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

Факультет компьютерных систем и сетей
Кафедра информатики
Дисциплина: Объектно-ориентированное программирование

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
к курсовому проекту
на тему

СЕРВЕРНАЯ ЧАСТЬ СИСТЕМЫ МЕНЕДЖМЕНТА БИБЛИОТЕКИ

БГУИР КП 1-40 04 01

Студент: Косяков М.М.
гр. 253505

Руководитель: Тушинская Е.В.

Минск 2024

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Система менеджмента библиотеки – это программное обеспечение, предназначенное для автоматизации основных библиотечных процессов. Она позволяет организовать и упростить хранение, поиск, выдачу и возврат книг, а также управление информацией о читателях библиотеки.

Целью данного курсового проекта является создание и разработка серверной части системы менеджмента библиотеки, которая позволит автоматизировать основные процессы библиотеки. Это включает в себя упрощение и оптимизацию процессов хранения, поиска, выдачи и возврата книг. Автоматизация этих процессов поможет библиотеке более эффективно управлять своими ресурсами и повысить качество обслуживания для читателей. Серверная часть системы менеджмента библиотеки обеспечит более удобный и быстрый доступ к ресурсам библиотеки. Это включает в себя разработку функционала для онлайн-каталога книг, который позволит работнику библиотеки просматривать доступные книги, проверять их наличие. Также она упростит взаимодействие между библиотекарями и читателями. Это включает в себя разработку функционала, позволяющего библиотекарям легко регистрировать новых читателей, вести учет выданных книг, а также управлять сроками их возврата.

Задачи данного курсового проекта:

– провести анализ требований к системе менеджмента библиотеки, определить функциональные и нефункциональные требования;

– разработать архитектуру системы, определить ее основных компоненты и интерфейс;

– разработать и реализовать программное обеспечение;

– сформулировать выводы, исходя из нашей работы.

В данном курсовом проекте будут рассмотрены такие главы, как анализ предметной области, проектирование программного средства, разработка программного средства, проверка работоспособности приложения и руководство пользователя.

1 АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

1.1 Обзор аналогов

Koha является одной из наиболее популярных и широко используемых в мире open-source систем управления библиотеками, предлагающей широкий спектр функций, включая каталогизацию, управление циркуляцией, комплектование, отчетность и онлайн-каталог, поддерживающей множество языков и валют, отличающейся высокой гибкостью и возможностью кастомизации под конкретные требования библиотеки. Разработку Koha начала в 1999 году Katipo Communications для библиотечного союза Хорофенуа, Новая Зеландия. Первая инсталляция состоялась в январе 2000 года. Название «koha» означает подарок, дар на языке маори (Новая Зеландия).

Koha версии 3.0.0 была выпущена 11 августа 2008 года. Новые возможности включают новый дизайн пользовательского интерфейса, более совершенные поисковые функции, лучшую поддержку многих подразделений, метки читателей и много общих улучшений. Koha преимущественно используется под Linux. Теоретически она может работать под Windows. Однако под Windows это требует сложной установки нескольких модулей, которые нужно загружать из Интернета и версии которых могли измениться с момента опубликования документации по Koha. Скриншот программного обеспечения Koha указан на рисунке 1.1.

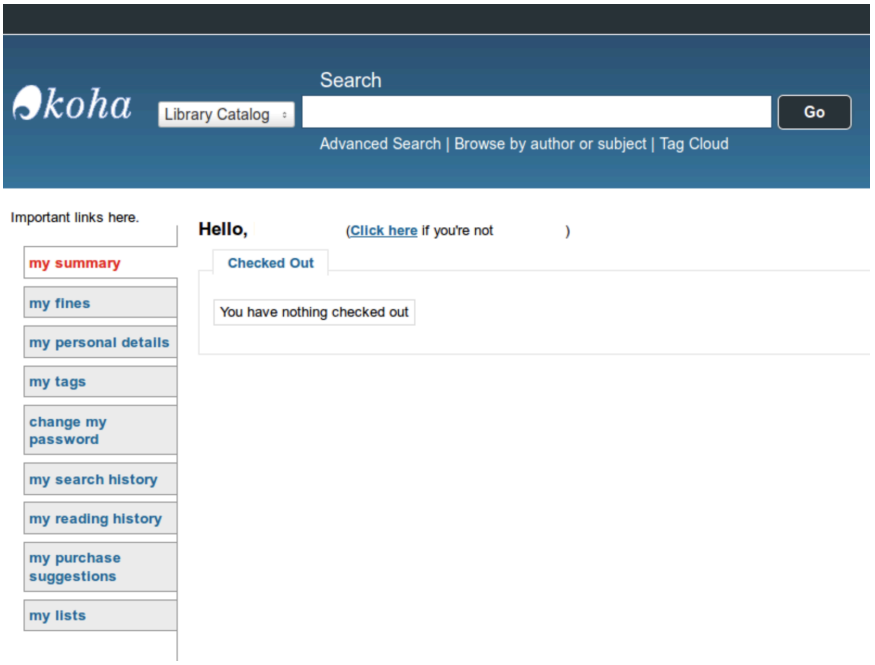

Рисунок 1.1 – Скриншот программного обеспечения «Koha»

OpenBiblio – это интегрированная библиотечная система с открытым исходным кодом. Программное обеспечение пользуется большой популярностью в небольших и сельских библиотеках по всему миру, благодаря своей простоте, обширной языковой поддержке и хорошей документации. Openbiblio был создан в 2002 году Дэйвом Стивенсом, заинтересованным в создании простой в использовании, хорошо документированной и простой в установке библиотечной системы. Текущий сопровождающий – Ханс ван дер Вейдж. После 2017 года на openbiblio.de была опубликована текущая версия с множеством опций и исправлений.

Хотя система все еще находится в стадии активной разработки, она уже широко используется в небольших библиотеках и архивах по всему миру. Исследователи из Национального автономного университета Мексики рекомендовали использовать эту систему в библиотеках коренных народов Мексики, в частности, из-за поддержки языка науатль. Национальная библиотека Армении рекомендовала использовать OpenBiblio для 900 небольших (менее 40 000 томов) и сельских библиотек страны.

Система была переведена на испанский язык профессором кастильского языка и используется в начальной школе Чили. Кроме того, по словам Вернера Вестермана из чилийской группы Educalibre, интерес к этой программе проявили Колумбия, Куба и Венесуэла.

Отдел развития библиотеки штата Вайоминг разработал серию учебных материалов для небольших библиотек, заинтересованных во внедрении OpenBiblio. Исследователи из отдела информационных наук Федерального университета Параибы также обсуждают использование OpenBiblio в обучении будущих библиотекарей системам автоматизации библиотек. Скриншот программного обеспечения OpenBiblio указан на рисунке 1.2.

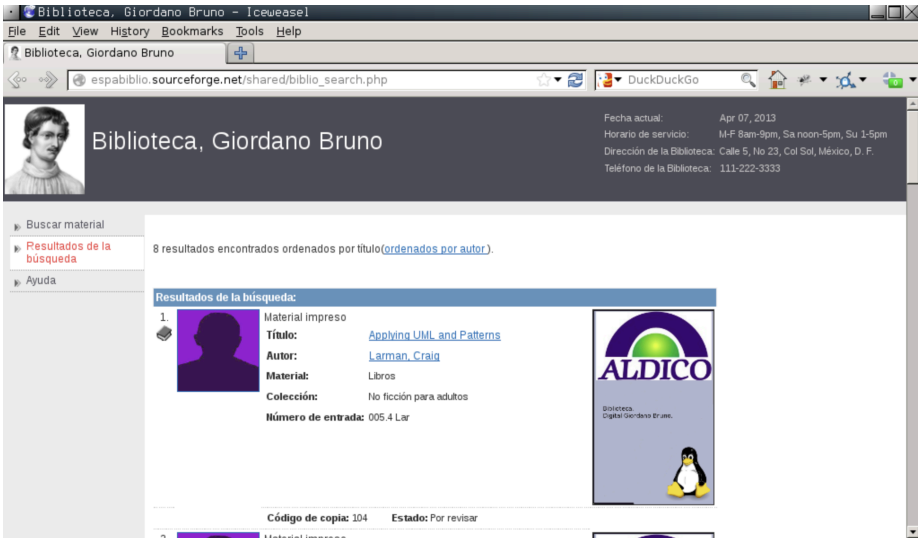

Рисунок 1.2 – Скриншот программного обеспечения «OpenBiblio»

Tellico – это бесплатное программное обеспечение с открытым исходным кодом, предназначенное для каталогизации и управления различными типами коллекций, включая книги, компакт-диски, фильмы, видеоигры и многое другое.

Некоторые ключевые особенности Tellico: поддержка большого количества различных типов коллекций и полей метаданных для их описания, возможность импорта данных из различных источников, в том числе онлайн-каталогов и баз данных, функции поиска, сортировки и фильтрации коллекций, генерация отчетов и статистики по коллекциям.

Tellico особенно полезен для личных коллекций, но также может быть применим и в небольших библиотечных учреждениях для каталогизации и управления книжными фондами. Его открытый исходный код и расширяемость позволяют адаптировать программу под конкретные потребности библиотеки. Скриншот программного обеспечения Tellico указан на рисунке 1.3.

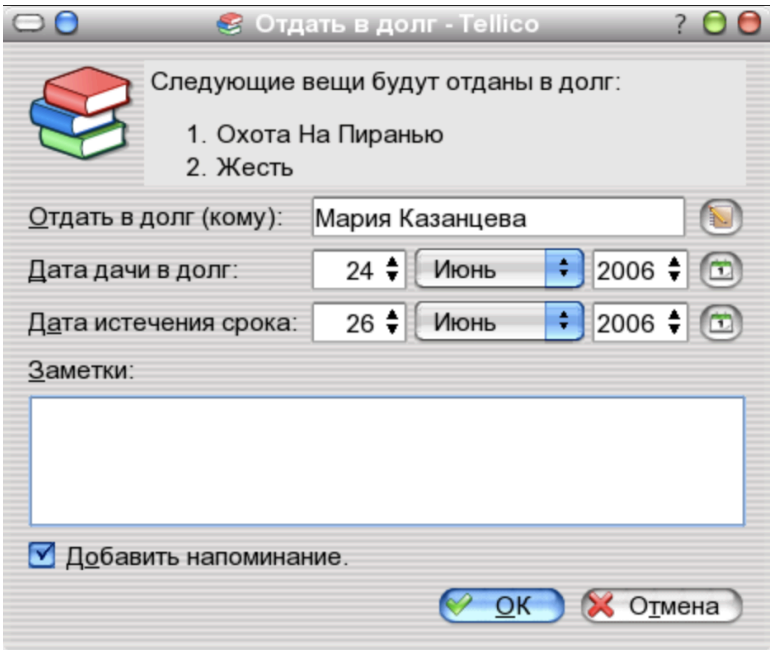

Рисунок 1.3 – Скриншот программного обеспечения «Tellico»

Библиотека БГУИР – всеобъемлющий онлайн-инструмент для поиска и доступа к информационным ресурсам вузовской библиотеки. Он предлагает широкий спектр возможностей, включая простой и расширенный поиск по различным параметрам, доступ к базам данных, содержащим сведения о книгах, статьях, стандартах, трудах преподавателей, а также к полнотекстовым версиям магистерских диссертаций. Зарегистрированные пользователи могут просматривать информацию о выданных и заказанных ими книгах. Каталог также предоставляет сведения о местах книговыдачи и расположении различных подразделений библиотеки. В целом, данный

ресурс является всеобъемлющим и незаменимым инструментом для эффективного поиска и управления информационными ресурсами в библиотеке БГУИР. Скриншот программного обеспечения Библиотека БГУИР указан на рисунке 1.4.

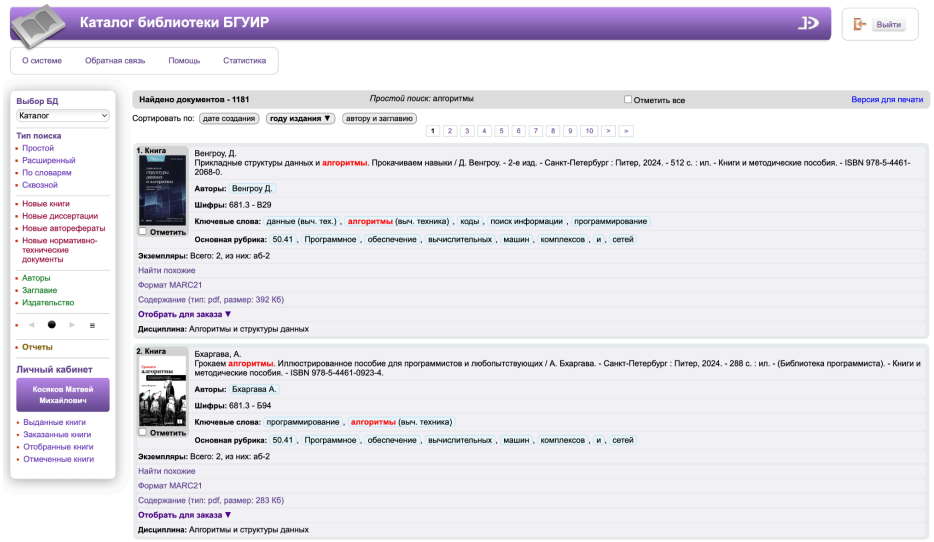

Рисунок 1.4 – Скриншот программного обеспечения «Библиотека БГУИР»

1.2 Постановка задачи

В рамках данного курсового проекта была поставлена задача: разработать серверную часть системы менеджмента библиотеки.

Проанализировав конкретно поставленную задачу, был выделен следующий ряд подзадач:

– описать модели данных проекта;

– реализовать механизмы сохранения, чтения, обновления и удаления данных в инструментах хранения данных;

– разработать функционал онлайн-каталога книг;

– разработать функционал управления читателями;

– разработать функционал учета и контроля за книгами;

– разработать приложение в формате веб-приложения, доступное через веб-браузер;

– настроить логирование;

– провести тестирование системы с использованием различных сценариев и нагрузок, чтобы проверить ее функциональность, производительность и надежность;

– разработать интерактивную документацию, которая будет содержать подробное описание функциональности системы, API-интерфейсов, примеры использования.

Разработав данный набор задач можно перейти непосредственно к проектированию программного средства.

2 ПРОЕКТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА

2.1 Общая информация

Для реализации системы будет использован язык программирования Golang (Go). Golang предоставляет высокую производительность и эффективность разработки, что особенно важно для системы, которая должна обрабатывать большое количество запросов на поиск и выдачу книг. Проект будет разрабатываться на операционной системе macOS, которая является одной из популярных операционных систем, разработанных и выпускаемых компанией Apple.

В качестве базы данных будет использована MongoDB, которая является документоориентированной NoSQL базой данных. MongoDB обеспечивает гибкую модель данных и хорошую масштабируемость, что позволит эффективно хранить информацию о книгах, читателях и других сущностях библиотеки.

Для удобства разработки будет использована система контроля версий Git и платформа GitHub для хостинга репозитория. Это позволит легко отслеживать изменения в коде и управлять версиями. Редактор кода Visual Studio Code (Vs Code) будет использоваться для разработки. Vs Code обладает множеством полезных функций, таких как автодополнение кода, отладка и интеграция с Git, что обеспечит удобную среду разработки и повысит производительность.

Для обработки HTTP-запросов и маршрутизации будет использован пакет julienschmidt/httprouter. Этот пакет предоставляет эффективный маршрутизатор HTTP-запросов, что поможет обрабатывать запросы и управлять маршрутами веб-приложения.

Для реализации интерфейса интерактивной документации системы менеджмента библиотеки будет использована комбинация технологий HTML, CSS и JavaScript. HTML будет использоваться для структурирования контента и определения элементов интерфейса. CSS будет применяться для оформления и стилизации элементов, обеспечивая приятный внешний вид документации. JavaScript будет использоваться для добавления интерактивности на клиентской стороне, обработки событий и взаимодействия с серверной частью.

Для обеспечения удобства развертывания и масштабируемости проекта он будет обернут в контейнеры Docker. Docker позволяет упаковать приложение и его зависимости в изолированный контейнер, который можно запустить на любой платформе, поддерживающей Docker. Для данного проекта будут созданы два контейнера: контейнер для веб-приложения и контейнер для базы данных MongoDB. Веб-приложение будет работать с базой данных, поэтому необходимо обеспечить их совместную работу.

2.2 Разработка функциональности программного средства

Проанализировав требования к проектируемому приложению были определены возможности пользователя, они представлены в виде use-case диаграммы на рисунке 2.1.

Рисунок 2.1 – Use-case диаграмма пользователя

2.3 Архитектура программного средства

На этапе продумывания приложения курсового проекта и проектирования было принято решение о необходимости реализовать следующие основные функции:

– описать модели данных проекта;

– реализовать механизмы сохранения, чтения, обновления и удаления данных в инструментах хранения данных;

– разработать функционал онлайн-каталога книг;

– разработать функционал управления читателями;

– разработать функционал учета и контроля за книгами;

– разработать приложение в формате веб-приложения, доступное через веб-браузер;

– настроить логирование;

– провести тестирование системы с использованием различных сценариев и нагрузок, чтобы проверить ее функциональность, производительность и надежность;

– разработать интерактивную документацию, которая будет содержать подробное описание функциональности системы, API-интерфейсов, примеры использования.

3 РАЗРАБОТКА ПРОГРАММНОГО СРЕДСТВА

3.1 Описание моделей данных

Схема моделей данных указана на рисунке 3.1.

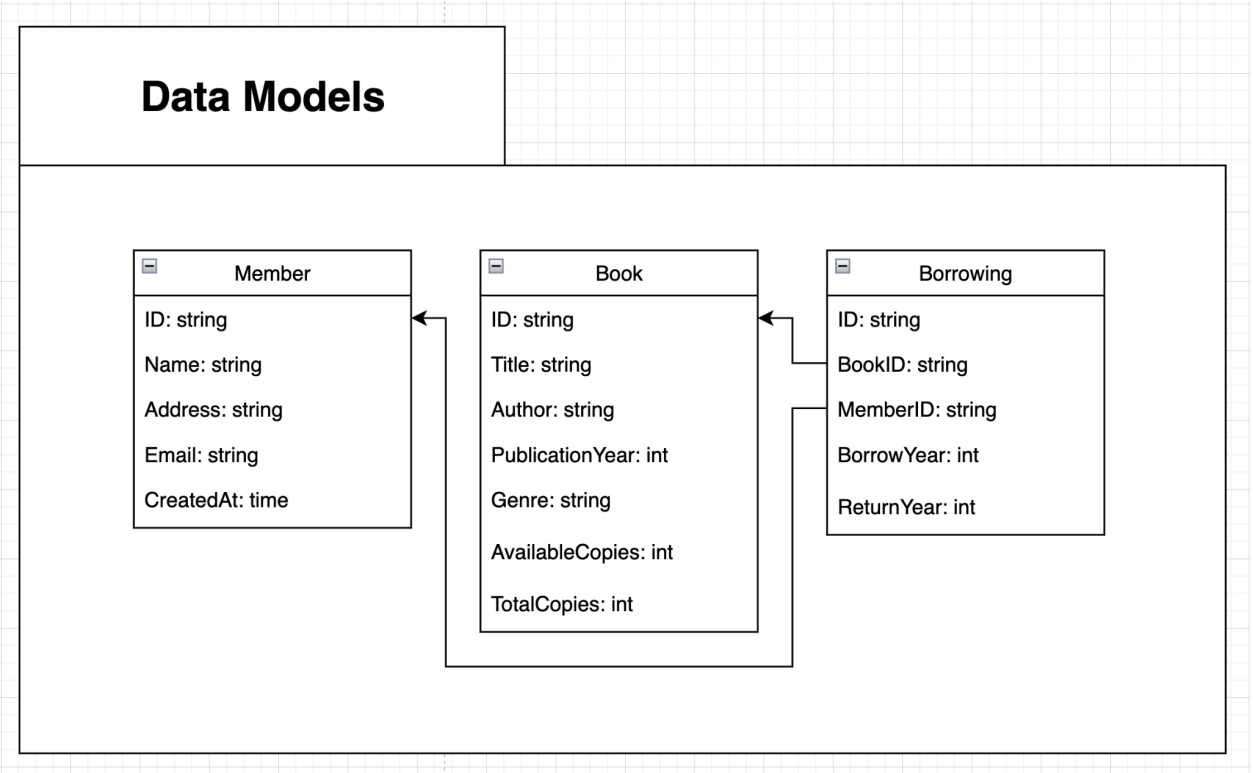

Рисунок 3.1 – Схема моделей данных приложения и их полей

Модель Member представляет информацию о члене библиотеки. У каждого члена библиотеки есть уникальный идентификатор (ID), имя (Name), адрес проживания (Address), электронная почта (Email) и время регистрации в системе (CreatedAt). Идентификатор используется для однозначной идентификации каждого члена библиотеки. Имя, адрес проживания и электронная почта предоставляют личные данные, а время регистрации указывает на дату и время, когда член был зарегистрирован в системе библиотеки.

Модель Book содержит информацию о книге в библиотеке. У каждой книги есть уникальный идентификатор (ID), название (Title), автор (Author), год публикации (PublicationYear), жанр (Genre), количество доступных копий (AvailableCopies) и общее количество копий (TotalCopies). Идентификатор используется для однозначной идентификации каждой книги. Название и автор указывают на основные характеристики книги. Год публикации указывает на год, когда книга была издана, жанр указывает на классификацию книги по жанру, количество доступных копий показывает, сколько копий этой

книги в данный момент доступно для заимствования, а общее количество копий указывает на общее число копий этой книги, которые есть в библиотеке.

Модель Borrowing представляет информацию о заимствовании книги. У каждой записи о заимствовании есть уникальный идентификатор (ID), идентификатор книги (BookID), идентификатор члена библиотеки (MemberID), год заимствования (BorrowYear) и год возврата (ReturnYear). Идентификатор используется для однозначной идентификации каждой записи о заимствовании, идентификатор книги указывает на книгу, которая была заимствована, идентификатор члена библиотеки указывает на члена, который заимствовал книгу, год заимствования указывает на год, когда книга была заимствована, а год возврата указывает на год, когда книга была возвращена в библиотеку.

Все модели используют строковый тип данных для идентификаторов, что позволяет им быть гибкими, использовать ObjectID() для генерации идентификаторов и иметь структуру для хранения их в MongoDB. Модель Member хранит базовую информацию о читателях. Модель Book хранит основную информацию о книгах. Модель Borrowing отслеживает заимствования книг. Все типы связи в моделях имеют отношение One-to-Many.

3.2 Реализация инструментов для хранения данных

Схема инструментов для хранения данных указана на рисунке 3.2.

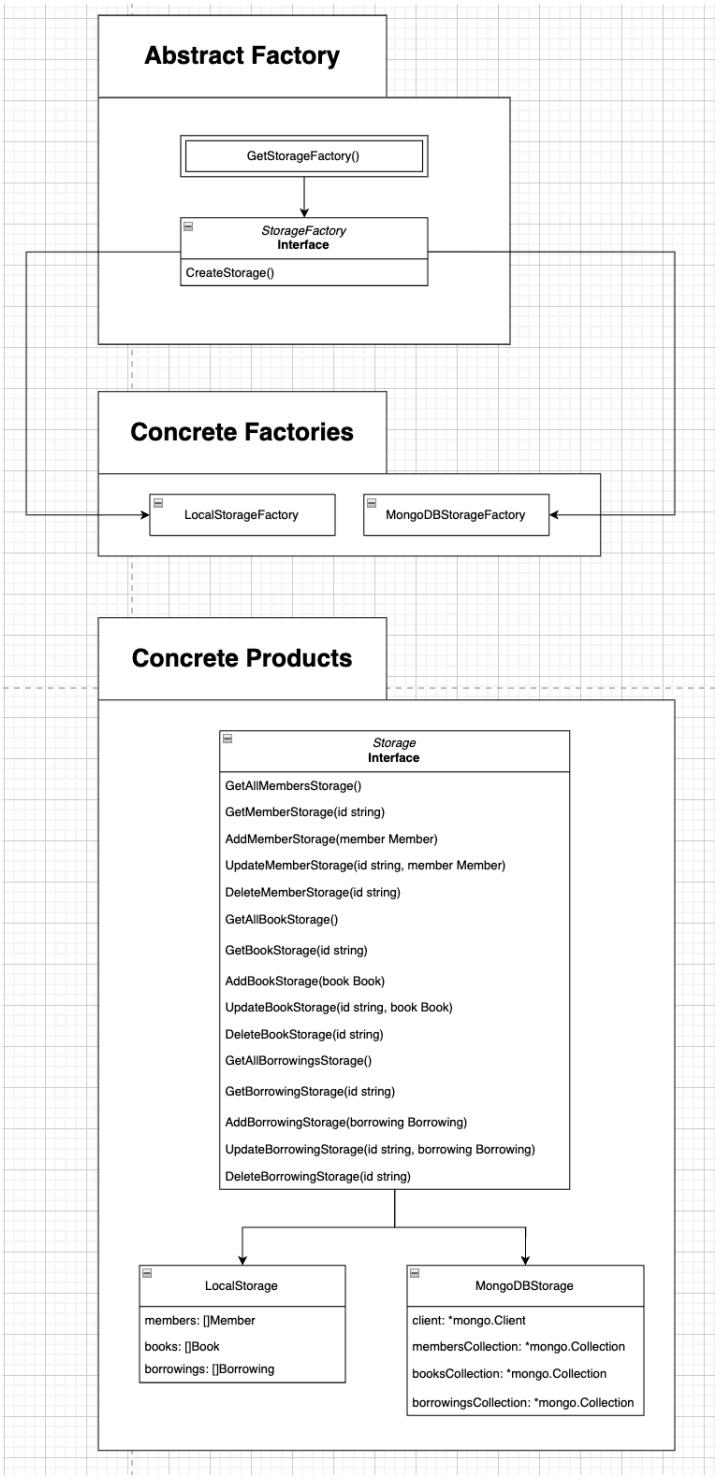

Рисунок 3.2 – Схема инструментов для хранения данных и их полей

Абстрактная фабрика (Abstract Factory) является паттерном проектирования, который позволяет создавать семейства связанных объектов без указания их конкретных классов. Вместо этого используется общий интерфейс, через который клиентский код может взаимодействовать с создаваемыми объектами. В данном случае у нас есть абстрактная фабрика StorageFactory, которая определяет контракт для создания объектов хранения.

Она объявляет методы, которые должны быть реализованы в конкретных фабриках.

В ходе разработки проекта были разработаны две конкретные фабрики. LocalStorageFactory – фабрика, которая реализует интерфейс StorageFactory и отвечает за создание экземпляров LocalStorage. Она выполняет логику создания объектов хранения локально, без использования базы данных. MongoDBStorageFactory – фабрика, которая также реализует интерфейс StorageFactory и отвечает за создание экземпляра MongoDBStorage. Она выполняет более сложную логику, включающую подключение к базе данных MongoDB и создание коллекций для хранения книг, читателей и заимствований.

Интерфейс Storage определяет методы для работы с книгами, читателями и заимствованиями. Он включает методы, такие как получение списка всех читателей, получение информации о конкретном читателе по его идентификатору, добавление нового читателя, обновление информации о читателе и другие. Конкретные реализации этого интерфейса, такие как LocalStorage и MongoDBStorage, предоставляют реализацию этих методов для соответствующих типов хранилищ. Конструктор GetStorageFactory() принимает тип хранилища (например, local или mongodb) и возвращает соответствующую фабрику.

Таким образом, абстрактная фабрика позволяет абстрагироваться от конкретных реализаций объектов хранения и создавать их с помощью общего интерфейса. Это позволяет легко заменять или добавлять новые типы хранилищ без изменения кода, который использует эти объекты.

3.3 Реализация бизнес-логики

Схема классов репозиториев указана на рисунке 3.3.

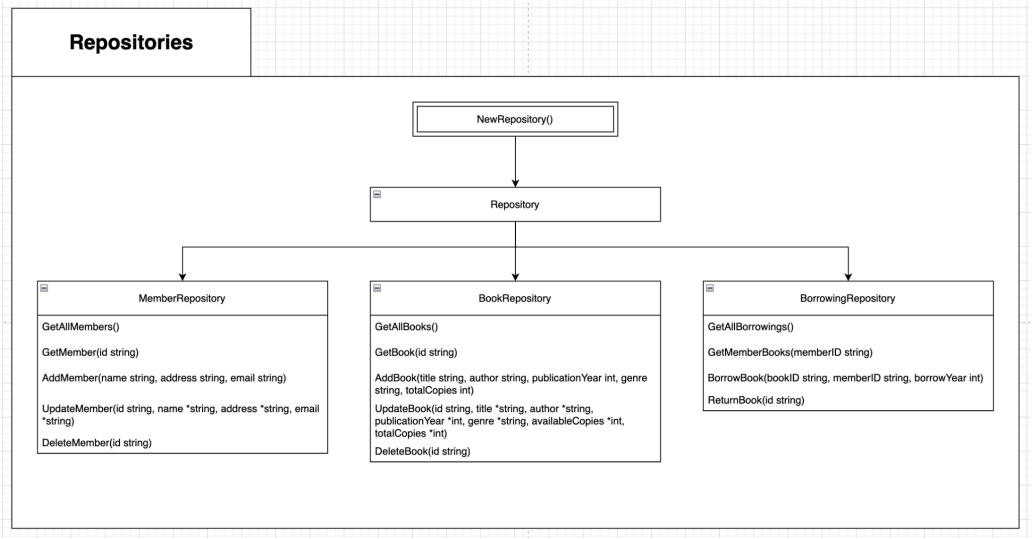

Рисунок 3.3 – Схема классов репозиториев и их полей

Класс Repository и его конкретные репозитории предоставляют удобный интерфейс для работы с данными о читателях, книгах и заимствованиях в библиотеке, а также позволяют выполнять различные операции с этими данными, включая добавление, обновление и удаление. Репозиторий представляет собой слой бизнес-логики, который обеспечивает доступ к данным и взаимодействие с хранилищем. Этот слой содержит класс Repository, а также конкретные репозитории для работы с различными моделями данных.

Конкретные репозитории включают MemberRepository (репозиторий для читателей), BookRepository (репозиторий для книг) и BorrowingRepository (репозиторий для заимствований). Каждый из этих репозиториев реализует бизнес-логику, связанную с соответствующей моделью данных.

У MemberRepository есть методы, которые позволяют получить список всех читателей, получить информацию о конкретном читателе по его идентификатору, добавить нового читателя и обновить или удалить информацию о читателе.

У BookRepository есть методы, которые позволяют получить список всех книг, получить информацию о конкретной книге по ее идентификатору, добавить новую книгу и обновить или удалить информацию о книге.

У BorrowingRepository есть методы, которые позволяют получить список всех заимствований книг, получить список книг, заимствованных конкретным читателем, зарегистрировать заимствование книги и зарегистрировать возврат книги.

Конструктор NewRepository() создает экземпляр класса Repository, который может быть использован для взаимодействия с данными через конкретные репозитории.

Таким образом, класс Repository и его конкретные репозитории предоставляют удобный интерфейс для работы с данными, связанными с читателями, книгами и заимствованиями в системе. Они абстрагируют детали взаимодействия с хранилищем данных и позволяют выполнять операции с данными более высокого уровня.

3.4 Реализация сервиса, предоставляющего работу с API

Схема сервиса указана на рисунке 3.4.

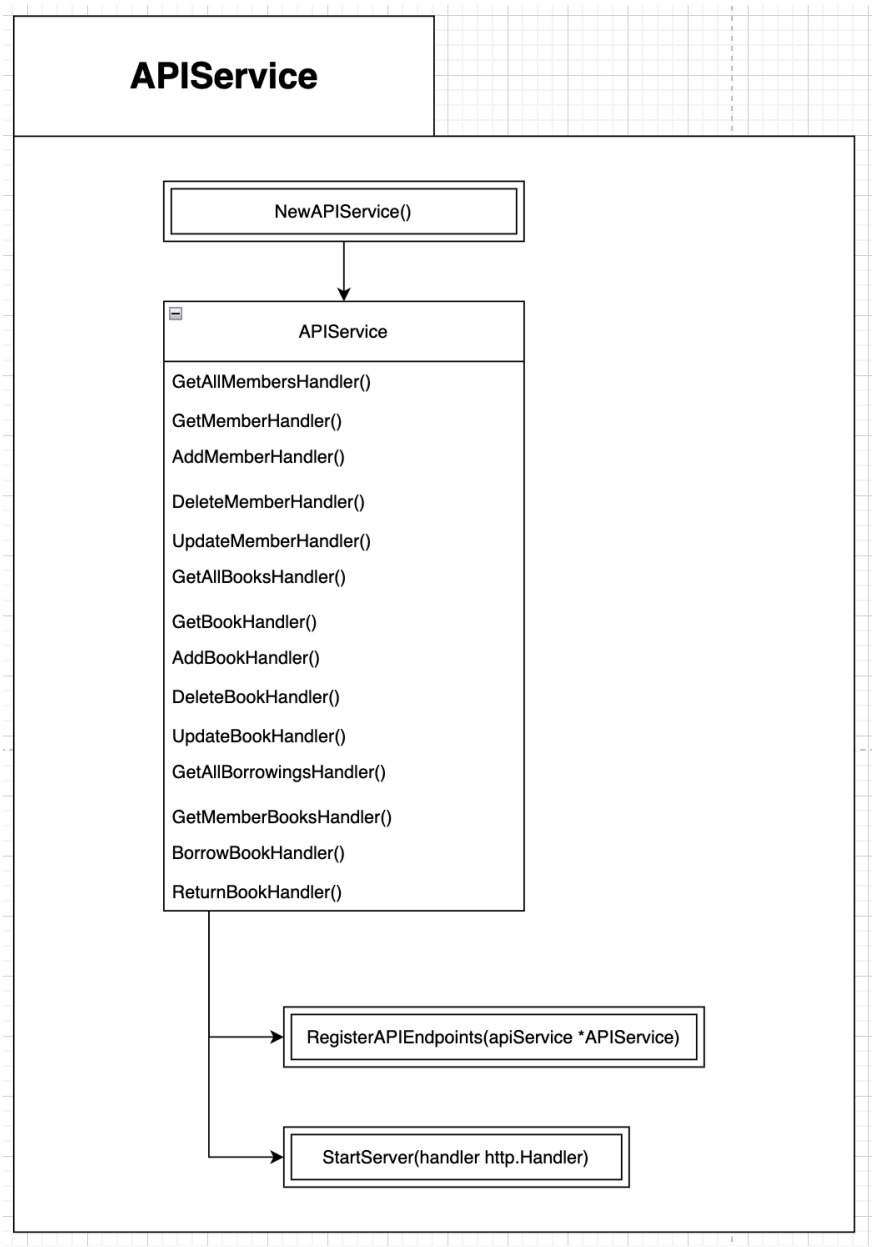

Рисунок 3.4 – Схема сервиса, предоставляющего работу с API

APIService – это сервис, который предоставляет API для взаимодействия с системой. Он использует слой бизнес-логики, в котором осуществляются основные операции с данными, а затем данные сохраняются в хранилище. APIService реализует обработку API-запросов и предоставляет методы для работы с различными сущностями. У сервиса есть несколько методов, которые обрабатывают API-запросы для различных сущностей.

Конструктор NewAPIService() создает новый экземпляр APIService. Реализация конструктора создает синглтон, что означает, что в рамках приложения будет существовать только один экземпляр APIService.
Дополнительная функция RegisterAPIEndpoints(apiService *APIService) регистрирует обработчики API-запросов для сервиса. Это позволяет

настроить маршрутизацию запросов к соответствующим методам обработки в APIService.

Функция StartServer(handler http.Handler) запускает сервер и принимает в качестве параметра обработчик HTTP-запросов. Она отвечает за обработку входящих запросов и их передачу на соответствующие методы обработки в APIService.

3.5 Настройка логирования

Для настройки логирования в проекте используется пакет log из стандартной библиотеки Go. Логирование выполняется в файл, который открывается и закрывается с помощью функций OpenLogFile и CloseLogFile соответственно.

Функция OpenLogFile открывает файл журнала и сохраняет его в переменной logFile. Она принимает путь к файлу журнала из функции loadenv.LoadGlobalEnv(). Файл открывается с флагами os.O_TRUNC (очистить файл, если он существует), os.O_CREATE (создать файл, если он не существует) и os.O_WRONLY (открыть файл только для записи). Режим доступа к файлу устанавливается на 0644. Если возникает ошибка при открытии файла, функция возвращает ошибку типа fmt.Errorf, содержащую информацию об ошибке.

Функция CloseLogFile закрывает файл журнала, если он был открыт. Если возникает ошибка при закрытии файла, вызывается лог-сообщение с помощью log.Fatal.

Функция LogWriter записывает сообщение в файл журнала. Она принимает параметры method (метод HTTP-запроса), path (путь запроса) и statusCode (код состояния HTTP-ответа). Скриншот файла с логами указан на рисунке 3.5.

```
Starting server on port 8080
[2024-05-25 16:48:27] GET / - 200
[2024-05-25 16:48:40] GET / - 200
[2024-05-25 16:48:43] GET /members - 200
[2024-05-25 16:48:48] GET /members - 200
[2024-05-25 16:49:08] POST /members - 201
[2024-05-25 16:49:10] GET /members - 200
[2024-05-25 16:49:14] GET /members/{memberID} - 200
[2024-05-25 16:49:20] GET /books - 200
[2024-05-25 16:49:35] POST /books - 201
[2024-05-25 16:49:39] GET /books - 200
[2024-05-25 16:49:43] GET /books/{bookID} - 200
[2024-05-25 16:49:47] DELETE /books/{bookID} - 200
[2024-05-25 16:49:48] GET /books - 200
```

Рисунок 3.5 – Файл с логами

3.6 Разработка интерактивной документации

Функция SwaggerHandler обрабатывает HTTP-запросы, связанные со страницей документации. Она загружает файл index.html из директории internal/swagger/static, парсит его в качестве HTML-шаблона и затем отображает этот шаблон.

Помимо index.html, в этой директории также находятся файлы style.css и app.js, которые, отвечают за стилизацию и функциональность страницы документации соответственно. Эти файлы загружаются и обрабатываются браузером клиента при отображении страницы.

Таким образом, данная функция обеспечивает отображение страницы интерактивной документации, которая позволяет пользователям ознакомиться с API, предоставляемым приложением. Эта функциональность часто используется для улучшения документации и упрощения взаимодействия с API для разработчиков, работающих с приложением. Скриншоты страниц приложения указаны на рисунках 3.6, 3.7, 3.8.

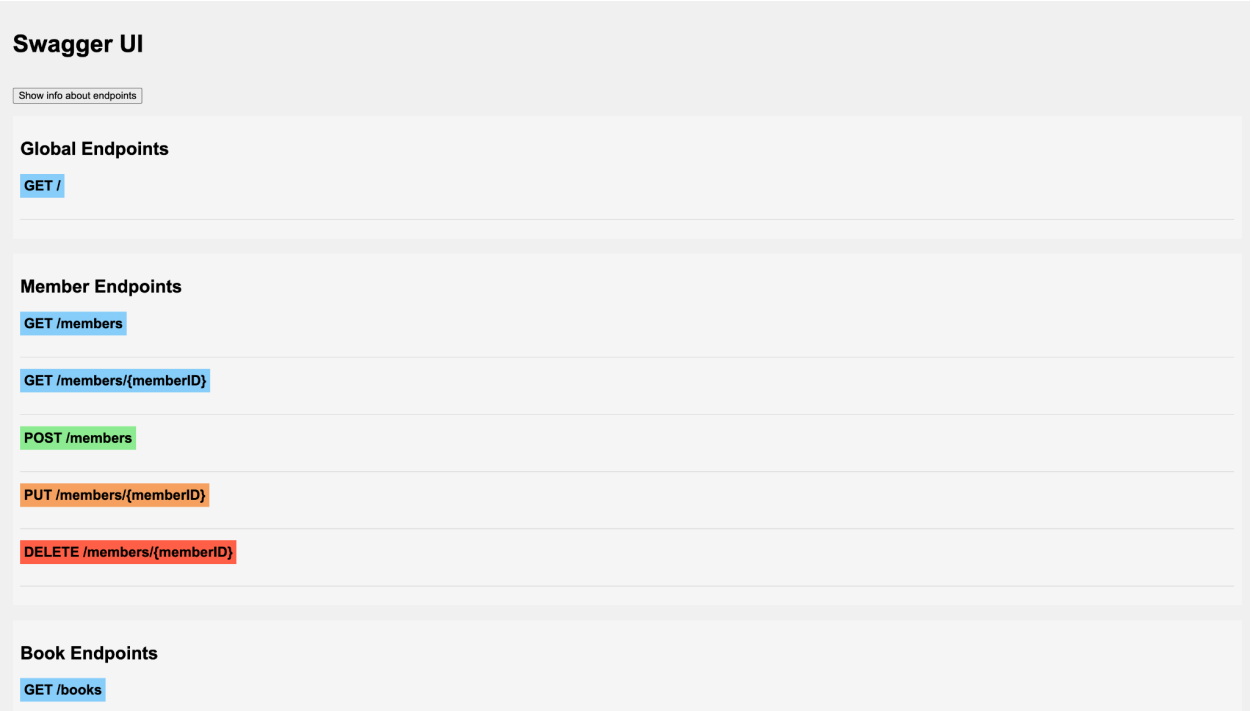

Рисунок 3.6 – Страница интерактивной документации

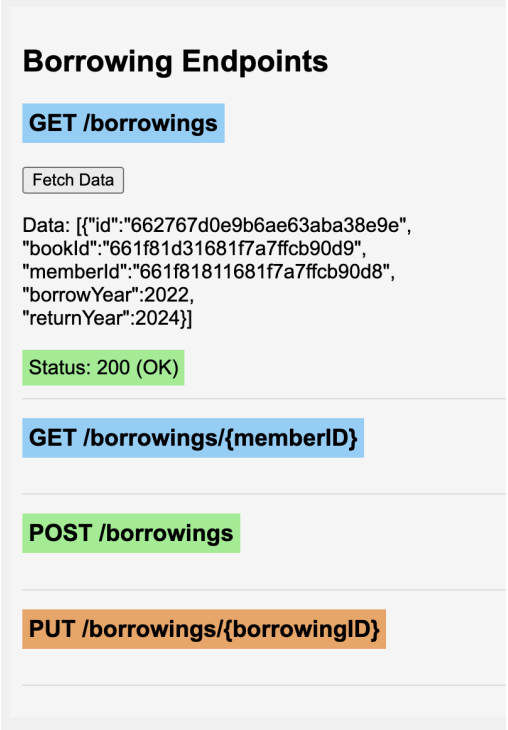

Рисунок 3.7 – Пример запуска обработчика

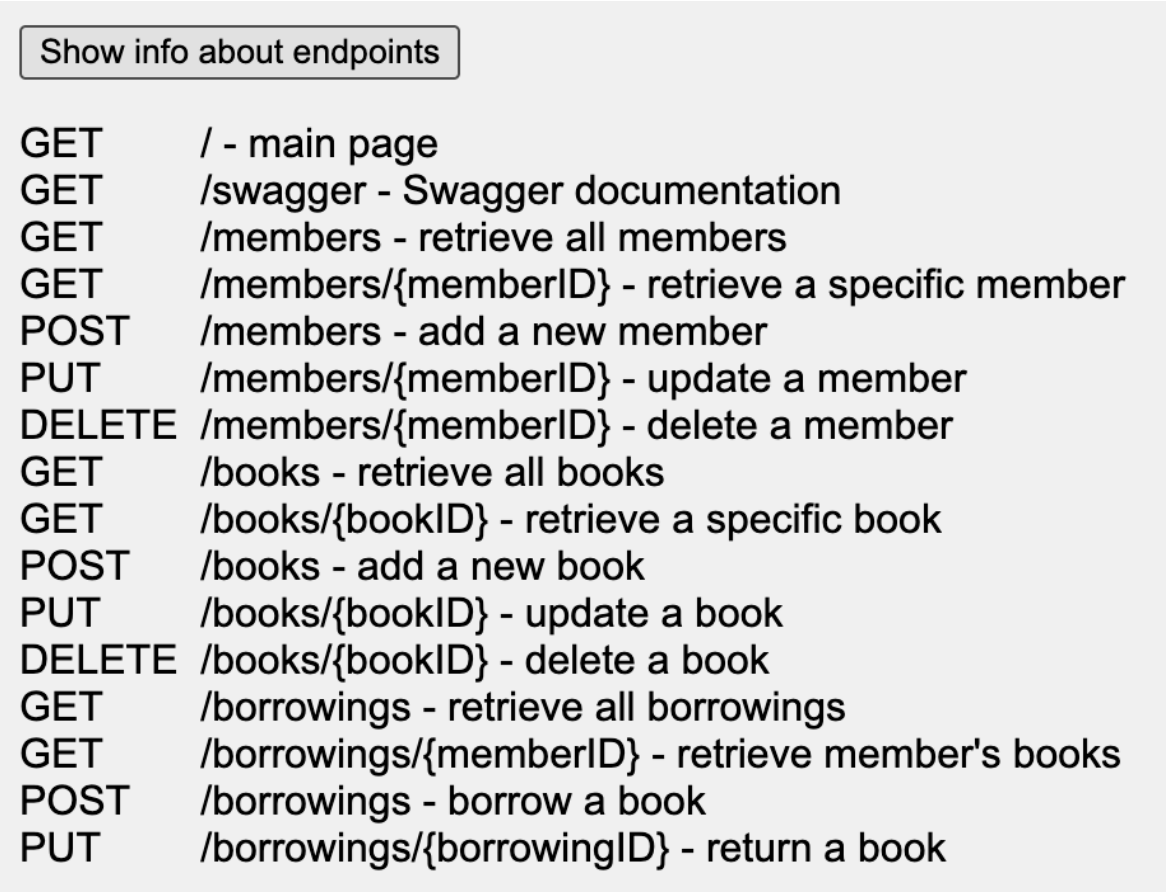

Рисунок 3.8 – Информация об обработчиках API

3.7 Разработка программного кода

Все программное обеспечение было разделено на библиотеку, в которой хранятся модели, репозитории, хранилища, различные утилиты и основное приложение, в которое была подключена библиотека. В основном приложении описаны сервис для работы с API, сериализаторы, тесты и интерактивная документация. Скриншоты файловой архитектуры указаны на рисунках 3.9, 3.10.

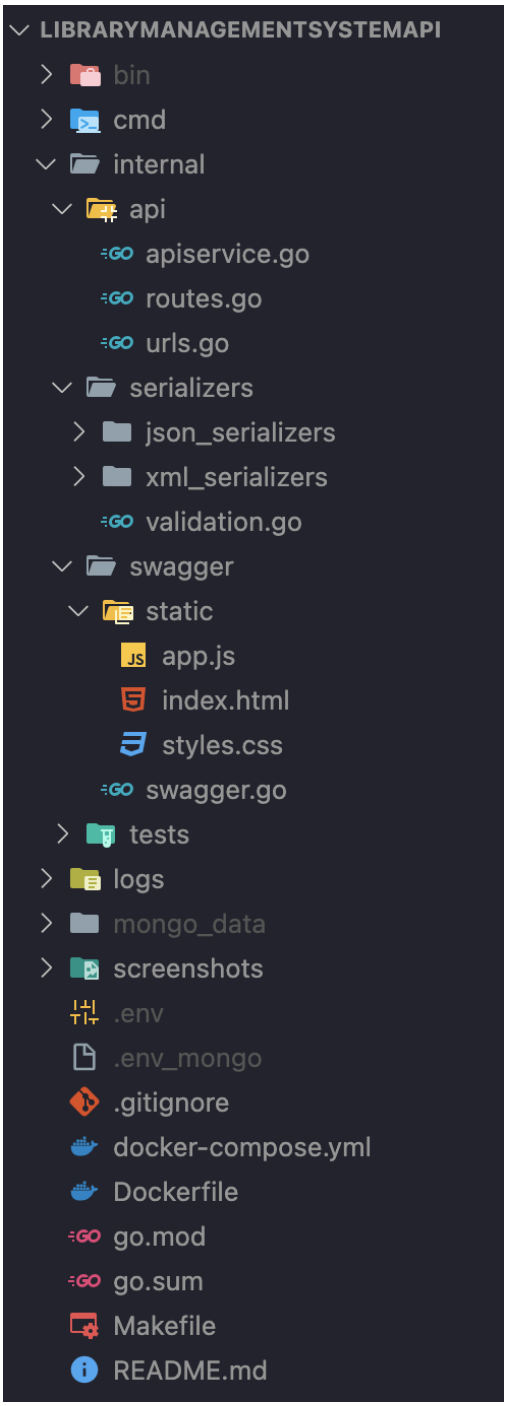

Рисунок 3.9 – Файловая архитектура основного приложения

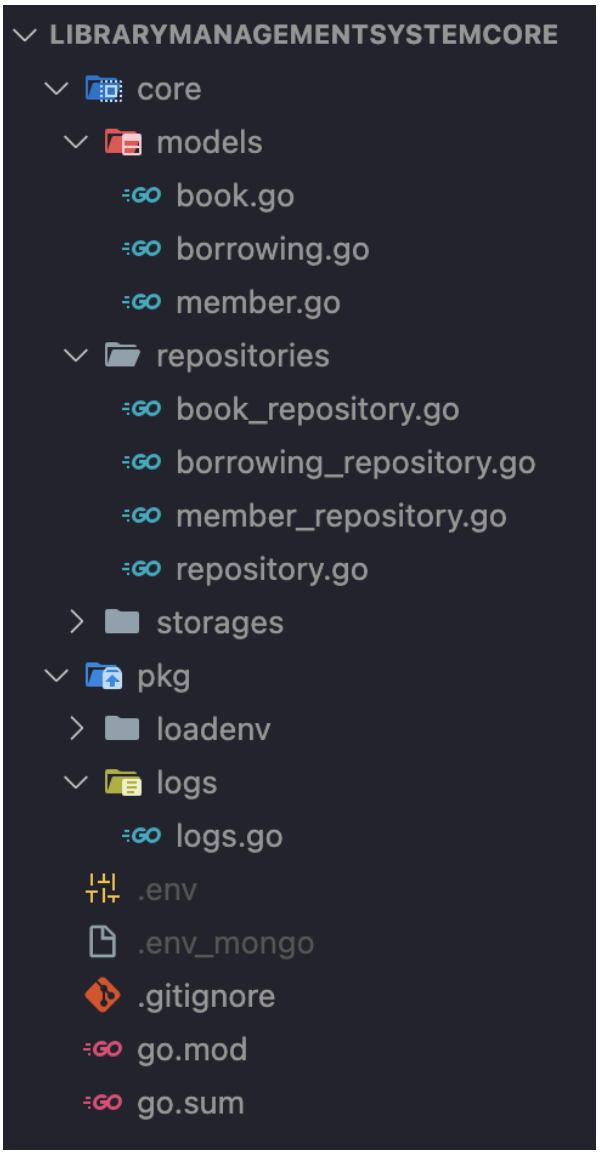

Рисунок 3.10 – Файловая архитектура библиотеки

4 ПРОВЕРКА РАБОТОСПОСОБНОСТИ ПРИЛОЖЕНИЯ

Осуществлялось функциональное тестирование; отчет о проведенном тестировании представлен в таблице 4.1.

Таблица 4.1 – Тестирование программного средства

№	Тестируемая функция	Ожидаемый результат	Полученный результат
1	Главная страница	Отображение главной страницы приложения	Соответствует заданному результату
2	Интерактивная документация	Отображение страницы с интерактивной документацией	Соответствует заданному результату
3	Получить всех участников	Получение списка всех участников	Соответствует заданному результату
4	Получить участника	Получение информации о конкретном участнике	Соответствует заданному результату
5	Добавить участника	Добавление нового участника	Соответствует заданному результату
6	Обновить участника	Обновление информации о конкретном участнике	Соответствует заданному результату
7	Удалить участника	Удаление участника	Соответствует заданному результату
8	Получить все книги	Получение списка всех книг	Соответствует заданному результату
9	Получить книгу	Получение информации о конкретной книге	Соответствует заданному результату
10	Добавить книгу	Добавление новой книги	Соответствует заданному результату
11	Обновить книгу	Обновление информации о конкретной книге	Соответствует заданному результату

Также был написан набор юнит-тестов для различных методов в репозитории программного средства. В каждом тесте проверяется функциональность соответствующего метода путем вызова методов репозитория и сравнения ожидаемых результатов с полученными. Эти тесты помогают проверить, что методы репозитория работают правильно и возвращают ожидаемые результаты для различных операций с данными (добавление, обновление, удаление и т. д.). Скриншоты тестирования приложения указаны на рисунках 4.1, 4.2, 4.3.

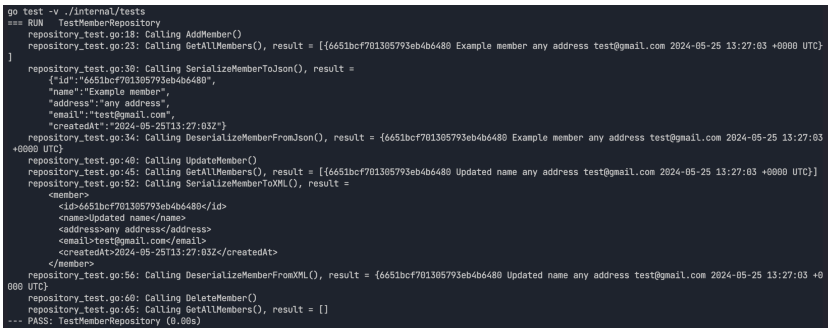

Рисунок 4.1 – Тестирование MemberRepository

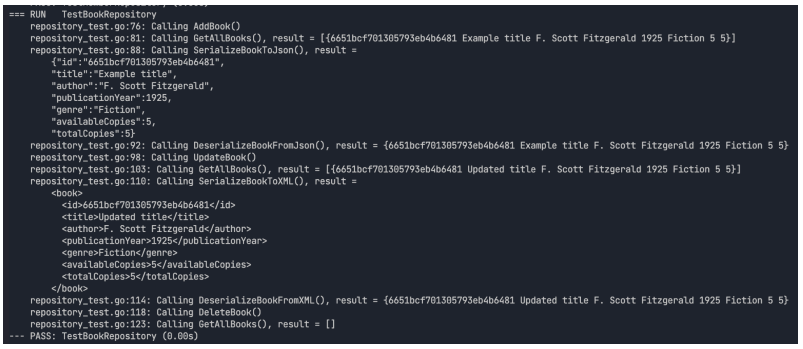

Рисунок 4.2 – Тестирование BookRepository

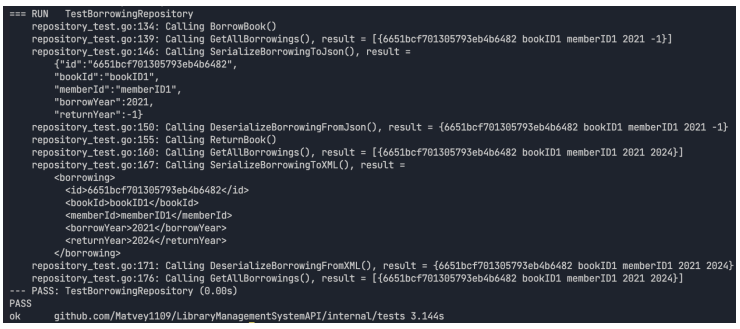

Рисунок 4.3 – Тестирование BorrowingRepository

Благодаря проведенному итоговому тестированию не было обнаружено никаких ошибок или неисправностей в работе приложения.

5 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

Требуется убедиться, что серверная часть системы настроена. Для начала работы необходимо запустить приложение. После запуска приложения откройте веб-браузер и введите адрес сервера в адресной строке.

Пользователю будет показана корневая веб-страница с текстом. Эта страница будет содержать информацию о приложении.

Чтобы получить доступ к интерактивной документации системы, введите в адресной строке браузера URL /swagger. Например, если адрес сервера – http://localhost:8080, то введите http://localhost:8080/swagger. Нажмите клавишу Enter или выполните запрос, чтобы перейти на страницу с интерактивной документацией.

На странице интерактивной документации вы найдете описание основных функций и возможностей системы менеджмента библиотеки. Это включает добавление и удаление книг, управление пользователями, поиск и фильтрацию данных, а также другие функции, связанные с библиотечным управлением.

В интерактивной документации будет представлено описание API-интерфейсов, которые могут использоваться для взаимодействия с системой. Это набор методов, которые клиентские приложения могут вызывать для выполнения различных операций в системе.

Описание API-интерфейсов включает информацию о доступных методах, параметрах, возвращаемых значениях и кодах состояния HTTP.

ЗАКЛЮЧЕНИЕ

В рамках данного курсового проекта была разработана серверная часть системы менеджмента библиотеки, предназначенная для автоматизации основных библиотечных процессов и улучшения доступа к ресурсам библиотеки. Целью проекта было повысить эффективность работы библиотеки, упростить взаимодействие между библиотекарями и читателями, а также улучшить качество обслуживания.

В ходе работы был проведен анализ требований к системе менеджмента библиотеки, что позволило определить функциональные и нефункциональные требования. Была разработана архитектура системы, включающая основные компоненты и интерфейсы. Затем было разработано и реализовано программное обеспечение, включающее онлайн-каталог книг, функционал для регистрации читателей, учета выданных книг и управления сроками их возврата.

Проверка работоспособности приложения показала его эффективность и функциональность. Система менеджмента библиотеки позволяет библиотекарям легко управлять ресурсами библиотеки, осуществлять поиск, выдачу и возврат книг, а также вести учет читателей. Онлайн-каталог книг обеспечивает удобный и быстрый доступ к информации о доступных книгах, а также проверку их наличия.

В результате данного курсового проекта были достигнуты поставленные цели и выполнены задачи. Разработанная система менеджмента библиотеки позволит библиотеке более эффективно управлять своими ресурсами и повысить качество обслуживания для читателей. Она упростит процессы хранения, поиска, выдачи и возврата книг, а также облегчит взаимодействие между библиотекарями и читателями.

В заключение можно отметить, что разработка системы менеджмента библиотеки имеет большую практическую значимость и может быть применена в различных библиотеках для автоматизации и оптимизации библиотечных процессов.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] TIOBE Index [Электронный ресурс]. – Режим доступа : https://www.tiobe.com/tiobe-index/.

[2] The Go Programming Language [Электронный ресурс]. – Режим доступа : https://go.dev/.

[3] MongoDB Documentation [Электронный ресурс]. – Режим доступа : https://www.mongodb.com/docs/.

[4] GitHub [Электронный ресурс]. – Режим доступа : https://github.com/.

[5] Docker Docs [Электронный ресурс]. – Режим доступа : https://docs.docker.com/.

[5] Архитектура REST / Хабр [Электронный ресурс]. – Режим доступа : https://habr.com/ru/articles/38730/.

[6] ООП / Хабр [Электронный ресурс]. – Режим доступа : https://habr.com/ru/articles/463125/.

ПРИЛОЖЕНИЕ А
(обязательное)
Листинг программного кода

```
module github.com/Matvey1109/LibraryManagementSystemAPI

go 1.21.3

require (
	github.com/joho/godotenv v1.5.1 // indirect
	github.com/julienschmidt/httprouter v1.3.0
	github.com/rs/cors v1.10.1
	go.mongodb.org/mongo-driver v1.15.0 // indirect
)

type Member struct {
	ID        string    `json:"id"`
	Name      string    `json:"name"`
	Address   string    `json:"address"`
	Email     string    `json:"email"`
	CreatedAt time.Time `json:"createdAt"`
}

type Book struct {
	ID              string `json:"id"`
	Title           string `json:"title"`
	Author          string `json:"author"`
	PublicationYear int    `json:"publicationYear"`
	Genre           string `json:"genre"`
	AvailableCopies int    `json:"availableCopies"`
	TotalCopies     int    `json:"totalCopies"`
}

type Borrowing struct {
	ID         string `json:"id"`
	BookID     string `json:"bookId"`
	MemberID   string `json:"memberId"`
	BorrowYear int    `json:"borrowYear"`
	ReturnYear int    `json:"returnYear,omitempty"`
}

// ! Abstract Factory
type StorageFactory interface {
	CreateStorage() (Storage, error)
}

func GetStorageFactory() (StorageFactory, error) {
	typeOfStorage, _, _ := loadenv.LoadGlobalEnv()

	if typeOfStorage == "local" {
```

```
          return &LocalStorageFactory{}, nil
     }

     if typeOfStorage == "mongodb" {
          return &MongoDBStorageFactory{}, nil
     }

     return nil, errors.New("typeOfStorage not found")
}

// ! Concrete Factories
type LocalStorageFactory struct{} // * Implements interface
StorageFactory

func (f *LocalStorageFactory) CreateStorage() (Storage, error) {
     return &LocalStorage{
          members:    []models.Member{},
          books:      []models.Book{},
          borrowings: []models.Borrowing{},
     }, nil
}

type MongoDBStorageFactory struct{} // * Implements interface
StorageFactory

func (f *MongoDBStorageFactory) CreateStorage() (Storage, error)
{
     MONGO_INITDB_ROOT_USERNAME, MONGO_INITDB_ROOT_PASSWORD :=
loadenv.LoadMongoEnv()
     uri := fmt.Sprintf("mongodb://%s:%s@mongo:27017/",
MONGO_INITDB_ROOT_USERNAME, MONGO_INITDB_ROOT_PASSWORD)

     client, err := mongo.Connect(context.TODO(),
options.Client().ApplyURI(uri))
     if err != nil {
          return nil, err
     }

     if err := client.Ping(context.TODO(), nil); err != nil {
          return nil, fmt.Errorf("failed to ping MongoDB: %w",
err)
     }

     database := client.Database("mydatabase")
     membersCollection := database.Collection("members")
     booksCollection := database.Collection("books")
     borrowingsCollection := database.Collection("borrowings")

     return &MongoDBStorage{
          client:              client,
          membersCollection:   membersCollection,
          booksCollection:     booksCollection,
```

```
          borrowingsCollection: borrowingsCollection,
     }, nil
}

// ! Abstract Product
type Storage interface {
     GetAllMembersStorage() ([]models.Member, error)
     GetMemberStorage(id string) (models.Member, error)
     AddMemberStorage(member models.Member) error
     UpdateMemberStorage(id string, member models.Member) error
     DeleteMemberStorage(id string) error

     GetAllBooksStorage() ([]models.Book, error)
     GetBookStorage(id string) (models.Book, error)
     AddBookStorage(book models.Book) error
     UpdateBookStorage(id string, book models.Book) error
     DeleteBookStorage(id string) error

     GetAllBorrowingsStorage() ([]models.Borrowing, error)
     GetBorrowingStorage(id string) (models.Borrowing, error)
     AddBorrowingStorage(borrowing models.Borrowing) error
     UpdateBorrowingStorage(id string, borrowing
models.Borrowing) error
     DeleteBorrowingStorage(id string) error
}

var (
     ExportStorageFactory, _ = GetStorageFactory()
     ExportStorage, _        =
ExportStorageFactory.CreateStorage()
)

// ! Implements interface Storage
type MongoDBStorage struct {
     client               *mongo.Client
     membersCollection    *mongo.Collection
     booksCollection      *mongo.Collection
     borrowingsCollection *mongo.Collection
}

var _ Storage = (*MongoDBStorage)(nil) // Checker

// * Member
func (ms *MongoDBStorage) GetAllMembersStorage()
([]models.Member, error) {
     cursor, err :=
ms.membersCollection.Find(context.Background(), bson.M{})
     if err != nil {
          return nil, err
     }
     defer cursor.Close(context.Background())
```

```
	var members []models.Member
	for cursor.Next(context.Background()) {
		var (
			mongoMemberMap map[string]interface{}
			member         models.Member
		)

		err := cursor.Decode(&mongoMemberMap)
		if err != nil {
			return nil, err
		}

		for key, value := range mongoMemberMap {
			if key == "_id" {
				member.ID = value.(primitive.ObjectID).Hex()
			} else if key == "name" {
				member.Name = value.(string)
			} else if key == "address" {
				member.Address = value.(string)
			} else if key == "email" {
				member.Email = value.(string)
			} else if key == "createdAt" {
				curTime := value.(primitive.DateTime).Time()
				member.CreatedAt, _ =
time.Parse(time.DateTime, curTime.Format(time.DateTime))
			}
		}
		members = append(members, member)
	}
	return members, nil
}
```

ПРИЛОЖЕНИЕ Б
(обязательное)
Алгоритмы, используемые в программном средстве

Блок-схемы алгоритмов указаны на рисунках Б.1, Б.2, Б.3.

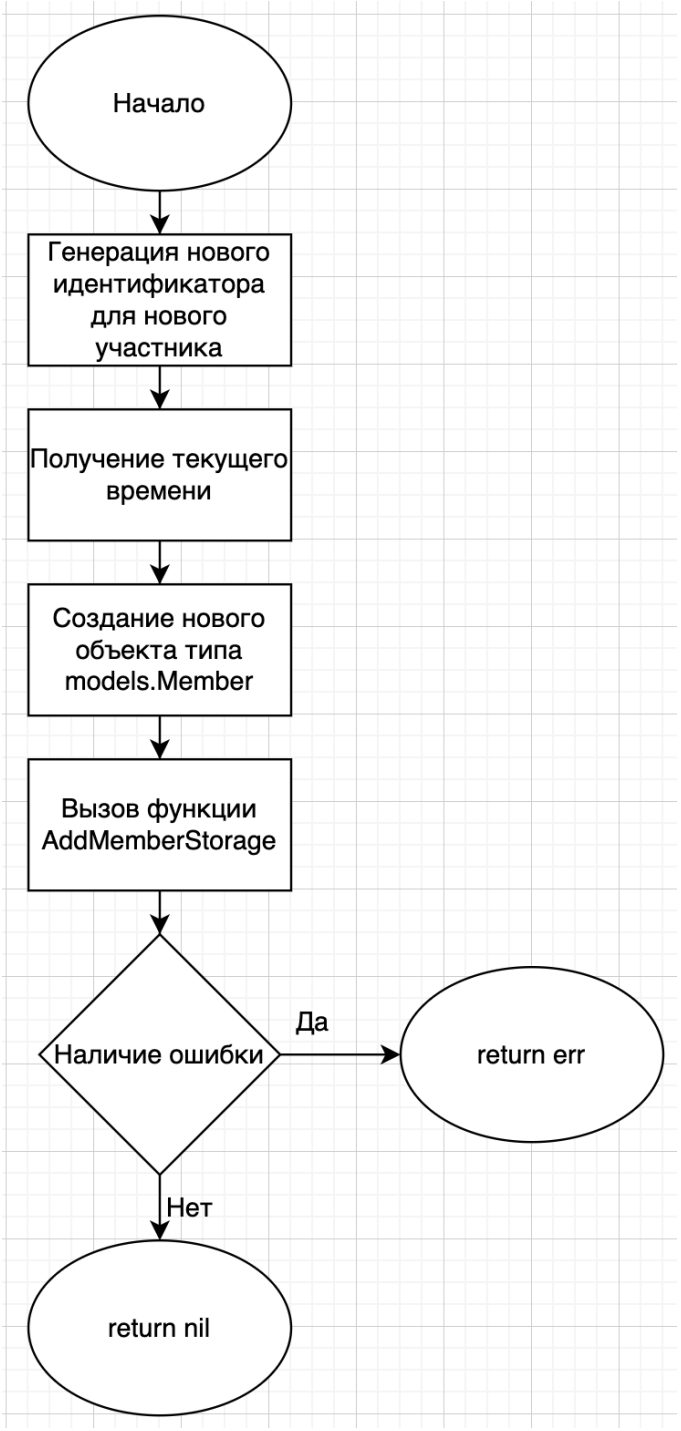

Рисунок Б.1 – Блок-схема функции добавления члена библиотеки

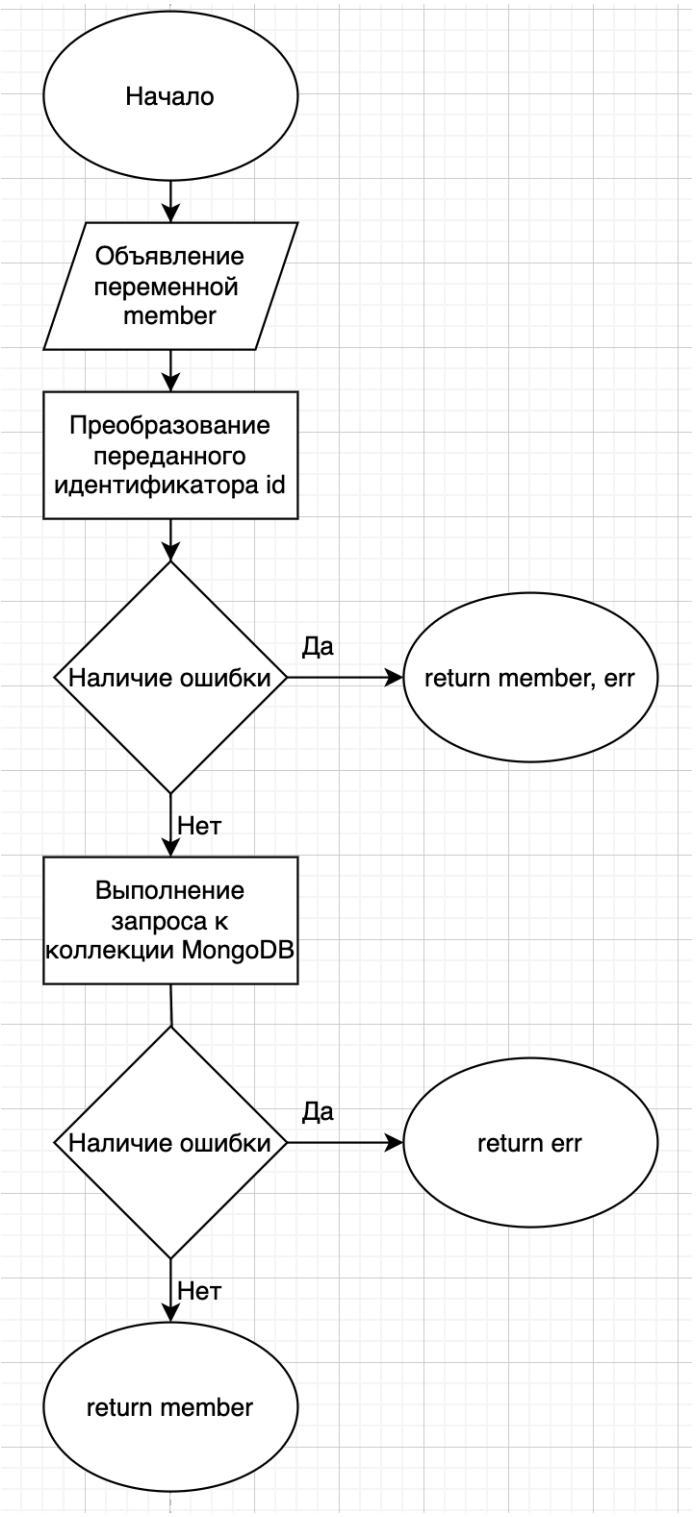

Рисунок Б.2 – Блок-схема функции получения члена библиотеки из базы данных

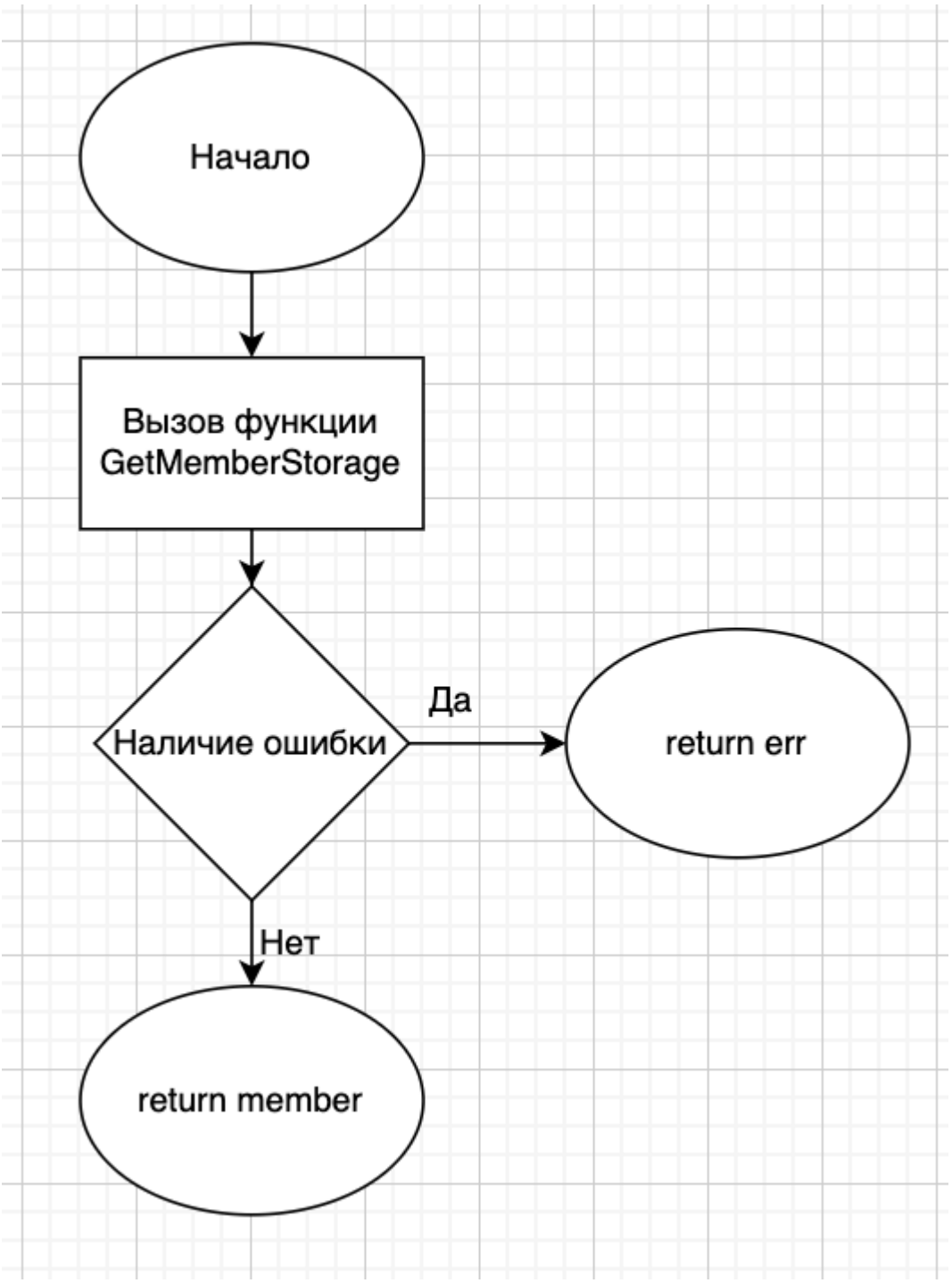

Рисунок Б.3 – Блок-схема функции получения члена библиотеки

Обозначение	Наименование	Дополнительные сведения
	Текстовые документы	
БГУИР КП 1-40 04 01	Пояснительная записка	34 с.
	Графические документы	
ГУИР 253505 010 СА	Серверная часть системы менеджмента библиотеки. Схемы алгоритмов	Формат А4

					БГУИР КП 1-40 04 01			
Изм.	Л.	№ докум.	Подп.	Дата	Серверная часть системы менеджмента библиотеки		Лист	Листов
Разраб.		*Косяков М.М.*		31.05.24	Ведомость курсового проекта	Т	34	34
Пров.		*Тушинская Е.В.*		31.05.24		Кафедра Информатики гр. 253505		