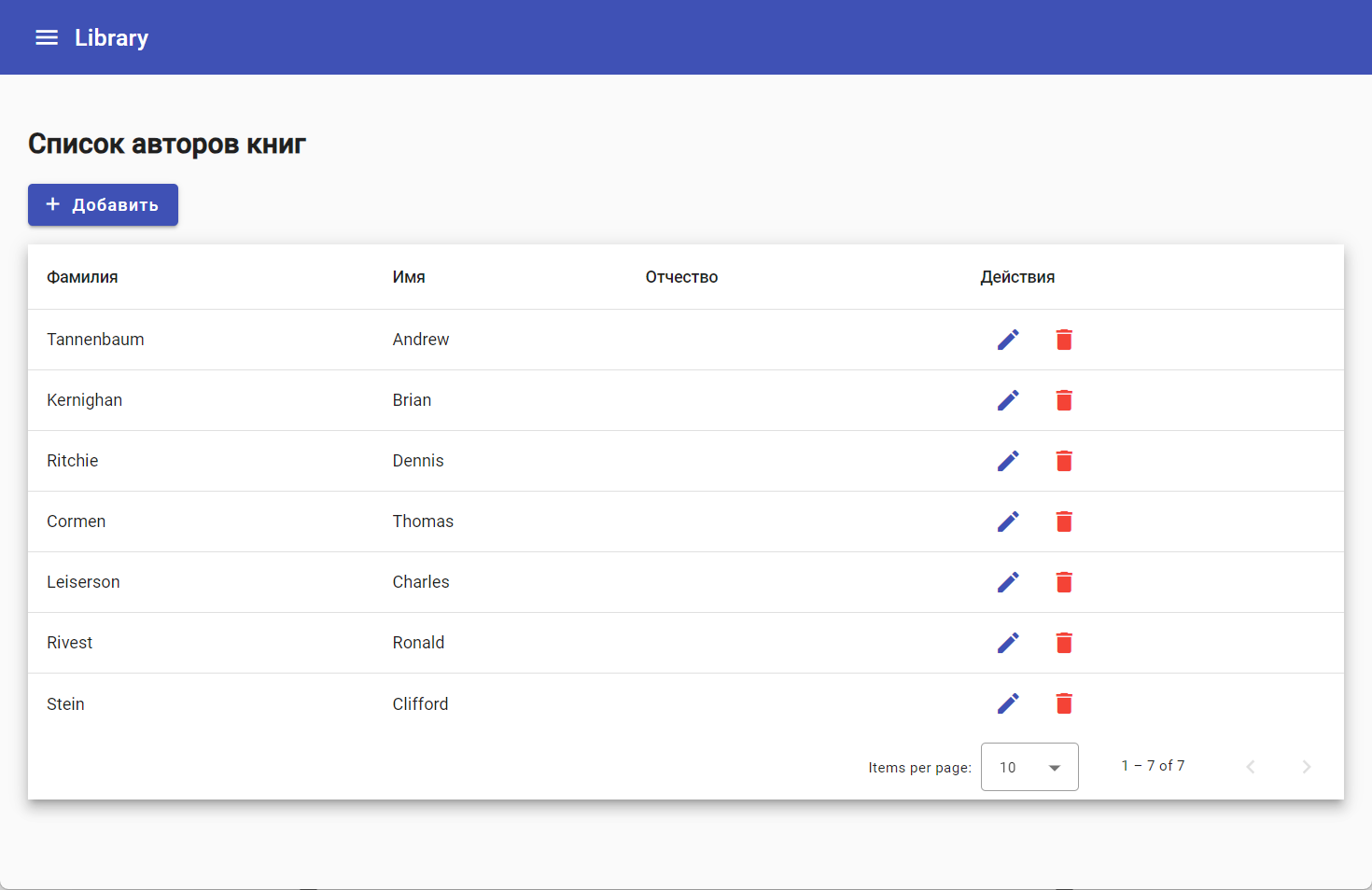


Рисунок 18 – Страница «Авторы»

Поля «Фамилия» и «Имя» являются обязательными для заполнения и заголовки полей отмечены звёздочкой. Эти поля должны иметь не менее двух символов (рисунок 19).

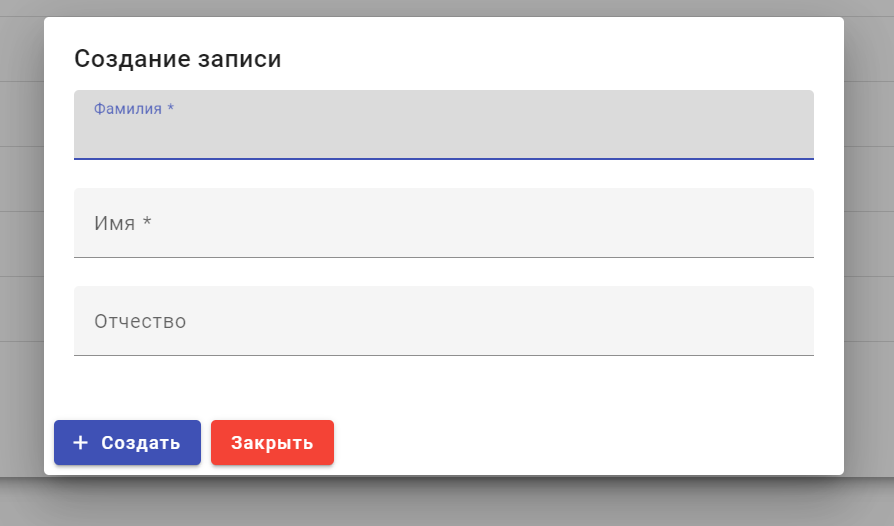


Рисунок 19 – Диалог редактирования записи автора

Для сортировки записей книг используется алгоритм сортировки вставками. Алгоритм представлен на рисунке 20 в виде блок-схемы.

*нет*

*Начало*

*Ввод массива a, размер n*

*Начало цикла i с 1 по n*

*key = a[i]*

*j = i-1*

*j >= 0 и a[j]>key*

*a[j+1] = a[j]*

*--j*

*a[j+1] = key*

*Конец*

*Конец цикла*

*да*

Рисунок 20 – Алгоритм сортировки вставками

## 3.2 Выбор компонентов программного обеспечения

В качестве основной среды разработки (IDE) используется Visual Studio Code, с установленными расширениями для написания приложения на языках программирования C# и TypeScript.

Для работы серверной части требуется среда исполнения .NET версии 7 и выше. Далее перечислены сторонние пакеты серверной части:

* фреймворк ASP.NET Core;
* библиотека для работы с СУБД Entity Framework;
* библиотека для работы с протоколом GRPC «Grpc.AspNetCore».

Для работы клиентской части нужен веб-сервер Nginx или Apache. В основном клиентская часть тестировалась на Nginx. Также как и с серверной частью использовались следующие сторонние пакеты:

* фреймворк Angular версии 16;
* библиотека UI компонентов Angular Material;
* библиотека для работы с файлами «file-saver».

Для разработки UML диаграмм использовалось программное обеспечение Visual Paradigm Community Edition версии 17.

### 3.2.1 Операционная система

-

### 3.2.2 Инструментальное средство разработки и язык программирования

-

### 3.2.3 Вспомогательное программное обеспечение

-

## 3.3 Разработка прикладного программного обеспечения

-

## 3.3.1 Структура прикладного программного обеспечения

-

## 3.3 Инсталляция и настройка

Диаграмма развёртывания программного обеспечения с помощью Docker представлена в виде UML диаграммы на рисунке 21.

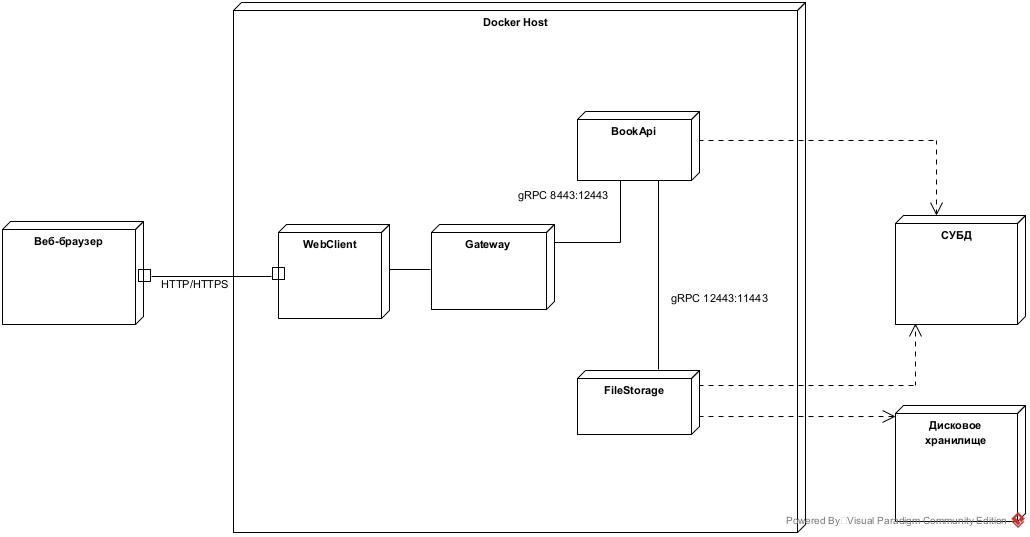


Рисунок 21 – Диаграмма развертывания с использованием контейнеров Docker

Для работы служб потребуется мультиплатформенная среда выполнения .NET Core. Образ Docker со средой выполнения весит примерно от 100 МБ и выше в зависимости от выбранного дистрибутива и размера исполняемого файла.

В основном в сервисах используется СУБД SQLite, но если нужно использовать другую СУБД, то нужно поменять код в Program.cs для BookApi и FileStorage и установить нужные зависимости. К примеру, с использования SQLite:

builder.Services.AddDbContext<AppDbContext>(options => options.UseSqlite("Data Source=book\_api.db"));

Перейти на СУБД PostgreSQL:

builder.Services.AddDbContext<AppDbContext>(options => options.UseNpgsql("Host=remote\_host;Database=book\_api;Username=root;Password=root"));

И далее нужно установить зависимость через утилиту *dotnet*:

dotnet add package Npgsql.EntityFrameworkCore.PostgreSQL

Также потребуется дисковое хранилище для хранения файлов книг в зависимости от их количества, которое пользователю нужно сохранить.

# 4 ТЕСТИРОВАНИЕ СИСТЕМЫ

Написать что такое тесты, их виды, что нужно тестировать и т.д.

## 4.1 Условия и порядок тестирования

-

## 4.2 Исходные данные для контрольных примеров

-

## 4.3 Результаты тестирования

-

# 5 ЭКОНОМИЧЕСКАЯ ЧАСТЬ

Написание программного обеспечения и документации заняло неделю, по 6 часов на день. Возьмём среднюю почасовую заработную плату с сайта «Upwork» для позиции Full Stack разработчика, которая равна 25 долларов США. Посчитаем все дни в недели на шесть, затем умножив на двадцать пять, получим 1050 долларов США.

Также нужно посчитать стоимость поддержки работы и также расходы на электроэнергию, которая используется разработанной системой. Стоимость поддержки за час работы может составить от 15 до 80 долларов США. Так как программа писалась для нужд самого автора дипломной работы, то не будем брать это в учёт.

Серверная часть будет работать на облачном выделенном сервере, предоставляемый облачным провайдером Timeweb. Клиентская часть будет работать на ПК автора с ОС Windows, на планшете под управлением ОС Android и на ноутбуке под управлением дистрибутива ОС Linux. Стоимость аренды выделенного сервера составляет 250 руб. в месяц.

# ЗАКЛЮЧЕНИЕ

В данной лабораторной работе было разработано программное обеспечение с микро-сервисной архитектурой, которое использует протоколы обмена данными gRPC и HTTP REST между службами внутри системы. Также было изучено, как работать с этими протоколами, и их работу в современном фреймворке для серверных и клиентских программ ASP.NET Core.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. ГОСТ 19.701-90. Схемы алгоритмов, программ, данных и систем. – введ. 01.01.92. – М. : Изд-во стандартов, 2010. – 24 с.
2. Ajay D. Kshemklayni. Distributed Computing: Principles, Algorithms, and Systems / Ajay D. Kshemklayni, Mukesh Singhal. – Cambridge University Press, 2008.
3. Artificial Intelligence: A Modern Approach, 4th US ed. / Stuart J. Russell., Peter Norvig. – Prentice Hall, 2020
4. Distributed Systems / Marteen Van Steen, Tannenbaum A.S.
5. Документация Angular: [Электронный ресурс] / Google LLC, 2023. – Режим доступа: <https://angular.io/docs>, свободный.
6. Документация Angular Material: [Электронный ресурс] / Google LLC, 2023. – Режим доступа: <https://material.angular.io/components/categories>, свободный.
7. ASP.NET Documentation: [Электронный ресурс] / Microsoft, 2023. – Режим доступа: <https://learn.microsoft.com/en-us/aspnet/core/?view=aspnetcore-7.0>, свободный.
8. gRPC Documentation: [Электронный ресурс] / gRPC Authors, 2023. - Режим доступа: <https://grpc.io/docs/>, свободный.
9. Language Guide (proto 3) | Protocol Buffers Documentation: [Электронный ресурс] / 2023 Google LLC. – Режим доступа: <https://protobuf.dev/programming-guides/proto3/>, свободный.

*нет*

# ПРИЛОЖЕНИЕ А

Код «Program.cs»

var builder = WebApplication.CreateBuilder(args);

var fileStorageUri = builder.Configuration.GetSection("FileStorageUri").Value ?? throw new ArgumentException("The FileStorageUrl argument is empty");

builder.Services.AddDbContext<AppDbContext>(options => options.UseSqlite("Data Source=book\_api.db"));

builder.Services.AddGrpc();

builder.Services.AddGrpcClient<Fss.FileStorageService.FileStorageServiceClient>(o =>

{

o.Address = new Uri(fileStorageUri);

});

var app = builder.Build();

var scope = app.Services.CreateScope();

var dbContext = scope.ServiceProvider.GetRequiredService<AppDbContext>();

if (dbContext.Database.EnsureCreated())

{

dbContext.Database.Migrate();

}

app.MapGrpcService<BookService>();

app.MapGrpcService<AuthorService>();

app.Run();

Файл «Utils.cs»:

public static class Utils

{

public static async Task<TEntity> GetItemByStringGuid<TEntity>(DbContext context, string id) where TEntity : class

{

if (!Guid.TryParse(id, out Guid guid))

throw new RpcException(new Status(StatusCode.InvalidArgument, $"The provided id = {id} is not valid Guid value."));

var found = await context.Set<TEntity>().FindAsync(guid)

?? throw new RpcException(new Status(StatusCode.NotFound, $"The {nameof(TEntity)} with id = {id} not found."));

return found;

}

}

Файл «AppDbContext.cs»:

namespace BookApi;

public class AppDbContext : DbContext

{

public AppDbContext(DbContextOptions<AppDbContext> options) : base(options) { }

public DbSet<Domains.Author> Authors => Set<Domains.Author>();

public DbSet<Domains.Book> Books => Set<Domains.Book>();

}

Файл «Config/BookApiOptions.cs»

namespace BookApi.Config;

public sealed class BookApiOptions

{

public const string BookApi = "BookApi";

public string FileStorageUrl { get; set; } = string.Empty;

}

Файл «Domain/Author.cs»

namespace BookApi.Domains;

public sealed class Author

{

public Guid Id { get; set; }

public required string FirstName { get; set; }

public required string LastName { get; set; }

public string? MiddleName { get; set; }

public List<Book> Books { get; set; } = new();

public AuthorMessage ToAuthorMessage()

{

return new AuthorMessage()

{

Id = Id.ToString(),

FirstName = FirstName,

LastName = LastName,

MidName = MiddleName,

};

}

}

Файл «Domains/Book.cs»

namespace BookApi.Domains;

public sealed class Book

{

public Guid Id { get; set; }

public required string Title { get; set; }

public string? ISBN { get; set; }

public Guid FileStorageId { get; set; } = Guid.Empty;

public List<Author> Authors { get; set; } = new();

public BookReply ToBookReply()

{

var reply = new BookReply()

{

Id = Id.ToString(),

Title = Title,

Isbn = ISBN,

HasFile = FileStorageId != Guid.Empty

};

reply.Authors.AddRange(Authors.Select(a => a.ToAuthorMessage()));

return reply;

}

}

Файл «Services/AuthorService.cs»

public sealed class AuthorService : Author.AuthorService.AuthorServiceBase

{

private readonly AppDbContext \_dbContext;

public AuthorService(AppDbContext context)

{

\_dbContext = context;

}

public override async Task GetAuthors(Empty request, IServerStreamWriter<AuthorMessage> responseStream, ServerCallContext context)

{

await foreach (var a in \_dbContext.Authors.AsAsyncEnumerable())

{

await responseStream.WriteAsync(new AuthorMessage()

{

Id = a.Id.ToString(),

FirstName = a.FirstName,

LastName = a.LastName,

MidName = a.MiddleName

});

}

}

public override async Task<AuthorMessage> AddAuthor(AuthorAddRequest request, ServerCallContext context)

{

var newAuthor = new Domains.Author()

{

FirstName = request.FirstName,

LastName = request.LastName,

MiddleName = request.MidName,

};

await \_dbContext.Authors.AddAsync(newAuthor);

await \_dbContext.SaveChangesAsync();

return newAuthor.ToAuthorMessage();

}

public override async Task<AuthorMessage> UpdateAuthor(AuthorMessage request, ServerCallContext context)

{

var found = await Utils.GetItemByStringGuid<Domains.Author>(\_dbContext, request.Id);

found.FirstName = request.FirstName;

found.LastName = request.LastName;

found.MiddleName = request.MidName;

await \_dbContext.SaveChangesAsync();

return request;

}

public override async Task<Empty> RemoveAuthor(RemoveRequest request, ServerCallContext context)

{

var found = await Utils.GetItemByStringGuid<Domains.Author>(\_dbContext, request.Id);

\_dbContext.Remove(found);

await \_dbContext.SaveChangesAsync();

return new Empty();

}

}

Файл «Services/BookService.cs»

namespace BookApi.Services;

public class BookService : Books.BookService.BookServiceBase

{

private readonly AppDbContext \_dbContext;

private readonly FileStorageService.FileStorageServiceClient \_fssClient;

public BookService(AppDbContext context, FileStorageService.FileStorageServiceClient fssClient)

{

\_dbContext = context;

\_fssClient = fssClient;

}

public override async Task GetBooks(Empty request, IServerStreamWriter<BookReply> responseStream, ServerCallContext context)

{

await foreach (var book in \_dbContext.Books.Include(b => b.Authors).AsAsyncEnumerable())

await responseStream.WriteAsync(book.ToBookReply());

}

public override async Task<BookReply> AddBook(BookAddRequest request, ServerCallContext context)

{

var newBook = new Book()

{

Title = request.Title,

ISBN = request.Isbn,

};

var authorIds = request.AuthorIds.Distinct().Select(id => id.ToUpper());

var authors = \_dbContext.Authors.Where((a) => authorIds.Contains(a.Id.ToString()));

newBook.Authors = await authors.ToListAsync();

dbContext.Books.Add(newBook);

await \_dbContext.SaveChangesAsync();

var response = new BookReply()

{

Id = newBook.Id.ToString(),

Title = newBook.Title,

Isbn = newBook.ISBN,

HasFile = false,

};

response.Authors.AddRange(authors.Select(a => a.ToAuthorMessage()));

return response;

}

public override async Task<BookReply> UpdateBook(BookUpdateRequest request, ServerCallContext context)

{

var found = await Utils.GetItemByStringGuid<Book>(\_dbContext, request.Id);

await dbContext.Entry(found).Collection(b => b.Authors).LoadAsync();

var authorIds = request.AuthorIds.Distinct().Select(id => id.ToUpper());

found.Title = request.Title;

found.ISBN = request.Isbn;

found.Authors.Clear();

var authors = \_dbContext.Authors.Where((a) => authorIds.Contains(a.Id.ToString()));

found.Authors = await authors.ToListAsync();

await \_dbContext.SaveChangesAsync();

return found.ToBookReply();

}

public override async Task<Empty> RemoveBook(RemoveRequest request, ServerCallContext context)

{

var found = await Utils.GetItemByStringGuid<Book>(\_dbContext, request.Id);

if (found.FileStorageId != Guid.Empty)

{

await \_fssClient.DeleteEntryAsync(new DeleteEntryRequest()

{

Uuid = found.FileStorageId.ToString(),

});

}

\_dbContext.Books.Remove(found);

await \_dbContext.SaveChangesAsync();

return new Empty();

}

public override async Task<HasFileReply> HasFile(HasFileRequest request, ServerCallContext context)

{

var found = await Utils.GetItemByStringGuid<Book>(\_dbContext, request.Id);

return new HasFileReply()

{

File = found.FileStorageId == Guid.Empty

};

}

public override async Task<Empty> UploadBookFile(IAsyncStreamReader<UploadBookFileChunkRequest> requestStream, ServerCallContext context)

{

await requestStream.MoveNext();

var it = requestStream.Current;

var found = await Utils.GetItemByStringGuid<Book>(\_dbContext, it.Id);

using var call = \_fssClient.Upload();

do

{

it = requestStream.Current;

await call.RequestStream.WriteAsync(new FileChunkMessage()

{

ChunkSize = it.ChunkSize,

FileSize = it.FileSize,

FileName = it.Filename,

Chunk = it.Chunk,

});

} while (await requestStream.MoveNext());

await call.RequestStream.CompleteAsync();

var reply = await call.ResponseAsync;

found.FileStorageId = Guid.Parse(reply.Uuid);

await \_dbContext.SaveChangesAsync();

return new Empty();

}

public override async Task DownloadBookFile(DownloadBookFileRequest request, IServerStreamWriter<FileChunkMessage> responseStream, ServerCallContext context)

{

var found = await Utils.GetItemByStringGuid<Book>(\_dbContext, request.Id);

using var call = \_fssClient.Download(new DownloadRequest()

{

Uuid = found.FileStorageId.ToString(),

});

await foreach (var message in call.ResponseStream.ReadAllAsync())

await responseStream.WriteAsync(message);

}

}