Федеральное государственное автономное
образовательное учреждение
высшего образования
«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»
Институт космических и информационных технологий
институт
Информационных систем
кафедра

УТВЕРЖДАЮ
Заведующий кафедрой ИС
_____ Дьячук П.П.
подпись фамилия, инициалы
« 21 » июля 2019 г.

БАКАЛАВРСКАЯ РАБОТА

09.03.02 Информационные системы и технологии

Автоматизация процесса учета книг в библиотеке

Руководитель _____ канд. техн. наук, доцент каф. И.А. Легалов
подпись, дата ученая степень, должность инициалы, фамилия

Студент КИ15-13Б 031509269 _____ С.А. Маркушин
номер группы номер зачетной книжки подпись, дата инициалы, фамилия

Нормоконтролер _____ Ю.В. Шмагрис
подпись, дата инициалы, фамилия

Красноярск 2019

Федеральное государственное автономное
образовательное учреждение
высшего образования
«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»
Институт космических и информационных технологий
институт
Информационных систем
кафедра

УТВЕРЖДАЮ
Заведующий кафедрой ИС
_____ Дьячук П.П.
подпись фамилия, инициалы
« » 2019 г.

ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ
в форме бакалаврской работы

Студенту: Маркушину Сергею Андреевичу

Группа: КИ15-13Б Направление: 09.03.02 «Информационные системы и технологии»

Тема выпускной квалификационной работы: Автоматизация процесса учета книг в библиотеке

Утверждена приказом по университету № 7237/с от 24.05.2019 г.

Руководитель ВКР: И. А. Легалов, канд. техн. наук, доцент кафедры «Информационные системы» ИКИТ СФУ

Исходные данные для ВКР: Требования к разрабатываемому веб-приложению, рекомендации руководителя, учебные материалы

Перечень разделов ВКР: Введение, анализ предметной области, проектирование веб приложения, программная реализация приложения, принципы работы с приложением, заключение, список использованных источников, список сокращений.

Перечень графического материала: презентация, выполненная в Microsoft Office PowerPoint 2016.

Руководитель ВКР _____ И. А. Легалов

подпись

Задание принял к исполнению _____ С.А. Маркушин

подпись

« » _____ 2019 г.

РЕФЕРАТ

Выпускная квалификационная работа «Автоматизация процесса учета книг в библиотеке» содержит 45 страниц текстового документа, 16 использованных источников.

ВЕБ – ПРИЛОЖЕНИЕ, ASP .NET Core, EF, MVC, Dependency Injection, IoC, HTML, CSS, BOOTSTRAP, MS SQL, трехуровневая архитектура, Code First, Use Case, ER.

Объектом исследования является библиотека.

Предметом исследования является процесс учета книг в библиотеке.

Цель работы – автоматизация процесса учета книг в библиотеке.

Для достижения поставленной цели были выдвинуты нижеследующие задачи:

1. провести анализ предметной области;
2. рассмотреть аналоги приложения;
3. определить инструментальные средства, которые будут использоваться в работе;
4. разработать бизнес-логику и интерфейс;
5. описать работу полученной в результате системы.

В результате выполнения выпускной квалификационной работы было разработано веб-приложение «AutoBook».

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Книги являются одним из наших самых больших ресурсов для получения знаний. Сотни лет человечество совершенствует технологии их издания. Глиняные таблички сменились электронными экземплярами, большие библиотеки с тысячами книг были переведены в цифру. В наше время главной задачей остается улучшение качества обслуживания библиотек.

Более удобное обслуживание библиотек - это не только комфорт и эффективность для посетителей. Но также значительное уменьшение рутинной работы для сотрудников, удобство обработки фондов с помощью современных инструментов. Достигнуть эту цель позволит автоматизация библиотек, в результате которой традиционные технологии соединяются с новейшими.

Автоматизированные библиотечно-информационные системы получили распространение в последние десятилетия. Накоплен опыт по внедрению инновационных методов. Данные систематизируются, каталоги стали электронными, книжный фонд подлежит автоматизированному учету. Поиск стал удобен в плане доступа и предоставления информации по необходимым книгам.

21 век – это век Интернета. Использование интернета для библиотек заключается в том, что они могут разместить там свои каталоги и воспользоваться чужими для удовлетворения интересов читателей. На практике они создают свои Web-сервера, которые включают ряд страниц, представляющих данное учреждение. Возможность заказывать литературу и удаленный поиск встречается редко. В связи с этим эффективность использования интернета можно назвать весьма. Пока в рамках автоматизации российских библиотек 2018 осуществляется лишь каталогизация и учет. Но применение информационных систем может позволить автоматизировать все функции этих учреждений. Библиотекам следует для оценки своей работы постепенно перейти к показателю считывания информации как таковой, а не выдачи документа.

Очень важным шагом при автоматизации, является процесс постановки задач, поскольку от этого выбора будет зависеть дальнейшее развитие организации в целом и определяться как материально-техническое, так и информационное состояние каждого подразделения этой библиотеки.

В каждой библиотеке существуют определенные виды работ, которые «не мешало бы» передать в ведение компьютера. И с каждым годом все большее количество задач можно решить, используя вычислительную технику. В связи с этим возникает желание ничего не делать, а дождаться «идеальной» программы, которая бы все это делала при нажатии одной клавиши. Но такая ситуация является с большой долей вероятности недостижимой. На настоящее время лучшим выбором будет заложить в разрабатываемое приложение возможность расширять функционал, добавлять новые функции без ущерба работе системы в целом.

Целью настоящей выпускной квалификационной работы является автоматизация процесса учета книг в библиотеке.

В рамках разработки проекта необходимо решить следующие задачи:

1. провести анализ предметной области;
2. спроектировать веб приложение;
3. разработать программную реализацию приложения;
4. описать работу полученной в результате системы.

1 Анализ предметной области

1.1 Требования к функционалу приложения

Внедрение автоматизации в библиотеку должно повысить производительность и качество труда работников библиотеки, эффективно обеспечивать пользователя необходимыми ему данными и ресурсами.

Чтобы достичь вышесказанного необходимо обеспечить:

- доступ к каталогу и возможность заказа книг через Web-приложение;
- возможность более быстрого поиска необходимой читателю литературы;
- удобство пользования интерфейсом приложения для работников библиотеки;
- возможность увеличения скорости оформления и списания книг фонда;
- автоматическое составление отчетов;
- упрощение процедуры инвентаризации;
- возможность увеличения скорости и повышение объективности получения информации об итогах движения фонда;
- возможность более быстрого процесса выдачи и возврата книг;

Из всего вышеперечисленного следует, что, целью решения задачи автоматизации является снижение времени, затрачиваемого библиотекарем на оформление различных документов, связанных с выдачей и приемом книг, снижение до минимума количества ошибок, допускаемых при заполнении, оптимизация сбора библиотечной статистики.

1.2 Описание аналогов

Рынок библиотечных продуктов не стоит на месте, все новые и новые приложения для учета литературы и онлайн покупки книг выходят на свет.

Их можно разделить на 3 большие группы.

1. онлайн библиотеки с возможностью покупки экземпляров в электронном виде;
2. приложения для небольших библиотек с фондом менее 100 000 книг. (Для школ, сельских библиотек, личных коллекций и т.д.);
3. приложения для крупных библиотек с фондом более 100 000 книг таких как муниципальные, вузовские, государственные библиотеки.

К 1 группе можно отнести известную компанию Литрес, каталог которой насчитывает 1 000 000 электронных книг на русском и иностранных языках, 48 000 бесплатных книг.

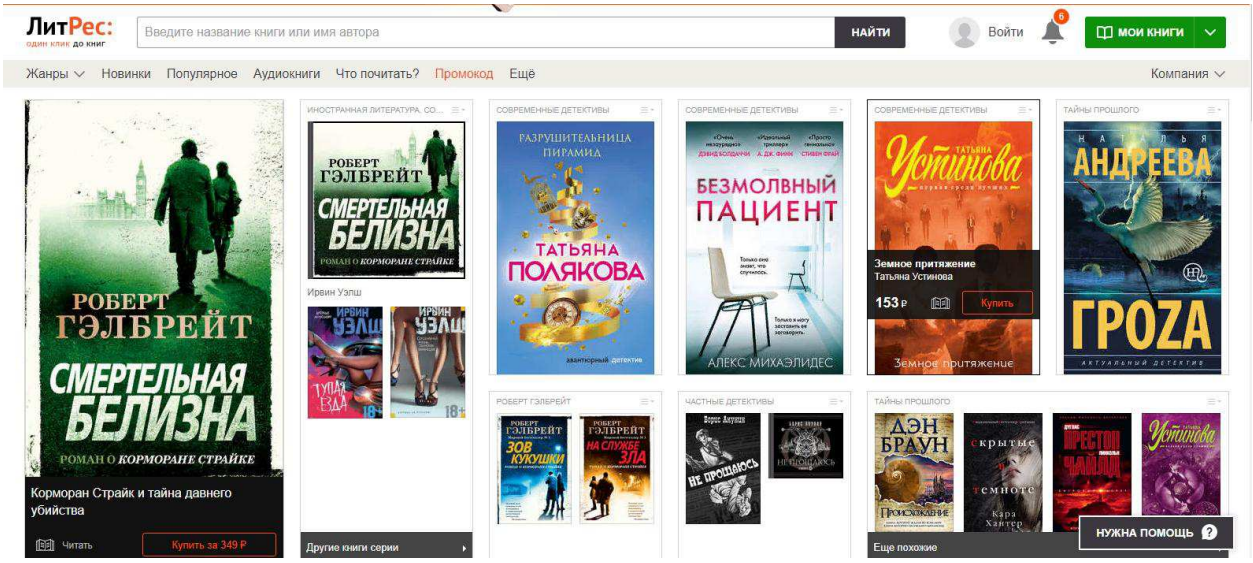


Рисунок 1 – Сайт www.litres.ru

Ещё несколько лет назад не было потребности в электронных книгах, но в последнее десятилетнее начался стремительный рост продаж электронных экземпляров книг и всевозможных устройств, с помощью которых можно прочитать нужную литературу в электронном виде.

Литрес - это больше, чем электронная библиотека**.** Как сказано на сайте, ресурс сотрудничает с реальными библиотеками, благодаря которым читатель получает доступ к самым интересным произведениям. Помимо специализированной литературы читателя интересуют книги развлекательного жанра: фантастика, детективы, проза, любовные романы, а также деловая литература: бизнес-книги, периодика и др. Компания «ЛитРес» единственная на книжном рынке предлагает библиотекам самый обширный каталог популярной литературы широкого профиля на русском языке в электронном виде.

ЭБС ЛитРес обеспечивает простую настройку и управление работы сервиса для персонала библиотек, а также быструю и удобную навигацию сервиса для читателей. Инструкция по подключению, управлению и формированию доступов в системе библиотеки ЛитРес позволяет оперативно обучить персонал. Для работы с ЭБС ЛитРес достаточно компьютеров, планшетов или любых других устройств и доступа в Интернет.

К 2 группе можно отнести программу для библиотек, книголюбов и коллекционеров – LibaBook. Она представляет собой универсальный каталогизатор любых предметов. В программе можно вести учёт и каталогизацию бумажных книг домашней/муниципальной библиотеки и даже, картотеку марок, DVD дисков и вообще чего угодно.

Рисунок 2 – Приложение LibaBook

Программа LibaBook позволяет быстро найти, где хранится нужный предмет (на какой полке, в каком шкафу, в какой комнате…), в каком состоянии он находится, кому и когда его выдавали, когда должны вернуть и т.д.

Базовые функции программы доступны совершенно бесплатно, а если зарегистрироваться и приобрести лицензионный ключ, то можно получить доступ к дополнительным возможностям.

Основные функции приложения:

- создание Базы Данных, которая хранится на компьютере;
- поиск описаний книг в интернете;
- просмотр истории перемещения книг/предметов;
- экспорт списка книг/предметов в формат csv, html;
- создание отчётов и просмотр статистики;
- поиск книг через видеокамеру по штрихкоду;
- автоматический и полуавтоматический поиск описания книг в интернете (платно);
- создание и распечатка своих штрихкодов.

К 3 группе можно отнести Российскую государственную библиотеку.

В фондах Российской государственной библиотеки хранятся 46,9 миллионов учётных единиц. Чтобы найти то, что нужно, создана целая система каталогов - Единый электронный каталог.

Единый электронный каталог содержит библиографические записи на все виды документов, включая статьи, изданные на русском и других языках на

различных носителях и в различные хронологические периоды. Можно вести поиск не только в едином каталоге, но и в отдельных каталогах, ограничивая поиск по определенному виду документа: каталог рукописей, диссертаций, нот, газет и др.

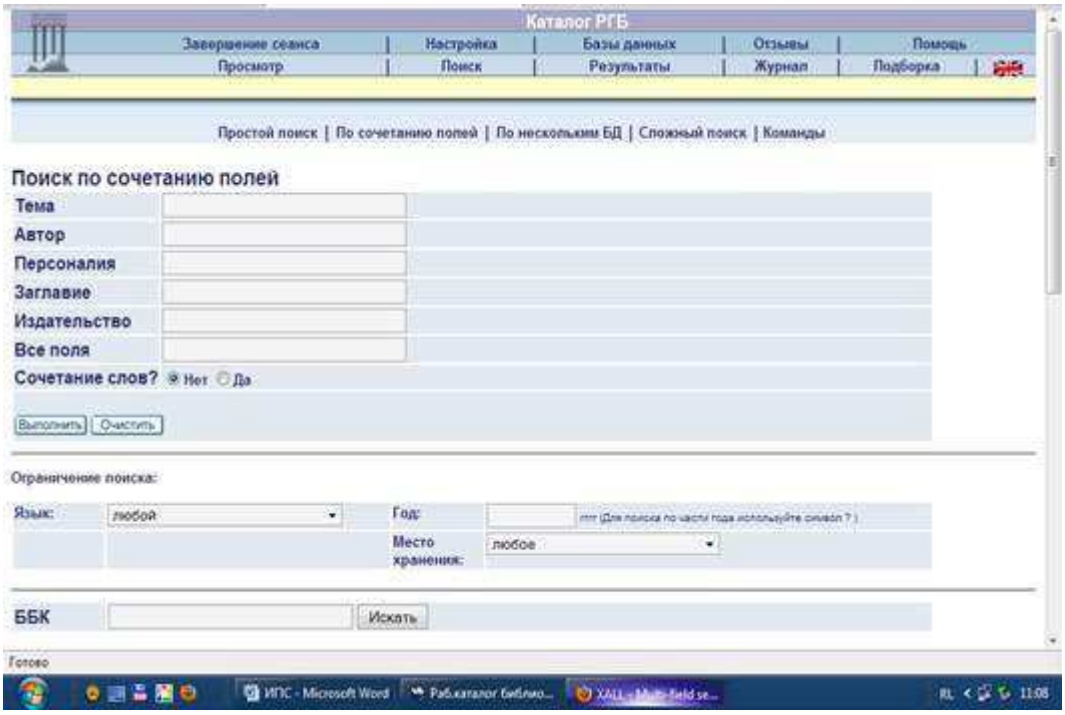


Рисунок 3 – Каталог Российской государственной библиотеки

Доступ к электронному каталогу осуществляется:

1. с компьютеров в читальных залах (с возможностью электронного заказа);
2. с читательских терминалов;
3. с любого компьютера через интернет (с возможностью электронного заказа).

2 Проектирование веб приложения

2.1 Используемые технологии

2.1.1 Язык программирования C#

На сегодняшний момент язык программирования C# является одним из самых мощных, быстро развивающихся и востребованных языков в ИТ-отрасли. В настоящий момент на нем можно написать все что угодно начиная от маленьких desktop приложений до крупных веб-сервисов, обслуживающих миллионы пользователей.

C# достаточно молодой язык, он появился на свет в июне 2000 г. в результате совместной работы большой группы разработчиков Microsoft. Авторами стали Скотт Вилтамут и Андерс Хейльсберг – создатель Turbo Pascal и Delphi, устроившийся в компанию в 1996 году.

Появление языка си шарп и инициативы .NET отнюдь не случайно совпало с началом лета 2000 г. Именно к этому моменту компания Microsoft подготовила промышленные версии новых компонентных технологий и решений в области обмена сообщениями и данными, а также создания Интернет-приложений (COM+, ASP+, ADO+, SOAP). Для продвижения новинок Microsoft создала инструментарий для разработчиков с обширной поддержкой. Кроме того, компания уже не могла добавлять новый функционал из-за обширной кодовой базы, возникающих конфликтов совместимости, требований поддержки нового оборудования. Наступил момент, когда нужно было начать с чистого листа для того, чтобы создать простой, но имеющий сложную структуру набор языков, сред и средств разработки, которые позволят разработчику легко создавать современные программные продукты.

C# и .NET стоят в начале пути к этой цели. Если говорить простым языком, то .NET представляет собой новую платформу, новый API для создания приложений под Windows, а C# ее новый язык, разработанный с нуля, для работы с этой платформой.

Необходимо отметить, что обратная совместимость сохранена. Программы, разработанные на более старой версии языка, будут выполняться и в новых его версиях. Платформа.NET была спроектирована таким образом, чтобы она могла функционировать с имеющимся программным обеспечением

На данный момент текущей версией языка является C# 7.1, который вышел в августе 2017 года. C# является объектно-ориентированным, он поддерживает полиморфизм, наследование, перегрузку операторов, статическую типизацию. Объектно-ориентированный подход позволяет решить задачи по построению крупных, но в тоже время гибких, масштабируемых и расширяемых приложений.

Особенности C#:

1. полный и хорошо определенный набор основных типов;
2. встроенная поддержка автоматической генерации XML-документации;
3. автоматическое освобождение динамически распределенной памяти;

4. полный доступ к библиотеке базовых классов .NET, а также легкий доступ к Windows API;

5. указатели и прямой доступ к памяти, если они необходимы. (Однако язык разработан таким образом, что практически во всех случаях можно обойтись и без этого);

6. возможность использования C# для написания динамических web-страниц ASP.NET;

7. простое изменение ключей компиляции.

C# продолжает активно развиваться, и с каждой новой версией появляется все больше интересных нововведений, как, например, лямбды, динамическое связывание, асинхронные методы и т.д.

2.1.2 Asp .NET Core

Перед тем как углубляться в детали, обсудим среду .NET в целом и место, которое в ней занимают .NET Core и .NET Standard.

Платформа .NET Framework появилась 15 лет назад. Тогда в её состав входил только один стек технологий .NET, с помощью которого можно было разрабатывать приложения для ПК под управлением Windows и веб-приложения. Но шло время и с тех пор появились и другие реализации .NET, например, Xamarin для разработки мобильных приложений для iOS и Android и классических приложений для macOS. Потом пришел mono фреймворк, появилась идея о .NET vNext в 2014г. Стало понятно, что классический .NET Freamwork не обладает должной гибкостью. Так возникла идея о стандартизации базовых API.

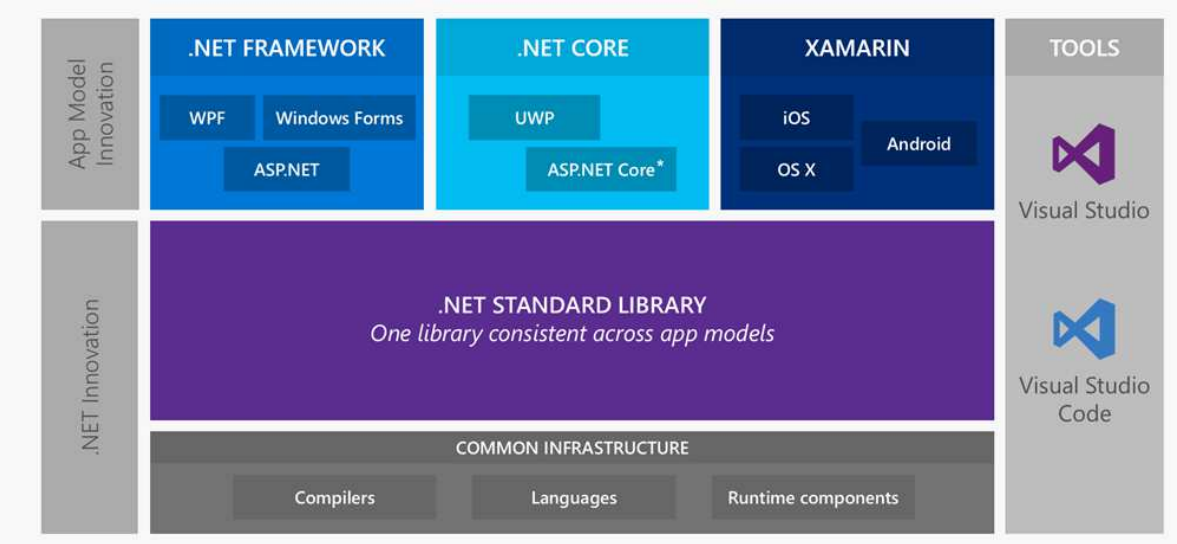


Рисунок 4 – Среда .NET

.NET Standard - это набор базовых API (другое их название - BCL, библиотека базовых классов), которые должны поддерживаться во всех реализациях .NET. .NET Standard позволяет создавать библиотеки, подходящие для любых приложений .NET, вне зависимости от реализации .NET или операционной системы, в которой они выполняются.

.NET Core - это открытая универсальная платформа разработки, которая поддерживается корпорацией Майкрософт и сообществом .NET на сайте GitHub. Она является кроссплатформенной, поддерживает Windows, Mac OS и Linux и может использоваться на устройствах, в облаке, во внедренных системах и в сценариях IoT (Интернета вещей). В её основе лежат технологии .NET Framework и Silverlight. Она оптимизирована для мобильных и серверных рабочих нагрузок.

ASP.NET Core включает в себя фреймворк MVC, который объединяет функциональность MVC, Web API и Web Pages. Данные технологии в предыдущих версиях реализовались отдельно и поэтому содержали много дублирующей функциональности. На данный момент они объединены в одну общую программную модель ASP.NET Core MVC. А Web Forms полностью ушли в прошлое.

Главные отличия ASP.NET Core от прошлых версий ASP.NET:

1. новый легковесный и модульный конвейер HTTP-запросов;
2. возможность развертывать приложение как на IIS, так и в рамках своего собственного процесса;
3. использование платформы .NET Core и ее функциональности;
4. распространение пакетов платформы через NuGet;
5. интегрированная поддержка для создания и использования пакетов NuGet;
6. расширяемость;
7. единый стек веб-разработки, сочетающий Web UI и Web API;
8. конфигурация для упрощенного использования в облаке;
9. встроенная поддержка для внедрения зависимостей;
10. кроссплатформенность: возможность разработки и развертывания приложений ASP.NET на Windows, Mac и Linux;
11. развитие как open source проекта, открытость к изменениям.

2.1.3 Microsoft SQL Server

Microsoft SQL Server - система управления реляционными базами данных (СУБД), разработанная корпорацией Microsoft. Первая версия вышла в 1987 году.

Центральным аспектом в MS SQL Server, как и в любой СУБД, является база данных. База данных представляет хранилище данных, организованных определенным способом. Нередко физически база данных представляет файл на жестком диске, хотя такое соответствие необязательно. Для хранения и администрирования баз данных применяются системы управления базами данных (database management system) или СУБД (DBMS). И как раз MS SQL Server является одной из такой СУБД.

Для организации баз данных MS SQL Server использует реляционную модель. Эта модель баз данных была разработана еще в 1970 году Эдгаром Коддом. А на сегодняшний день она фактически является стандартом для организации баз данных.

Реляционная модель предполагает хранение данных в виде таблиц, каждая из которых состоит из строк и столбцов. Каждая строка хранит отдельный объект, а в столбцах размещаются атрибуты этого объекта.

Для идентификации каждой строки в рамках таблицы применяется первичный ключ (primary key). В качестве первичного ключа может выступать один или несколько столбцов. Используя первичный ключ, мы можем ссылаться на определенную строку в таблице. Соответственно две строки не могут иметь один и тот же первичный ключ.

Через ключи одна таблица может быть связана с другой, то есть между двумя таблицами могут быть организованы связи. А сама таблица может быть представлена в виде отношения (relation).

Для взаимодействия с базой данных применяется язык SQL. Клиент отправляет запрос на языке SQL посредством специального API. СУБД должным образом интерпретирует и выполняет запрос, а затем посылает клиенту результат выполнения.

Microsoft SQL Server в качестве языка запросов использует версию SQL, получившую название Transact-SQL. T-SQL позволяет использовать дополнительный синтаксис для хранимых процедур и обеспечивает поддержку транзакций (взаимодействие базы данных с управляющим приложением).

В зависимости от задачи, которую выполняет команда T-SQL, он может принадлежать к одному из следующих типов:

1. DDL (Data Definition Language - Язык определения данных). К этому типу относятся различные команды, которые создают базу данных, таблицы, индексы, хранимые процедуры и т.д. В общем определяют данные;

2. DML (Data Manipulation Language - Язык манипуляции данными). К этому типу относят команды на выбор данных, их обновление, добавление, удаление - в общем все те команды, с помощью которыми мы можем управлять данными;

3. DCL (Data Control Language - Язык управления доступа к данным). К этому типу относят команды, которые управляют правами по доступу к данным. В частности, это следующие команды.

Версия SQL Server 2017 – это огромный шаг вперед на пути к платформе, универсальной для многих языков, типов данных, онпремисного софта и облачных хранилищ, доступной как для Linux и Linux Docker-контейнеров, так и для традиционной Windows.

2.1.4 Entity Framework Core

Entity Framework Core (EF Core) представляет собой объектно-ориентированную, легковесную и расширяемую технологию от компании Microsoft для доступа к данным. EF Core является ORM-инструментом (object-relational mapping - отображения данных на реальные объекты). То есть EF Core позволяет работать базами данных, но представляет собой более высокий уровень абстракции: EF Core позволяет абстрагироваться от самой базы данных и ее таблиц и работать с данными независимо от типа хранилища. Если на физическом уровне мы оперируем таблицами, индексами, первичными и

внешними ключами, но на концептуальном уровне, который нам предлагает Entity Framework, мы уже работаем с объектами.

EF Core предоставляет универсальный API для работы с данными. Если сменить целевую СУБД, то основные изменения в проекте будут касаться прежде всего конфигурации и настройки подключения к соответствующим провайдерам. А код, который непосредственно работает с данными, получает данные, добавляет их в БД и т.д., останется прежним.

Как технология доступа к данным Entity Framework Core может использоваться на различных платформах стека .NET. Это и стандартные платформы типа Windows Forms, консольные приложения, WPF, ASP.NET 4.6/4.7. Это и новые технологии как UWP и ASP.NET Core. При этом кроссплатформенная природа EF Core позволяет задействовать ее не только на ОС Windows, но и на Linux и Mac OS X.

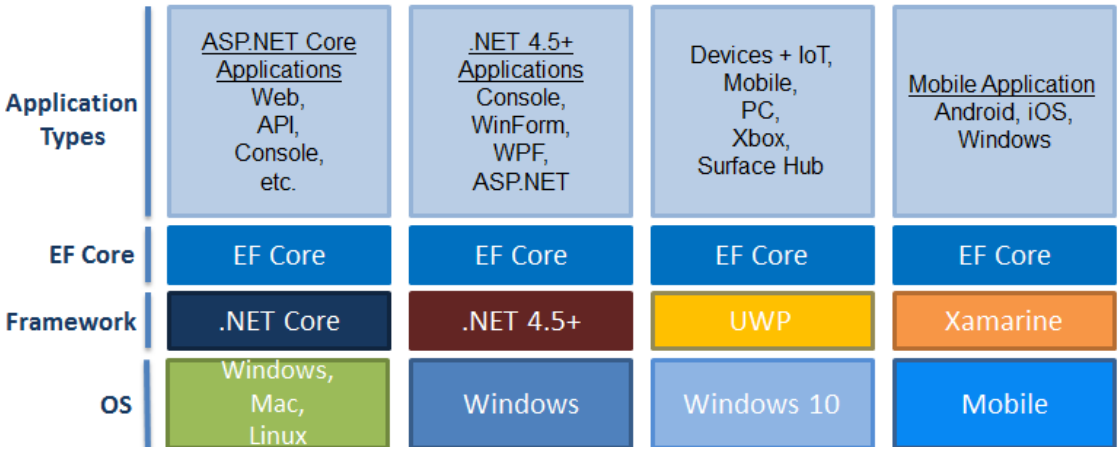


Рисунок 5 – Платформы которые поддерживает Entity Framework Core

Отличительной чертой Entity Framework Core, как технологии ORM, является использование запросов LINQ для выборки данных из БД. С помощью LINQ мы можем создавать различные запросы на выборку объектов, в том числе связанных различными ассоциативными связями. А Entity Framework при выполнении запроса транслирует выражения LINQ в выражения, понятные для конкретной СУБД (как правило, в выражения SQL).

EF Core поддерживает два подхода к разработке:

1. code-first;
2. database-first.

EF Core в основном ориентирован на подход Code-first и обеспечивает небольшую поддержку подхода database-first, поскольку визуальный конструктор или мастер для модели DB не поддерживается, начиная с EF Core 2.0.

Code-first EF Core API создает базу данных и таблицы с помощью миграции на основе соглашений и конфигурации, предоставляемых в классах домена. Этот подход полезен при проектировании на основе домена.

Database-first EF Core API создает классы домена и контекста на основе существующей базы данных с помощью команд EF Core. Это имеет

ограниченную поддержку в EF Core, поскольку он не поддерживает visual designer или wizard.

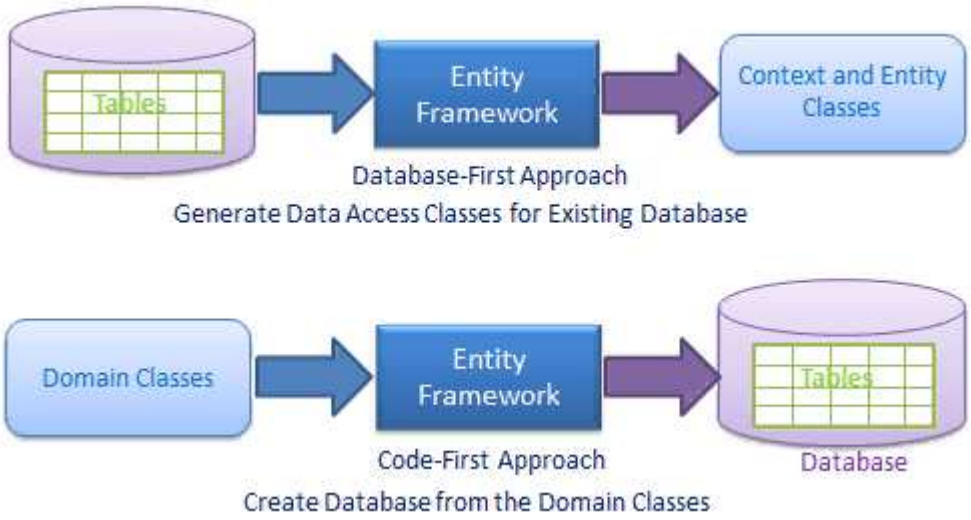


Рисунок 6 – Два подхода к разработке на EF Core

2.1.5 HTML 5 и CSS 3

HTML – это язык гипертекстовой разметки страницы, это стандарт разметки страниц для интернета, считывая этот язык браузеры интерпретируют его и отображают уже как сформированный документ. Грубо говоря, этот язык помогает вам создать тот вид документа, который вы хотите увидеть и правильно отобразить его для всех в интернете.

Язык HTML придуман уже достаточно давно, тогда этот язык был придуман для обмена научной и технической документацией, он содержал в себе не так уж и много дескрипторов, которые так же называются тегами. Благодаря этому несложному языку можно было делать хорошо структурированные и красиво отображающиеся документы.

Интернет развивался и развивался язык, усложняясь и приобретая все новые возможности. На данный момент самая последняя версия языка это HTML.

CSS – это каскадные таблицы стилей, это язык, который направлен на описание внешнего вида документа, написанного на языке гипертекстовой разметки, то есть html. По сути, это язык стилей, который имеет огромные возможности в плане формирования внешнего вида страниц. Так же этот язык непосредственно используется при верстке сайтов.

Основной задачей при разработке данного языка являлась разделение описания логической структуры, от описания внешнего вида этой страницы. Суть в том, чтобы язык html не имел лишних дополнительных параметров и атрибутов, а являлся лишь для разметки, а вот все свойства и настройки всех тегов и элементов дизайна были выделены в отдельную таблицу стилей, что собственно у них и получилось.

Первая версия языка была разработана в 1996 году, она была достаточно скромной и охватывала лишь свойства текста, цвета, выравнивание, и отступы.

Но в последующих версиях языка, функционал, безусловно, значительно вырос, и уже сейчас просто невозможно описать в нескольких строчках все его возможности. Актуальной версией языка на сегодняшний день является CSS3.

2.1.6 Sass

CSS-препроцессоры — это «программистский» подход к CSS. Они позволяют при написании стилей использовать свойственные языкам программирования приёмы и конструкции: переменные, вложенность, наследуемость, циклы, функции и математические операции. Синтаксис препроцессоров похож на обычный CSS. Код, написанный на языке препроцессора, не используется прямо в браузере, а преобразуется в чистый CSS-код с помощью специальных библиотек. Три самых известных препроцессора — это LESS, SASS и Stylus.

Sass это расширение CSS, которое придаёт мощи и элегантности этому простому языку. Sass даст вам возможность использовать переменные, вложенные правила, миксины, инлайновые импорты и многое другое, всё с полностью совместимым с CSS синтаксисом. Sass помогает сохранять огромные таблицы стилей хорошо организованными, а небольшим стилям работать быстро.

Как только Вы начинаете пользоваться Sass, препроцессор обрабатывает ваш Sass-файл и сохраняет его как простой CSS-файл, который Вы сможете использовать на любом сайте.

2.2 Архитектура приложения

Архитектура информационной системы – концепция, определяющая модель, структуру, выполняемые функции и взаимосвязь компонентов информационной системы.

Многоуровневая архитектура клиент-сервер – разновидность архитектуры клиент-сервер, в которой функция обработки данных вынесена на один или несколько отдельных серверов. Это позволяет разделить функции хранения, обработки и представления данных для более эффективного использования возможностей серверов и клиентов.

2.2.1 Трехуровневая архитектура

Выбранная для проекта архитектура является трехуровневой. Изображение принципа работы данной архитектуры представлено на рисунке.

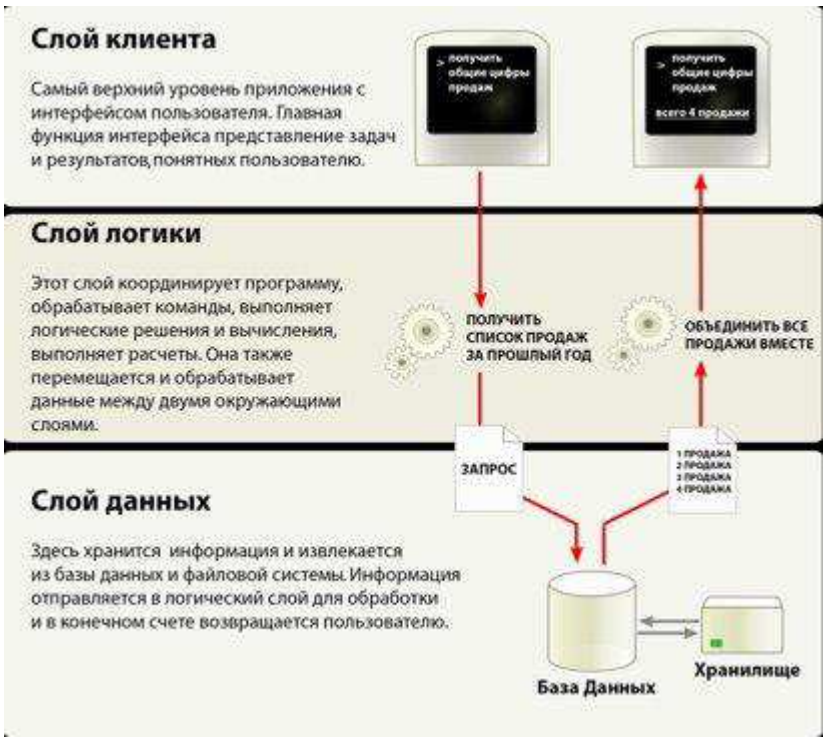


Рисунок 7 - Схема трехуровневой архитектуры

Presentation layer (слой клиента): это тот уровень, с которым непосредственно взаимодействует пользователь. Включает в себя компоненты пользовательского интерфейса, механизм получения ввода от пользователя и вывода ответов на запросы. Применительно к ASP .NET Core MVC на данном уровне расположены представления и все те компоненты, который отрисовывают пользовательский интерфейс (html, css, javascript).

Business layer (слой бизнес-логики): содержит набор компонентов, которые отвечают за обработку полученных от уровня представлений данных, реализует всю необходимую логику приложения, все вычисления, взаимодействует с базой данных и передает уровню представления результат обработки.

Data Access layer (слой доступа к данным): хранит модели, описывающие используемые сущности, также здесь размещаются специфичные классы для работы с разными технологиями доступа к данным, например, класс контекста данных который используется в Entity Framework.

Компоненты, как правило, должны быть слабосвязанными (loose coupling), поэтому неотъемлемой частью многоуровневых приложений является внедрение зависимостей (dependency injection).

2.2.2 Паттерн MVC

Также в приложении используется паттерн MVC (Model-View-Controller) - схема разделения данных приложения, пользовательского интерфейса и управляющей логики на три отдельных компонента: модель, представление и контроллер — таким образом, что модификация каждого компонента может осуществляться независимо. Принцип работы паттерна изображен на рисунке 8.

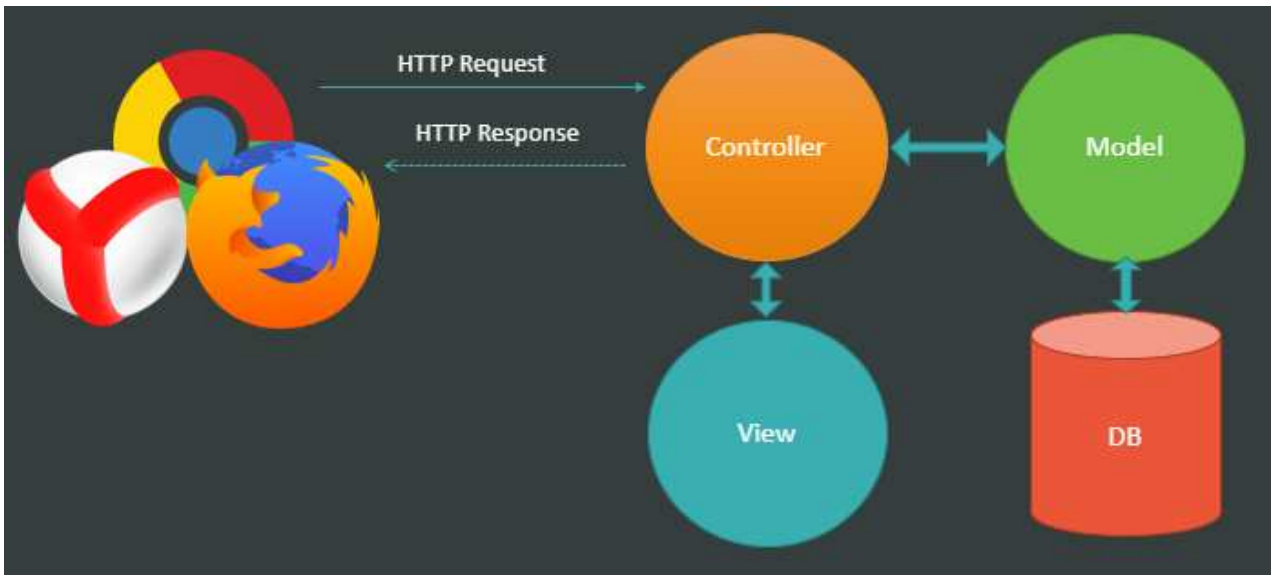


Рисунок 8 – Схема работы MVC

Модель в MVC приложении представляет состояние приложения и любую бизнес логику, и операции, которые оно должно выполнить. Бизнес логика и любая логика, касающаяся состояния приложения, должны быть инкапсулированы в модели. Строго типизированные представления обычно используют типы ViewModel, которые содержат данные для отображения в этом представлении; контроллер создаст и заполнит эти экземпляры ViewModel из модели.

Представления отвечают за отображение контекста в пользовательском интерфейсе. Они используют движок представления Razor, чтобы включить .NET код в HTML разметку. В представлениях должно быть по минимуму логики, и эта логика должна касаться отображаемого контента. Если вам нужно внести большой кусок логики в файлы представлений, чтобы отобразить данные из сложной модели, используется компонент представления, ViewModel или шаблон, чтобы упростить представление.

Контроллеры - это компоненты, которые обрабатывают пользовательские запросы, работают с моделью и выбирают представление для отображения контента. В MVC приложении представление только отображает информацию, а контроллер обрабатывает пользовательские запросы.

2.2.3 Разделение ответственности с помощью Dependency Injection

Разделение ответственности делает приложение более простым, поскольку нам легче кодировать, отлаживать и тестировать компонент (модель, представление или контроллер), который делает какую-то одну работу и следует принципу единственной обязанности. Нам довольно сложно обновлять, тестировать и отлаживать код, в котором присутствуют зависимости, общие для двух или трех этих компонентов. Например, представления, как правило, меняется чаще, чем бизнес логика. Если код представления и бизнес логика объединены в один объект, вам необходимо менять объект, содержащий бизнес логику каждый раз, когда вы вносите изменения в UI. Скорее всего, тогда у вас

появятся новые ошибки и вам потребуется заново тестировать всю бизнес логику, даже после малейшего изменения пользовательского интерфейса.

Именно для этого и служит такой механизм как Dependency Injection (внедрение зависимостей). Внедрение зависимостей (DI) - это технология, которая используется для того, чтобы по максимуму разделить объекты и зависимости. Вместо прямой установки компонентов или использования статических ссылок, объект, который нужен классу для выполнения некоторых действий, предоставляется этому классу в иной манере. Наиболее часто классы заявляют о зависимостях через конструктор, позволяя им следовать принципу явных зависимостей. Такой подход известен как «внедрение конструктора».

Если классы используют DI, их компоненты не зависят друг от друга напрямую. Это называется «Принципом инверсии зависимостей», который гласит, что «модули высокого уровня не должны зависеть от модулей низкого уровня - и те, и другие должны зависеть от абстракций». Вместо использования конкретных реализаций, классы запрашивают абстракции (обычно interface), которые предоставляются им после создания.

Если система использует DI, когда много классов используют зависимости через конструктор (или свойства), то полезно, чтобы определенные классы были связаны с нужными зависимостями. Эти классы называются контейнерами, также они называются контейнерами инверсии управления (IoC). Контейнер - это, в принципе, фабрика, которая предоставляет экземпляры требуемых типов. Если у данного типа есть зависимости, а также существует контейнер для предоставления этих зависимостей, зависимости будут созданы как часть запрашиваемого экземпляра. Таким способом сложные зависимости предоставляются классу без жесткого кодирования объекта. Кроме того, контейнеры обычно управляют жизненным циклом объекта внутри приложения.

ASP.NET Core включает в себя простой встроенный контейнер (представленный интерфейсом IServiceProvider), который по умолчанию поддерживает внедрение конструктора, и ASP.NET предоставляет через DI определенные сервисы. Контейнер ASP.NET работает с типами как с сервисами. Можно настраивать встроенные сервисы контейнера с помощью метода ConfigureServices класса Startup.

2.3 Диаграмма вариантов использования

Диаграмма вариантов использования (Use Case) - исходное концептуальное представление системы в процессе ее проектирования и разработки. С ее помощью описывается функциональное назначение системы, то, что система будет выполнять в процессе своей эксплуатации. Суть диаграммы заключается в следующем: актеры, взаимодействуют с системой через, так называемые, варианты использования.

Актер – действующее лицо, которое взаимодействует с системой через вариант использования. Действующие лица могут быть как реальными

пользователями, так и частью разрабатываемой системы или внешними событиями. Когда человек взаимодействует с системой различными способами (предполагая различные роли), он отображается несколькими действующими лицами. Например, зарегистрированным в приложении пользователем и не зарегистрированным может быть по сути одним и тем же человеком, но с разными вариантами использования. Актер графически изображается «человечком».

Вариант использования (прецеденты) - сервисы, которые система предоставляет актерам. Набор действий, которые должны привести к конкретному результату без описания внутреннего устройства данного процесса. Прецеденты это всего лишь описания типичных взаимодействий между системой и ее пользователями. Они описывают внешний интерфейс системы и определяют, что система должна делать. Прецеденты на диаграмме изображаются в виде эллипса, внутри которого пишется его название.

Отношение - связь между отдельными элементами модели. Для описания взаимодействия актеров и прецедентов существуют различные отношения. У одного актера есть возможность взаимодействовать с различными вариантами использования. В таком случае этот актер использует возможности сразу нескольких сервисов системы. Два прецедента, в одной моделируемой системе, также имеют возможность взаимодействовать друг с другом, но характер этой связи будет отличаться от отношения с актерами.

Язык UML предоставляет несколько стандартных видов отношений:

1. Отношение ассоциации (Association) - служит для обозначения уникальной роли актера при взаимодействии его с отдельным прецедентом. Графически на диаграмме вариантов использования это отношение изображается в виде сплошной линией между вариантом использования и актером.

2. Отношение включения (Include) - указывает на то, что поведение одного прецедента включается в некоторой точке в другой прецедент в качестве составного компонента. Особенности включения заключаются в том, что включаемый прецедент должен быть обязательным для дополняемого (включение должно быть безусловным, а дополняемый вариант использования без включения не сможет выполняться), т.е. это отношение задает очень сильную связь. Данное отношение является направленным бинарным отношением.

Графически данное отношение изображается в виде пунктирной линии со стрелкой, направленной от базового варианта использования к включаемому прецеденту. Данная линия при этом помечается надписью <<include>>.

3. Отношение расширения (Extend) - отражает возможное присоединение одного варианта использования к другому в некоторой точке (точке расширения). При этом подчеркивается то, что расширяющий вариант использования выполняется лишь при определенных условиях и не является обязательным для выполнения основного прецедента.

На диаграмме такой вид отношения изображается пунктирной линией со стрелкой, направленной к расширяемому прецеденту, в отдельном разделе которого может быть описана точка расширения, а условия расширения могут быть приведены в комментарии с ключевым словом Condition. Данная линия при этом помечается надписью <<extend>>.

4. Отношения обобщения (Generalization) – это ситуация, когда у двух или более актеров могут быть общие свойства, т. е. взаимодействовать с одним и тем же множеством вариантов использования одинаковым образом. Кроме того, прецедент, который наследуется от родителя, может переопределять его поведение.

Графически отношение обобщения изображается сплошной линией со стрелкой в форме треугольника, указывающей на родительский прецедент.

На рисунке 9 представлена Use Case диаграмма для проектируемого приложения AutoBook, которое позволит автоматизировать процесс учета книг в библиотеке.

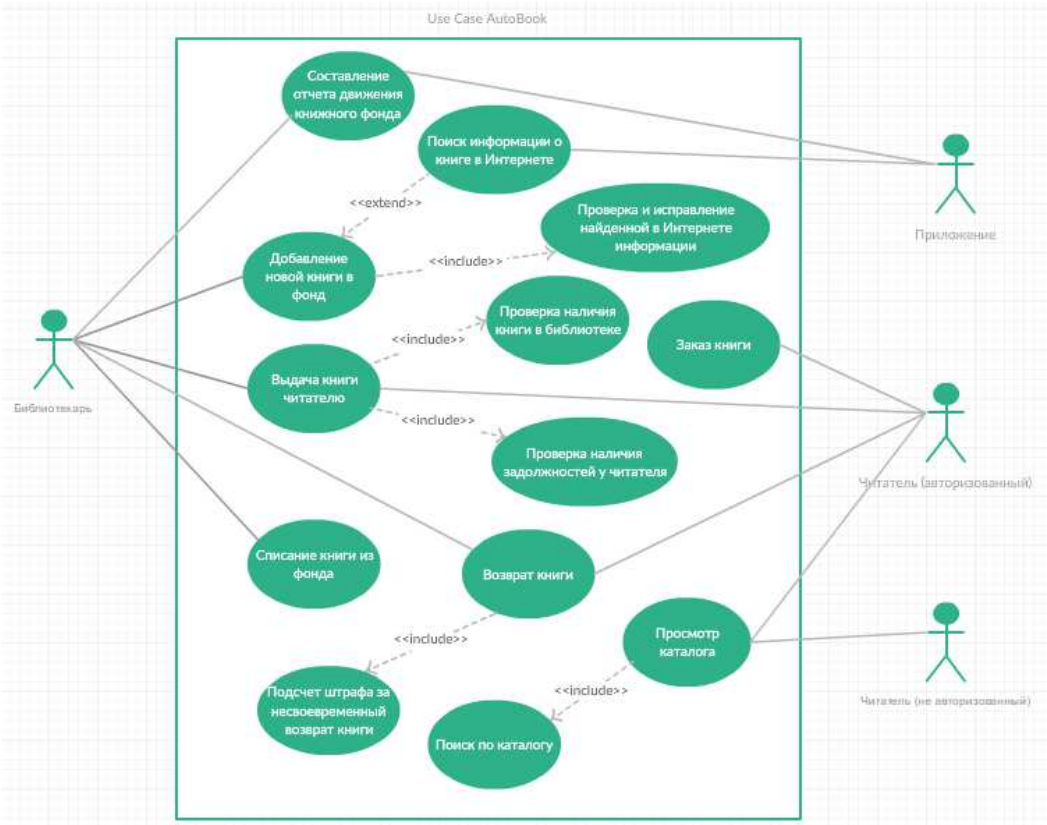


Рисунок 9 – Диаграмма Use Case AutoBook

На рисунке выше изображены 4 актера: библиотекарь, читатель авторизованный и читатель не авторизованный в системе, приложение.

Перечень основных функций:

1. добавление новой книги в фонд;
2. списание книги из фонда;
3. выдача / возврат книг читателем;
4. заказ книги на определенную дату;

5. поиск книг в фонде (по автору, названию, категории);
6. составление отчетов движения книжного фонда.

2.4 Диаграмма баз данных

Одним из главного компонента практически любой информационной системы является база данных. База данных – это организованный и четко структурированный, набор данных, объединенный в соответствии с некоторой выбранной моделью. Именно БД позволяет хранить данные пользователей и информацию о книгах, на ее основе строятся отчеты и рекомендации читателям.

В данной работе для проектирования базы данных была использована ER диаграмма (диаграмма «сущность – связь»). С её помощью были выделены ключевые сущности и установлены связи между ними. ER диаграммы позволяют применять легкие для восприятия наглядные графические обозначения для моделирования сущностей и их связей. Главным преимуществом метода является, то, что модель строится методом последовательного уточнения путем дополнения первоначальных диаграмм.

ER диаграмма состоит из атрибутов, сущностей и отношений.

Сущность - это объект предметной области, информацию о котором нужно хранить в базе данных. Графически на диаграмме базы данных изображается в виде прямоугольника, содержащим имя сущности в верхней части.

Атрибут сущности - это какая - либо характеристика или свойство, являющееся значимым для данной сущности. Название атрибута выражается существительными в единственном числе. Например, для сущности «Книга», атрибутами могут служить «Количество страниц», «Автор».

Графически на диаграмме атрибуты изображаются внутри прямоугольника, определяющего сущность. Каждый атрибут находится на отдельной строке.

Различают следующие виды атрибутов:

1. Идентифицирующие и описательные атрибуты. Идентифицирующими атрибутами могут быть только те, кто имеют уникальное значение для сущностей данного типа и потенциально могут быть ключами. С их помощью можно однозначно распознавать экземпляры сущности. Первичный ключ выбирается из их числа. В качестве первичного ключа чаще всего используется тот потенциальный ключ, по которому большее количество раз происходит обращение к экземплярам сущности. Остальные атрибуты являются описательными.

2. Простые и составные атрибуты. Простой атрибут имеет в своем составе один компонент, значение его неделимо. Составной атрибут состоит из комбинации нескольких компонентов, например, ФИО. Стоит ли использовать составной атрибут или лучше разбить его на компоненты, это всегда частный случай, который зависит от характера его обработки приложением и формата его пользовательского представления.

3. Однозначные и многозначные атрибуты. Могут иметь соответственно одно или множество значений для каждого экземпляра сущности.

4. Основные и производные атрибуты. Значение основного атрибута не зависит от других атрибутов. Значение производного атрибута вычисляется на основе значений других атрибутов. Например, дата возврата книги вычисляется на основе даты выдачи.

Связь - это некоторая ассоциация между двумя сущностями. Одна сущность может быть связана с другой сущностью или сама с собою.

Графически связь изображается линией, на концах которой подписывается тип связи (1 или ∞).

Типы связи:

1. Один-к-одному означает, что один экземпляр одной сущности (левой) связан с одним экземпляром другой сущности (правой). На практике связь один-к-одному используется не часто. Например, она нужна, когда требуется разделить данные одной таблицы на несколько с целью безопасности.

2. Связь типа многое-ко-многим используется, когда нужно чтобы каждый экземпляр одной сущности мог быть связан с несколькими экземплярами другой сущности, и также в другую сторону. Организовывается это посредством связывающей таблицы, где данный тип заменяется двумя связями типа один-ко-многим.

3. Связь типа один-ко-многим используется, когда нужно, чтобы один экземпляр одной сущности (левой) был связан с несколькими экземплярами другой сущности (правой). Этот тип связи используется наиболее часто. Левая сущность со стороны 1 называется родительской, а правая со стороны ∞ - именуется дочерней.

На рисунке 10 показана диаграмма базы данных для приложения AutoBook.

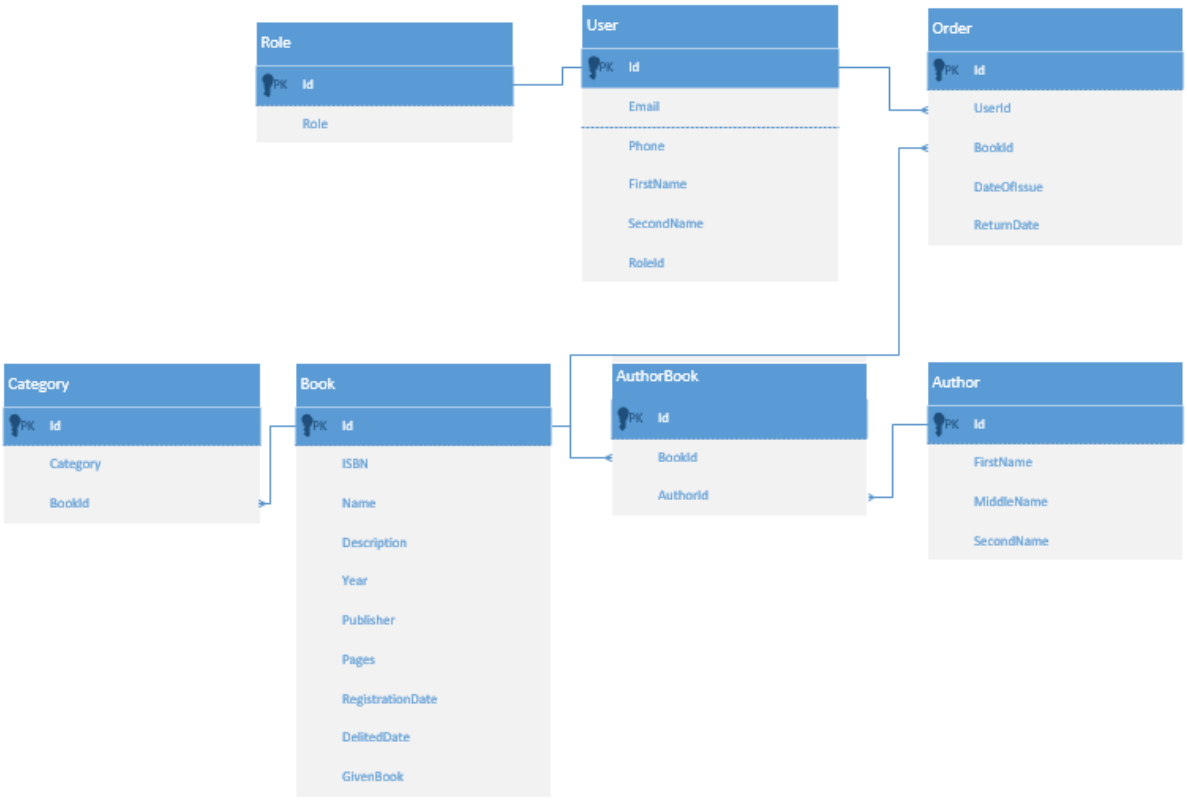


Рисунок 10 – ER диаграмма AutoBook

На рисунке выше изображены 7 основных сущностей, используемых в приложении и связи между ними:

- Book – вся информацию о конкретных экземплярах книг;
- Category – названия категорий и жанров литературы;
- AuthorBook – связующая таблица для организации связи многие-ко-многим между таблицами Author и Book;
- Author – авторы произведений;
- User – данные пользователей приложения;
- Role – роли пользователей в приложении (администратор, библиотекарь и читатель);
- Order – заказы книг читателями.

3 Программная реализация приложения

На рисунке 11 представлена структура разрабатываемого приложения. Оно состоит из 3 проектов, что обеспечивает ему слабую связываемость между разными уровнями. DataLayer отвечает за хранение всех данных приложения, BusinessLayer обрабатывает бизнес логику, а AutoBook занимается отображением представлений для пользователей.

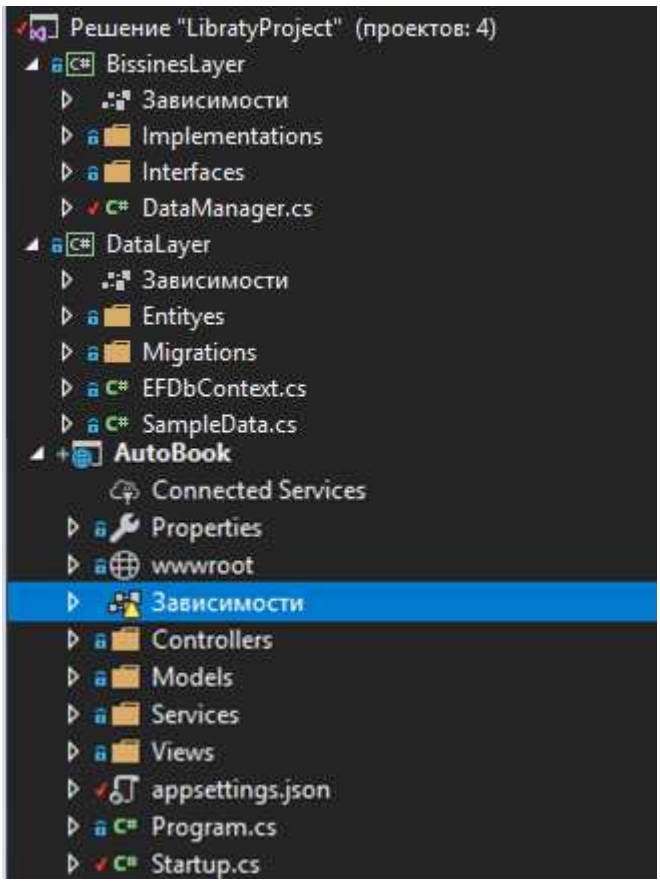


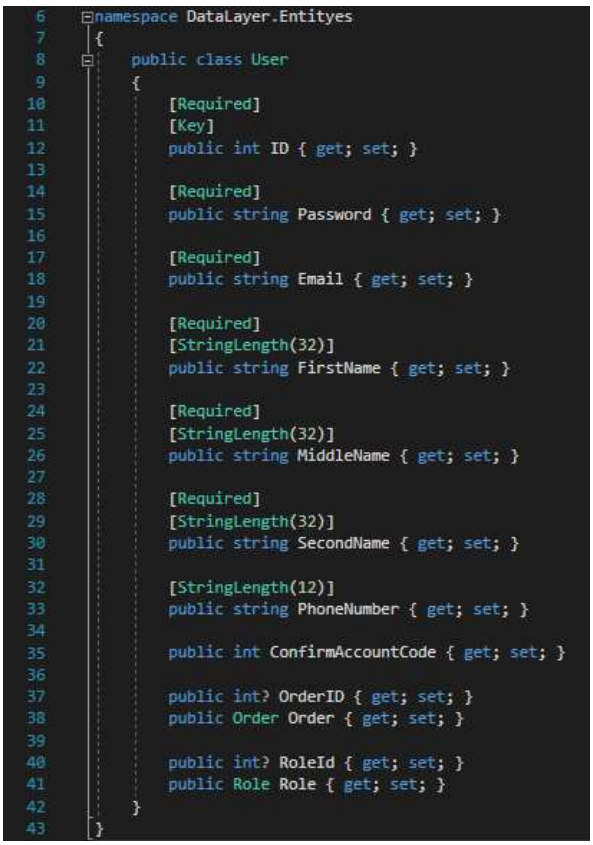
Рисунок 11 – Структура проекта

3.1 Разработка базы данных

В приложении используется реляционная база данных Microsoft SQL, а также используется фреймворк Entity Framework Core. По умолчанию в проекте библиотеки Entity Framework отсутствуют, и их необходимо добавить с помощью диспетчера пакетов Nuget:

- Microsoft.EntityFrameworkCore;
- Microsoft.EntityFrameworkCore.SqlServer;
- Microsoft.EntityFrameworkCore.Tools.

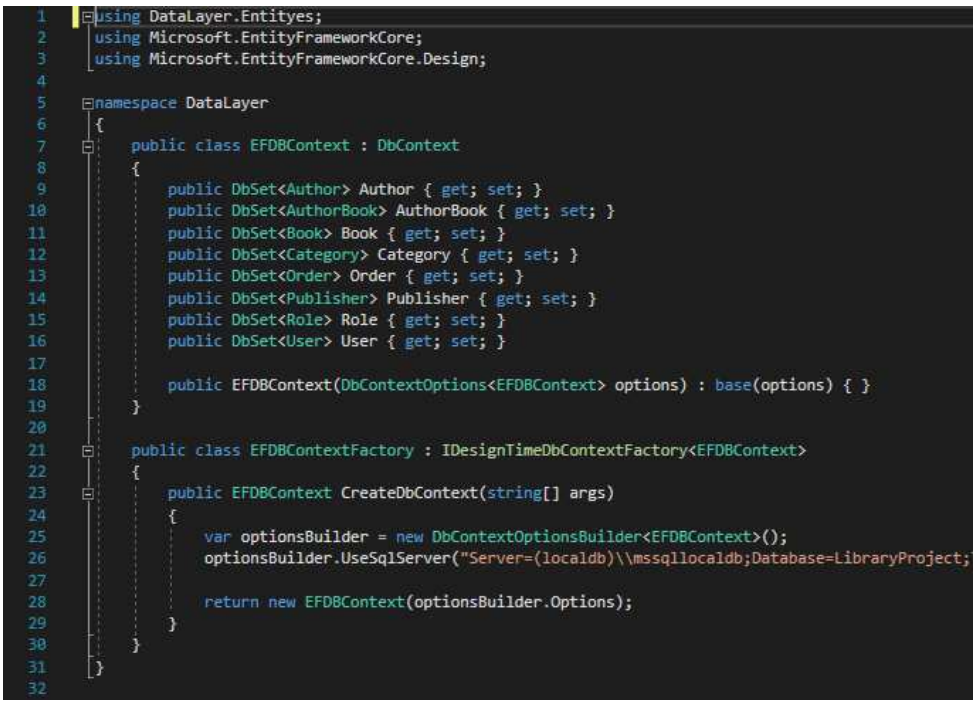
В слое DataLayer имеются две папки Entities и Migrations. Entities хранит объекты создаваемые в будущем в базе данных. На рисунке представлена один из таких объектов. В классе сущности User перечисляются атрибуты, которые будут храниться в базе данных.



```
namespace DataLayer.Entityes
{
    public class User
    {
        [Required]
        [Key]
        public int ID { get; set; }

        [Required]
        public string Password { get; set; }

        [Required]
        public string Email { get; set; }

        [Required]
        [StringLength(32)]
        public string FirstName { get; set; }

        [Required]
        [StringLength(32)]
        public string MiddleName { get; set; }

        [Required]
        [StringLength(32)]
        public string SecondName { get; set; }

        [StringLength(12)]
        public string PhoneNumber { get; set; }

        public int ConfirmAccountCode { get; set; }

        public int? OrderID { get; set; }
        public Order Order { get; set; }

        public int? RoleId { get; set; }
        public Role Role { get; set; }
    }
}
```

Рисунок 12 – Класс сущности User

Для взаимодействия с базой данных через Entity Framework нужен контекст данных - класс, унаследованный от класса DbContext. Поэтому в папку DataLayer имеется класс, EFDbContext. Контекст соответствует сеансу взаимодействия с базой данных и позволяет запрашивать и сохранять экземпляры классов сущностей.



```
using DataLayer.Entityes;
using Microsoft.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore.Design;

namespace DataLayer
{
    public class EFDBContext : DbContext
    {
        public DbSet<Author> Author { get; set; }
        public DbSet<AuthorBook> AuthorBook { get; set; }
        public DbSet<Book> Book { get; set; }
        public DbSet<Category> Category { get; set; }
        public DbSet<Order> Order { get; set; }
        public DbSet<Publisher> Publisher { get; set; }
        public DbSet<Role> Role { get; set; }
        public DbSet<User> User { get; set; }

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

    public class EFDBContextFactory : IDesignTimeDbContextFactory<EFDBContext>
    {
        public EFDBContext CreateDbContext(string[] args)
        {
            var optionsBuilder = new DbContextOptionsBuilder<EFDBContext>();
            optionsBuilder.UseSqlServer("Server=(localdb)\\mssqllocaldb;Database=LibraryProject;

            return new EFDBContext(optionsBuilder.Options);
        }
    }
}
```

Рисунок 13 – Контекст базы данных

Добавление контекста данных в виде сервиса в файле Startup.cs позволит затем получать его в конструкторе репозиториев в BusinessLayer через механизм внедрения зависимостей.

При разработке проекта модель данных может претерпевать изменения. Например, необходимо внести в нее новые свойства. Но при этом уже имеется существующая база данных, в которой есть какие-то данные. И чтобы без потерь обновить базу данных ASP.NET Core предлагает нам такой механизм как миграции. В папке Migrations, содержатся файлы миграций, созданные автоматически.

При создании миграции в проект добавляются три файла:

1. XXXXXXXXXXXXX_InitialCreate.cs - главный файл миграций. Содержит операции, необходимые для ее применения;
2. XXXXXXXXXXXXXX_InitialCreate.Designer.cs — файл метаданных миграций. Содержит сведения, используемые EF;
3. MyContextModelSnapshot.cs - моментальный снимок текущей модели. Используется для определения появившихся изменений при добавлении следующей миграции.

Метка времени в именах файлов позволяет расположить их в хронологическом порядке, чтобы вы могли отслеживать ход изменений.

В файле SampleData база данных заполняется базовыми данными, необходимыми для работы. Такими как названия ролей пользователей.

3.2 Разработка бизнес логики

Слой BusinessLayer содержит всю бизнес-логику, все необходимые вычисления, получает объекты из уровня доступа к данным и передает их на уровень представления, либо, наоборот, получает данные с уровня представления и передает их на уровень данных.

ASP.NET Core разработан специально для поддержки и использования инъекции зависимостей. Таким образом в папке Interfaces, мы создаем интерфейсы репозиториев для сущности, для того, чтобы мы могли разрабатывать слабо связанные приложения. На рисунке представлен фрагмент кода для интерфейса IUserRepository.

```
1   using DataLayer.Entityes;
2
3   namespace BissinesLayer.Interfaces
4   {
5       public interface IUserRepository
6       {
7           User GetUserById(int userId);
8           User GetUserByEmail(string email);
9           void CreateUser(User user);
10          User UpdateUser(User user);
11          User CheckRegisteredEmail(User user);
12          string GetHashPasswordSHA512(string input);
13      }
14  }
```

Рисунок 14 – Интерфейс IUserRepository

В папку Implementations хранятся классы, наследуемые от этих интерфейсов и позволяющие выполнять операций CRUD над сущностями. Этот репозиторий содержит параметризованный конструктор с контекстом в качестве параметра. При создании экземпляра репозитория мы можем работать с сущностями, хранящимися в БД, через контекст. Ниже приведен фрагмент кода для класса EFUserRepository.

```
namespace BissinesLayer.Implementations
{
    public class EFUserRepository : IUserRepository
    {
        private EFDBContext context;

        public EFUserRepository(EFDBContext context)
        {
            this.context = context;
        }

        public User CheckRegisteredEmail(User user)
        {
            return context.Set<User>().FirstOrDefault(u => u.Email == user.Email);
        }

        public void CreateUser(User user)
        {
            if (CheckRegisteredEmail(user) == null)
            {
                context.Set<User>().Add(user);
                context.SaveChanges();
            }
        }

        public User GetUserById(int userId)
        {
            return context.Set<User>().FirstOrDefault(x => x.ID == userId);
        }

        public User GetUserByEmail(string email)
        {
            return context.Set<User>().FirstOrDefault(x => x.Email == email);
        }
```

Рисунок 15 – Фрагмент EFUserRepository

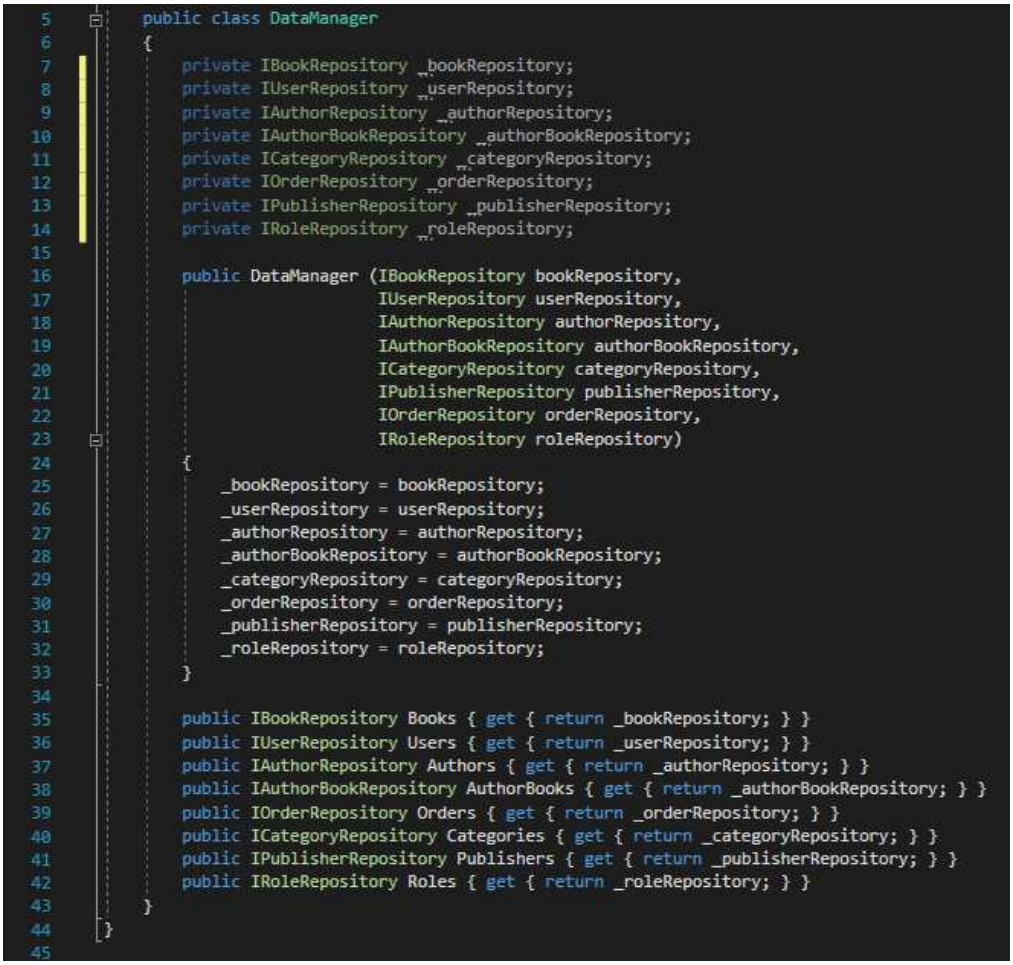
Для работы dependency injection необходимо добавить в файл Startup.cs строки, показанные на рисунке 16.

```
services.AddTransient<IBookRepository, EFBookRepository>();
services.AddTransient<IUserRepository, EFUserRepository>();
services.AddTransient<IAuthorRepository, EFAuthorRepository>();
services.AddTransient<IAuthorBookRepository, EFAuthorBookRepository>();
services.AddTransient<IOrderRepository, EFOrderRepository>();
services.AddTransient<ICategoryRepository, EFCategoryRepository>();
services.AddTransient<IPublisherRepository, EFPublisherRepository>();
services.AddTransient<IRoleRepository, EFRoleRepository>();
```

Рисунок 16 – Фрагмент файла Startup.cs

Метод AddTransient создает transient-объекты. Такие объекты создаются при каждом обращении к ним. В данном методе каждый интерфейс связывается с его реализацией.

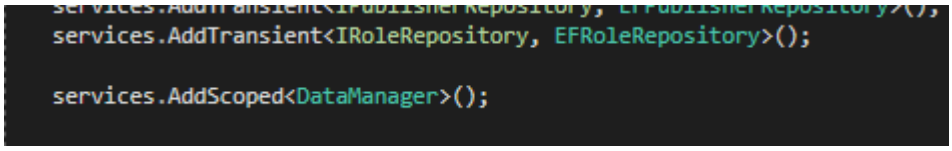
Для того чтобы не прописывать каждый репозиторий в контроллерах, создана специальная прослойка, названная DataManager. Создается один экземпляр DataManager, а у него в качестве полей ссылки на репозитории.



```
public class DataManager
{
    private IBookRepository _bookRepository;
    private IUserRepository _userRepository;
    private IAuthorRepository _authorRepository;
    private IAuthorBookRepository _authorBookRepository;
    private ICategoryRepository _categoryRepository;
    private IOrderRepository _orderRepository;
    private IPublisherRepository _publisherRepository;
    private IRoleRepository _roleRepository;

    public DataManager (IBookRepository bookRepository,
                        IUserRepository userRepository,
                        IAuthorRepository authorRepository,
                        IAuthorBookRepository authorBookRepository,
                        ICategoryRepository categoryRepository,
                        IPublisherRepository publisherRepository,
                        IOrderRepository orderRepository,
                        IRoleRepository roleRepository)
    {
        _bookRepository = bookRepository;
        _userRepository = userRepository;
        _authorRepository = authorRepository;
        _authorBookRepository = authorBookRepository;
        _categoryRepository = categoryRepository;
        _orderRepository = orderRepository;
        _publisherRepository = publisherRepository;
        _roleRepository = roleRepository;
    }

    public IBookRepository Books { get { return _bookRepository; } }
    public IUserRepository Users { get { return _userRepository; } }
    public IAuthorRepository Authors { get { return _authorRepository; } }
    public IAuthorBookRepository AuthorBooks { get { return _authorBookRepository; } }
    public IOrderRepository Orders { get { return _orderRepository; } }
    public ICategoryRepository Categories { get { return _categoryRepository; } }
    public IPublisherRepository Publishers { get { return _publisherRepository; } }
    public IRoleRepository Roles { get { return _roleRepository; } }
}
}
```

Рисунок 17 – Класс DataManager

В файле Startup.cs с помощью метода AddScoped создает один экземпляр объекта для всего запроса.



```
services.AddTransient<IRoleRepository, EFRoleRepository>();

services.AddScoped<DataManager>();
```

Рисунок 18 – Фрагмент файла Startup.cs

3.3 Разработка ViewModels

Нередко появляется необходимость взять только некоторые поля сущности из базы данных, не запрашивая ее целиком или надо передать в представление объекты сразу двух моделей. В этом случае используются модели представления.

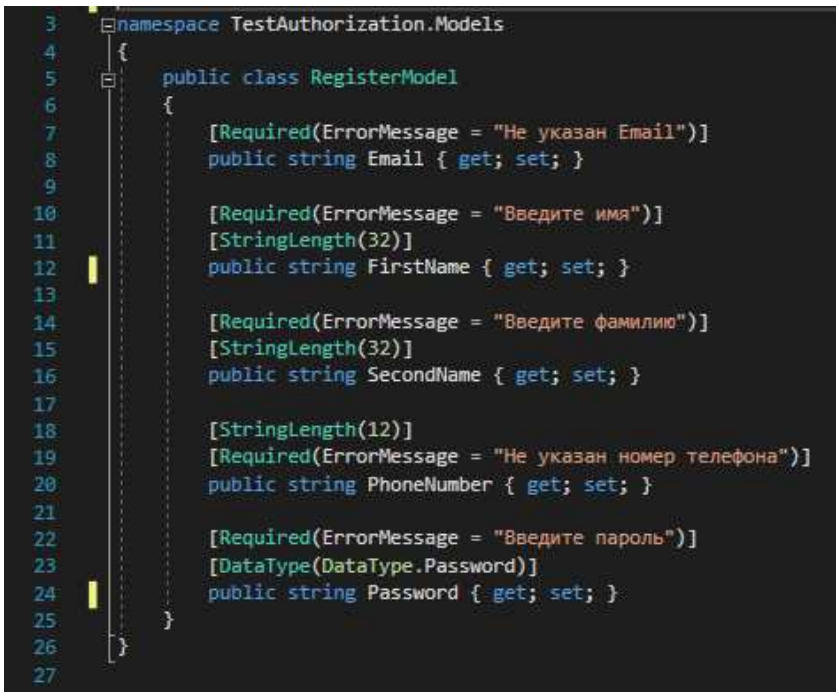


Рисунок 19 – Класс RegisterModel

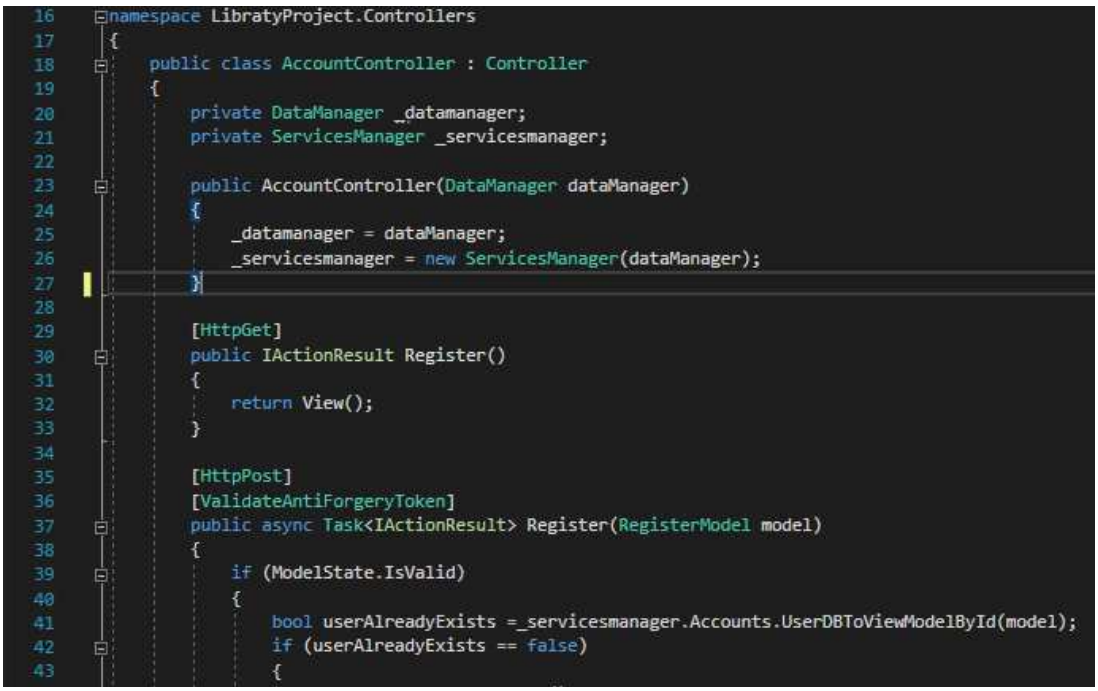
На рисунке 19 представлен пример модели для прохождения регистрации. Здесь представлены только те поля, которые должен заполнять пользователь. Поля ConfirmAccountCode нет, т.к. оно нужно только для работы приложения.

Также в данном случае имеется механизм валидации, который используется в MVC по умолчанию. Используются атрибуты валидации при объявлении модели. Для каждого свойства атрибут указан Required, благодаря чему фреймворк понимает, что данное свойство обязательно должно содержать некоторое значение. В случае если его не будет, будет выведена сообщение с ошибкой.

3.4 Разработка контроллеров

Контроллер является центральным звеном в архитектуре ASP.NET Core MVC. При получении запроса система маршрутизации выбирает для его обработки подходящий контроллер и передает ему данные запроса. Контроллер обрабатывает эти данные и посылает обратно результат обработки.

В ASP.NET Core контроллер представляет обычный класс на языке C#, который наследуется от абстрактного базового класса Controller.



```
namespace LibratyProject.Controllers
{
    public class AccountController : Controller
    {
        private DataManager _datamanager;
        private ServicesManager _servicesmanager;

        public AccountController(DataManager dataManager)
        {
            _datamanager = dataManager;
            _servicesmanager = new ServicesManager(dataManager);
        }

        [HttpGet]
        public IActionResult Register()
        {
            return View();
        }

        [HttpPost]
        [ValidateAntiForgeryToken]
        public async Task<IActionResult> Register(RegisterModel model)
        {
            if (ModelState.IsValid)
            {
                bool userAlreadyExists =_servicesmanager.Accounts.UserDBToViewModelById(model);
                if (userAlreadyExists == false)
                {
```

Рисунок 20 – Account контроллер

Для того, чтобы обратиться контроллеру из веб-браузера, нужно в адресной строке набрать адрес_сайта/Имя_контроллера/Действие_контроллера. Например, для прохождения регистрации этот путь будет выглядеть следующим образом: адрес_сайта/Account/Register.

3.5 Разработка представлений

Представления формируют внешний вид приложения. В ASP.NET Core - это файлы с расширением cshtml, которые содержат код пользовательского интерфейса в основном на языке html, а также конструкции Razor - специального движка представлений, который позволяет включать конструкции C# кода в язык разметки.

Все представления приложения хранятся в одной папке под названием Views. В этой папке уже имеется некоторая структура. Для каждого контроллера в проекте создается подкаталог, который называется по имени контроллера и который хранит представления, используемые методами данного контроллера.

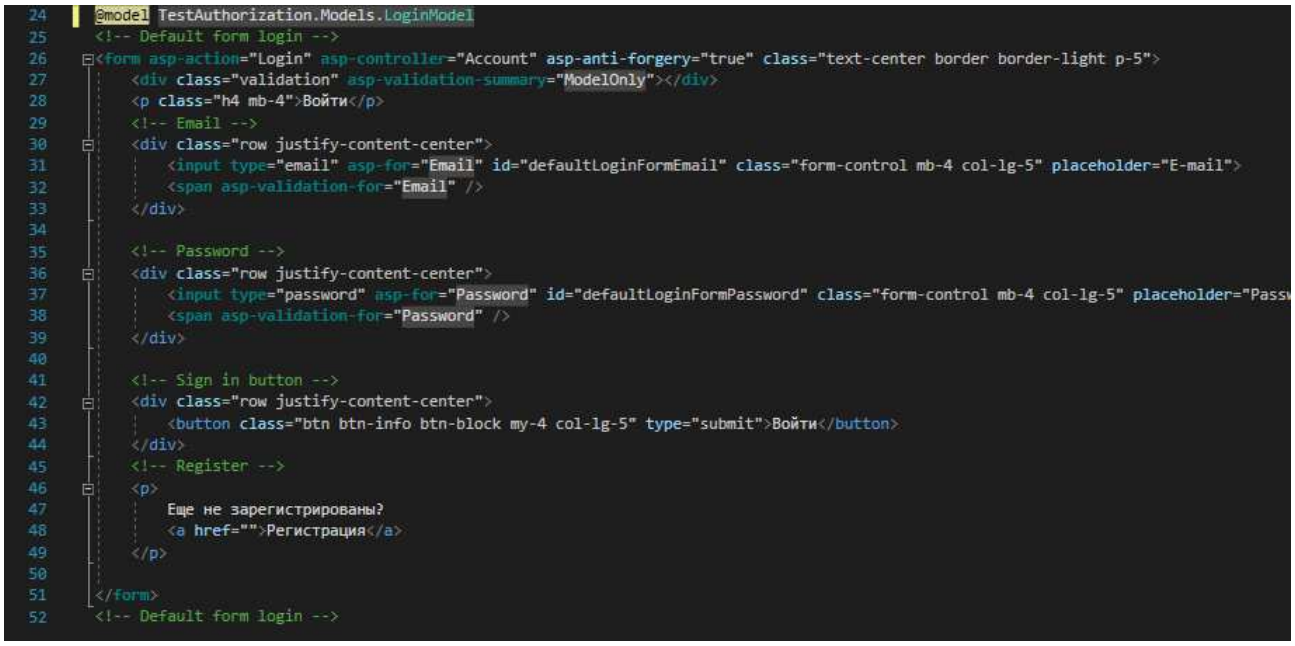
Так, например, для контроллера AccountController в папке Views есть подкаталог Account с представлениями для его методов.

Также здесь находится папка Shared, которая хранит общие представления для всех контроллеров. По умолчанию это файлы _Layout.cshtml (используется в качестве мастер-страницы), Error.cshtml (используются для отображения ошибок) и _ValidationScripsPartial.cshtml (частичное представление, которое подключает скрипты валидации формы).

В корне каталога Views можно найти два файла _ViewImports.cshtml и _ViewStart.cshtml. Эти файлы содержат код, который автоматически добавляется ко всем представлениям. _ViewImports.cshtml устанавливает

некоторые общие для всех представлений пространства имен, а _ViewStart.cshtml устанавливает общую мастер-страницу.

На рисунке 21 представлено одно из представлений приложения.



```
@model TestAuthorization.Models.LoginModel
<!-- Default form login -->
<form asp-action="Login" asp-controller="Account" asp-anti-forgery="true" class="text-center border border-light p-5">
    <div class="validation" asp-validation-summary="ModelOnly"></div>
    <p class="h4 mb-4">Войти</p>
    <!-- Email -->
    <div class="row justify-content-center">
        <input type="email" asp-for="Email" id="defaultLoginFormEmail" class="form-control mb-4 col-lg-5" placeholder="E-mail">
        <span asp-validation-for="Email" />
    </div>

    <!-- Password -->
    <div class="row justify-content-center">
        <input type="password" asp-for="Password" id="defaultLoginFormPassword" class="form-control mb-4 col-lg-5" placeholder="Passw
        <span asp-validation-for="Password" />
    </div>

    <!-- Sign in button -->
    <div class="row justify-content-center">
        <button class="btn btn-info btn-block my-4 col-lg-5" type="submit">Войти</button>
    </div>
    <!-- Register -->
    <p>
        Еще не зарегистрированы?
        <a href="">Регистрация</a>
    </p>

</form>
<!-- Default form login -->
```

Рисунок 21 – Представление для авторизации

Для передачи данных в контроллер используются tag-хелперы asp-for. При нажатии кнопки регистрации данные формы будут переданы в контроллер с помощью LoginModel и там уже обработаны. За валидацию отвечают tag-хелперы asp-validation-for.

4 Принципы работы с приложением

В данном проекте существует 3 роли:
- администратор;
- библиотекарь;
- читатель.

Администратор имеет самые большие полномочия в системе, может:
- создавать, удалять и редактировать учётные записи библиотекарей;
- составлять отчеты о движении книг в фонде или во всех фондах.

Библиотекарь - главное лицо в рамках отдельно взятой школьной библиотеки, может:
- авторизоваться на сайте;
- контролирует процесс взаимодействия читателей с библиотекой;
- обрабатывать запросы читателей;
- выдавать книги;
- редактирует учетные записи читателей;
- составлять отчеты о движении книг в фонде.

Читатель может:
- авторизоваться на сайте;
- самостоятельно заказывать, получать и читает книги.

4.1 Инструкция для библиотекаря

4.1.1 Вход библиотекаря в систему

Для начала работы с приложением в первую очередь библиотекарю необходимо авторизоваться. Для этого в верхней части сайта нужно нажать на ссылку «Войти». На открывшейся странице ввести в поля формы логин и пароль библиотекаря, полученные от администратора.

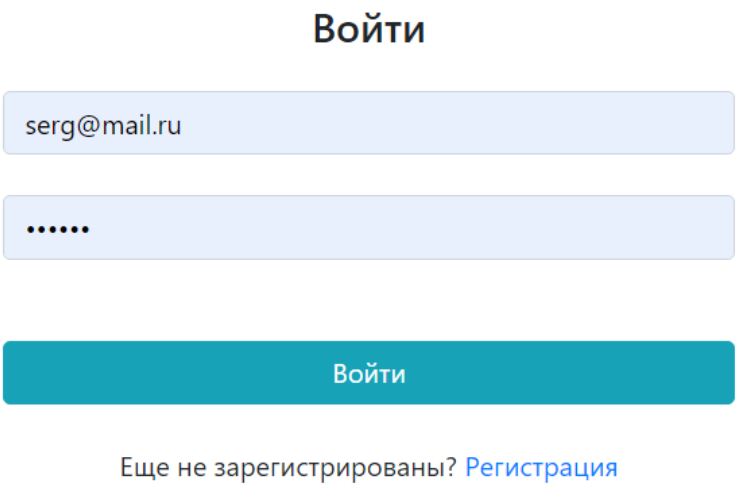


Рисунок 22 – Окно авторизации в приложении

После успешного прохождения авторизации в правом верхнем углу сайта появится имя и фамилия библиотекаря.

4.1.2 Выдача книг читателям

Сценарий 1: «самообслуживание» - самостоятельное получение книг читателями

«Самообслуживание» является одним из основных сценариев работы с системой, позволяющий читателям - быстро приступать к чтению.

Суть этого подхода заключается в том, что читатель при личном посещении библиотеки самостоятельно выбирает любые книги, соответствующие его возрасту.

Библиотекарь настраивает количество книг, которое читатель может иметь на руках в один момент времени. И заполняет в приложении информацию о выдаче.

Сценарий 2: «запрос-выдача» - читатели запрашивают книги, а библиотекарь вручную формируют заказ и выдают его при посещении.

«Запрос-выдача» является наиболее удобным сценарием как для читателей, так и для библиотекарей. Однако он вынуждает читателя ожидать некоторое время перед началом чтения.

Для того чтобы воспользоваться этим подходом, читатель должен выбрать понравившеюся ему книгу и нажать кнопку «Заказать книгу» во вкладке «Каталог» или во вкладке «Информация о книге», выбрать желаемую дату, когда будет удобно забрать заказ.

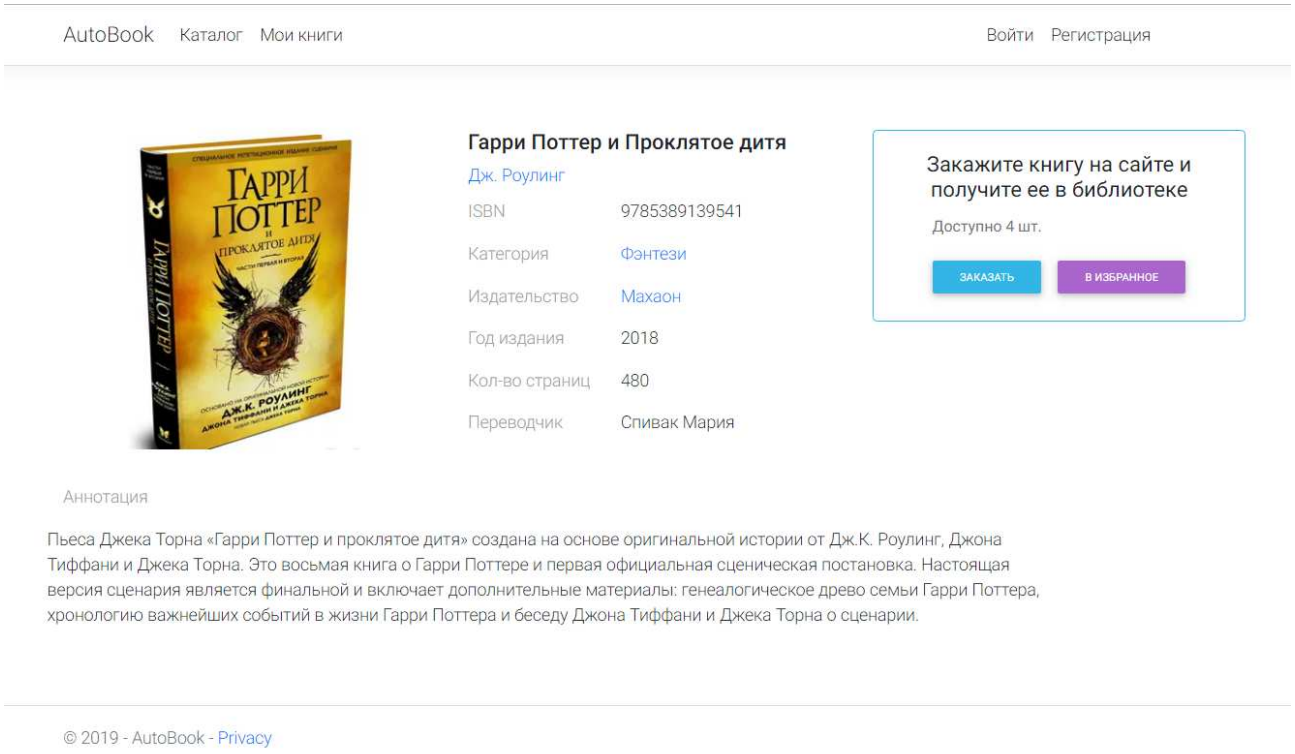


Рисунок 23 – Окно заказа книги

Библиотекарь должен периодически просматривать информацию о заказах книг в разделе «Заказанные» и при поступлении новых своевременно

их обрабатывать. Библиотекарь индивидуально по каждому из них принимает решение: выдать книгу или отказать в выдаче.

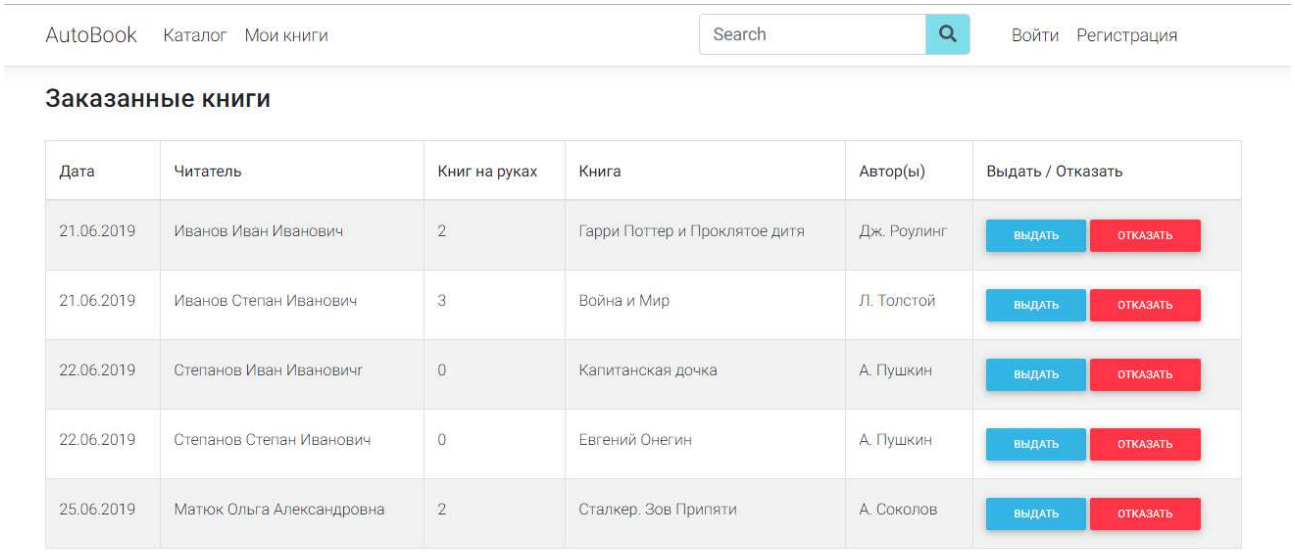


Рисунок 24 – Раздел «Заказанные»

Если библиотекарь подтверждает выдачу, то читателю отправляется письмо на email с информацией, что заказ будет готов к указанному дню и в приложении в разделе «Заказанные» ставится статус «Принято».

При отказе в выдаче читателю отправляется письмо на email, где библиотекарь прописывает причину отказа.

4.1.3 Составление отчетов и статистики

Приложение электронной библиотеки позволяет собирать и хранить большое количество статистических данных. Для последующего анализа имеется возможность составления удобных отчётов в формате Excel.

Общая статистика хранит информацию

- о количестве выданных книг;
- о количестве возвращенных книг;
- о количестве посетителей библиотеки;
- о количестве новых регистраций читателей.

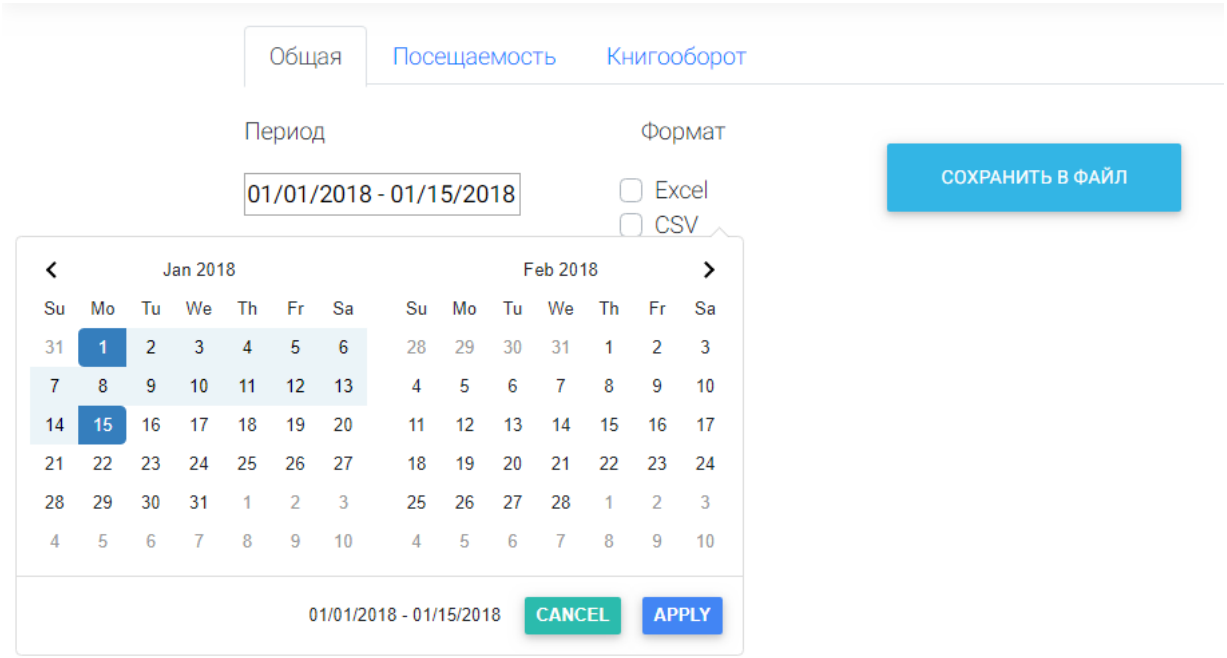


Рисунок 25 – Статистика общая

В статистике о посещаемости приведена информация о читателях, которые хотя бы раз воспользовавшихся сервисом в течение отчётного дня. Под посещением считается любое действие: запрос на книги, открытие сайта и др.

С помощью статистики книгообороту имеется возможность анализировать частоту выдачи определенных книг, а также отслеживать количество должников.

На основе этой информации на главной странице сайта будут выводится самые популярные книги по количеству прочтений за месяц.

На рисунке 26 представлен пример отчета по книгообороту за период с 04.07.2016 по 03.08, экспортированный в Excel документ.

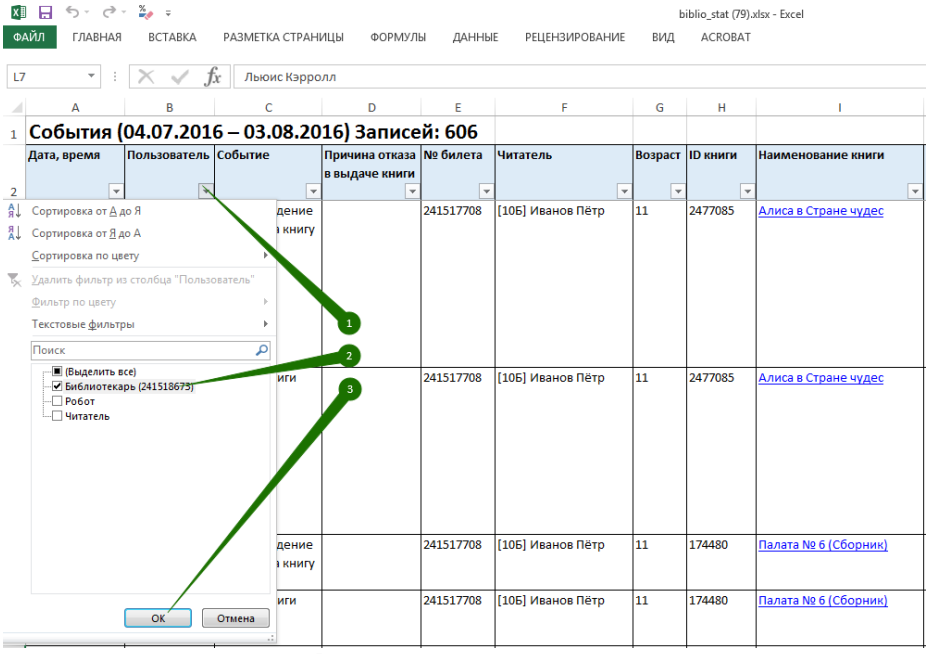


Рисунок 26 – Excel документ статистики по книгообороту

4.1.4 Просмотр библиотечного фонда

Все книги, которые добавлены в библиотеку в ручном режиме или импортированы из Excel файла находятся во вкладке «Каталог», ссылка на нее имеется на главной странице.

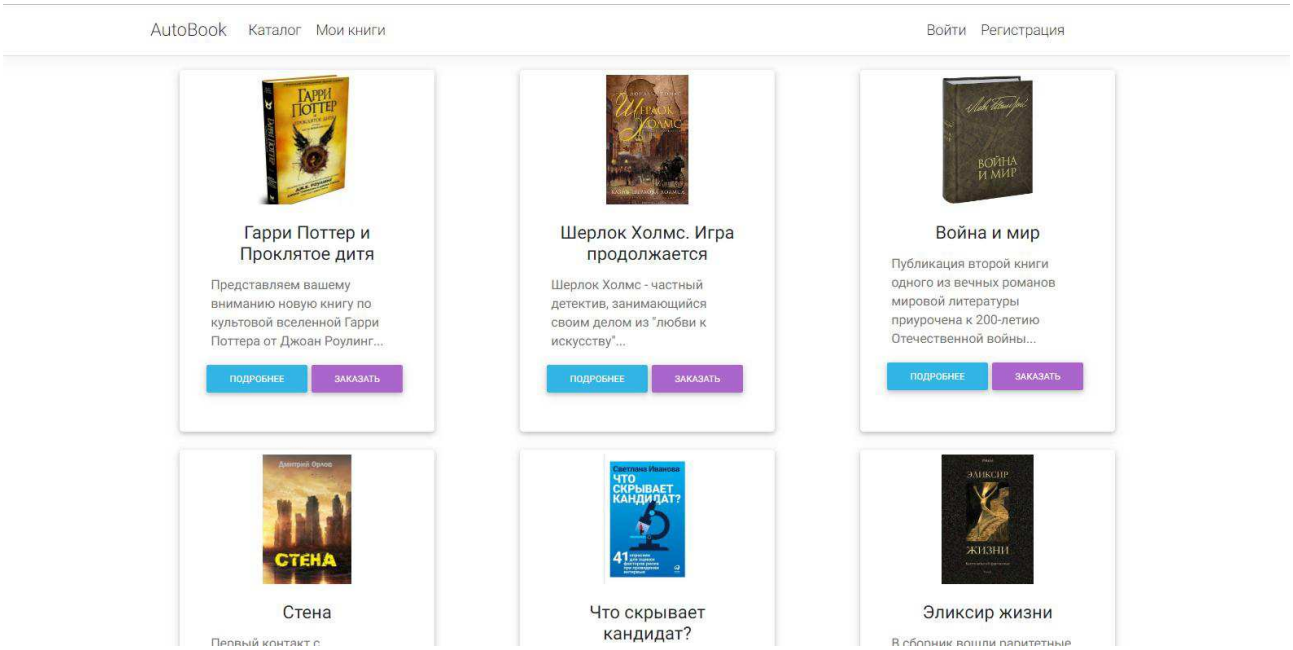


Рисунок 27 – Вкладка «Каталог»

К этому разделу имеют доступ как библиотекарь, так и читатели, что позволяет им незамедлительно найти популярные книги. Пользователи, не прошедшие авторизацию, также могут просматривать список книг из вашего фонда.

3.1.5 Редактирование фонда

У библиотекаря имеется служебный раздел для работы с библиотечным фондом где он может:

- просматривать списки выданных и свободных книг;
- выгружать в фонд книги в формате Excel(CSV);
- добавлять книги в фонд (поштучно);
- списывать книги из фонда.

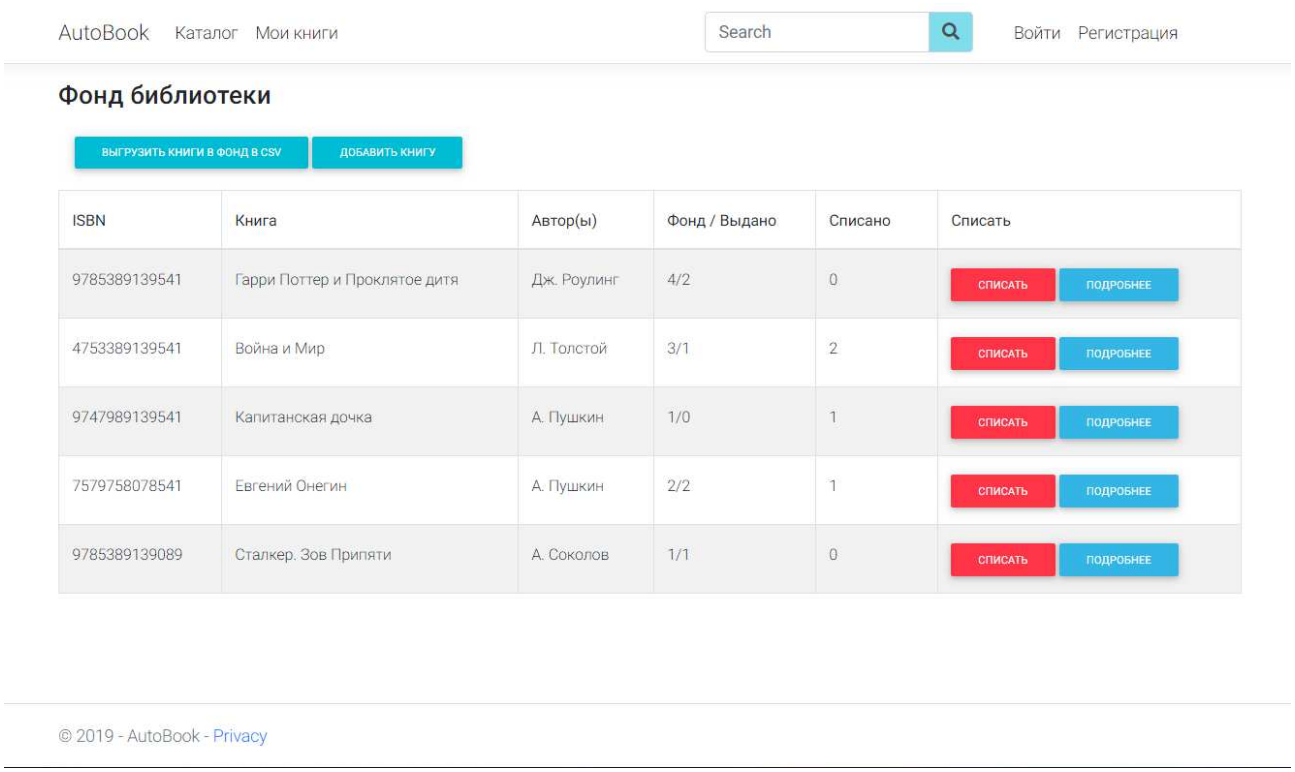


Рисунок 28 – Список выданных и свободных книг

Для того чтобы добавить новую книгу в фонд, библиотекарь может написать ISBN код в соответствующее поле и приложение произведет поиск по базе данных. Если имеется информация о ней, то она самостоятельно заполнит все необходимые поля, а библиотекарю останется проверить правильность этих данных и подкорректировать их, если это необходимо.

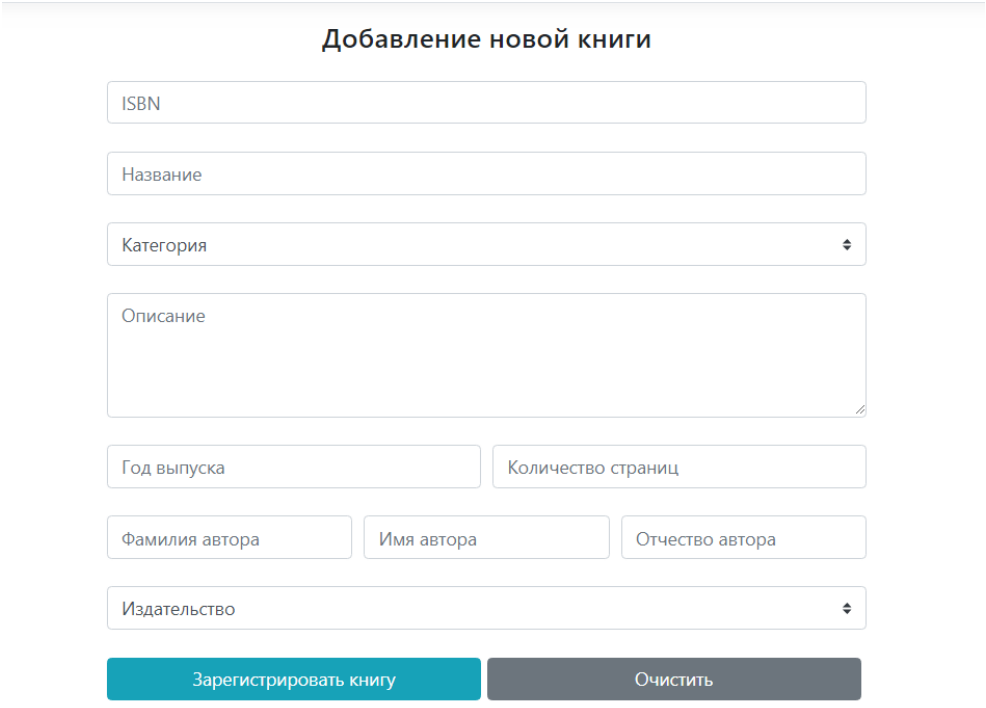


Рисунок 29 – Окно добавления новой книги в фонд

Для списания книги из фонда библиотекарь должен выбрать необходимую книгу в списке выданных и свободных книг и нажать кнопку «Списать». В открывшемся окне указать причину списания. Информация о списанных книгах все равно хранится в приложении в разделе «Списанное из фонда».

Рисунок 30 – Раздел «Списанное из фонда»

4.2 Инструкция для читателей

4.2.1 Регистрация читателей в системе

Для регистрации в приложении в верхней части сайта нужно нажать на ссылку «Регистрация». В открывшемся окне ввести:

- имя;
- фамилию;
- email;
- номер телефона;
- пароль.

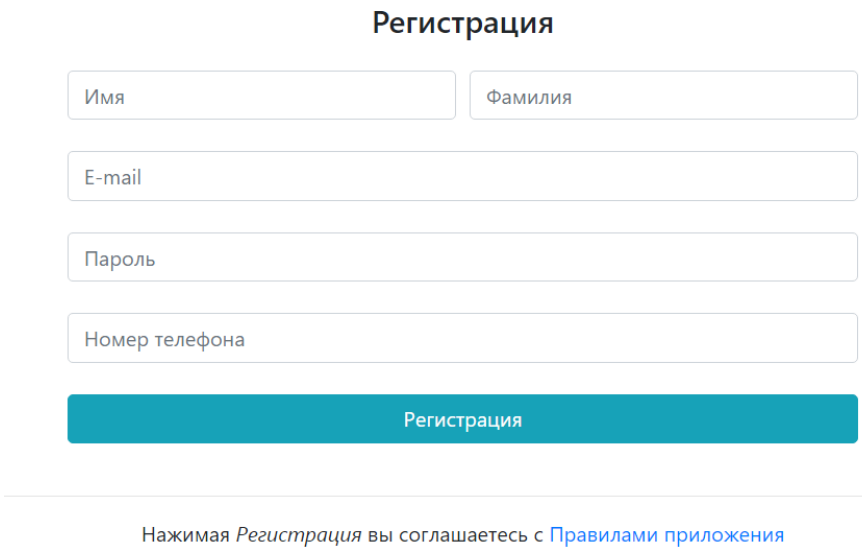


Рисунок 31 – Окно регистрации читателя в приложении

На электронную почту будет выслано письмо с кодом подтверждения, его необходимо ввести в соответствующее поле для завершения процесса регистрации.

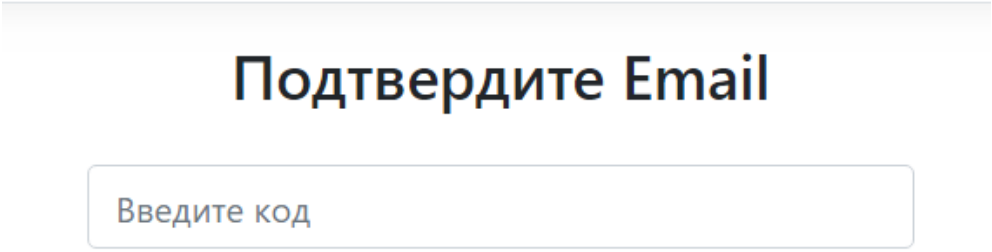


Рисунок 32 – Окно подтверждения Email

4.2.2 Авторизация читателей в системе

Используя логин и пароль, читатель может в любом месте, где есть интернет авторизоваться на сайте и приступить к чтению книг.

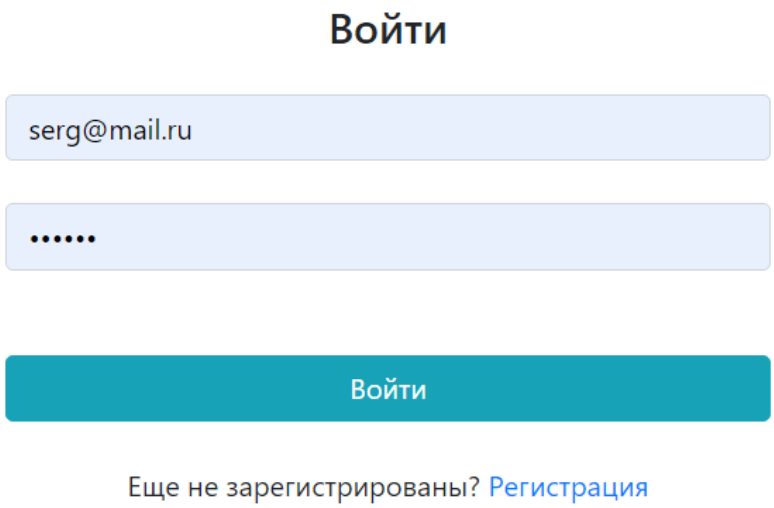


Рисунок 33 – Окно авторизации в приложении

4.2.3 Поиск и заказ книг

Для просмотра всех книг библиотечного фонда перейдите на вкладку «Каталог». Здесь вы можете выбрать любую понравившуюся книгу, просмотреть подробную информация о ней и заказать с помощью кнопки «Заказать».

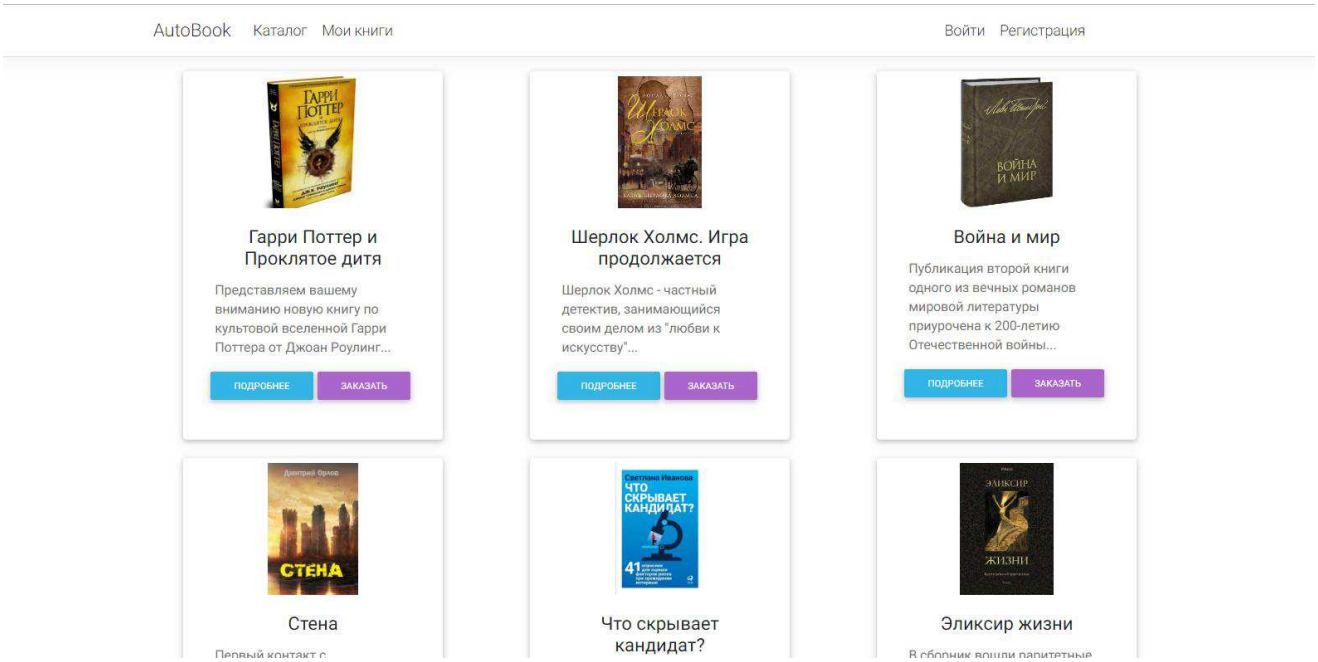


Рисунок 34 – Вкладка «Каталог»

В открывшемся окне необходимо выбрать дату, когда будет удобно забрать книги. В указанный день вам на почту придёт email уведомление с напоминаем, что нужно получить заказ.

Чтобы найти интересующую книгу, воспользуйтесь строкой поиска, находящейся в верхней части вкладки «Каталог». Поиск можно осуществить по автору или названию книги.

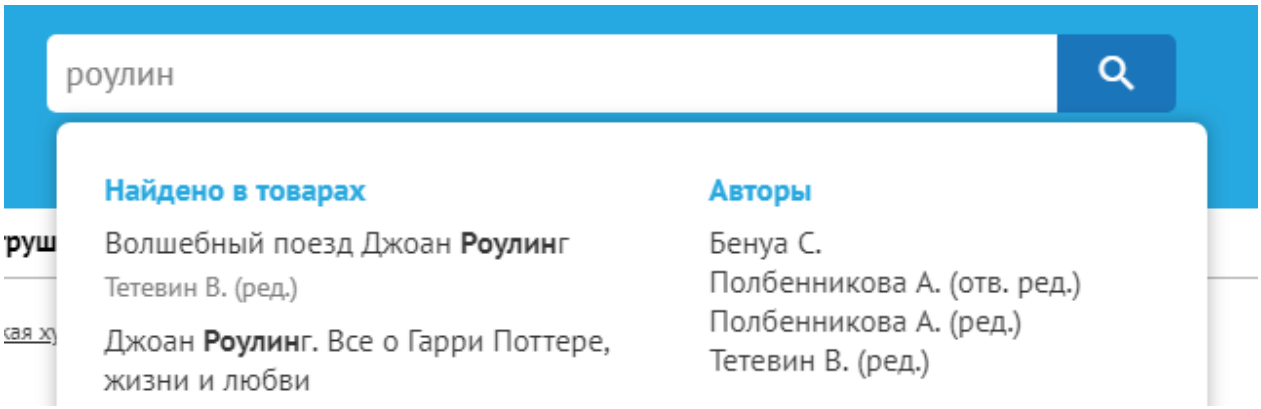


Рисунок 35 – Строка поиска книги

После одобрения библиотекарем вашего запроса и выдачи книги, вы получите уведомление на e-mail, а также эта книга появится в разделе «Мои книги». Все, когда - либо полученные читателем книги будут собираться в этом разделе.

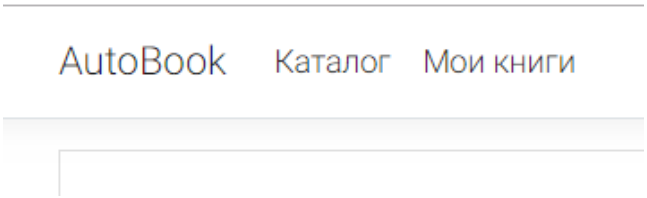


Рисунок 36 – Раздел «Мои книги»

Библиотекарь в некоторых случаях может отказать в выдаче книги. Вы получите email уведомление, с описанием причины отказа.

Когда вы закончите чтение книги, вам не нужно будет вернуть в библиотеку. Если вы забудете, приложение автоматически пришлет вам письмо за 3 дня до окончания срока пользования книгой.

ЗАКЛЮЧЕНИЕ

В результате выполнения бакалаврской работы разработано веб приложение, позволяющее автоматизировать процесс учета книг в библиотеке. Требования, поставленные перед началом работы, были выполнены.

На этапе проектирования был проведен общий анализ аналогов, существующих на рынке, таких как «Litres.ru» и «LibaBook». Построены Use Case диаграмма и диаграмма базы данных.

Были выявлены функциональные требования к ИС:

- добавление новых книг в фонд;
- списание книги из фонда;
- выдача и возврат книг читателем;
- возможность заказа книги читателем на определенную дату;
- поиск книг в фонде (по автору, названию, категории);
- составление отчетов движения книжного фонда.

Приложение было создана на языке C# с использованием фреймворка ASP NET Core MVC. Для хранения информации была использована база данных Microsoft SQL Server. Результатом бакалаврской работы стало веб приложение для библиотекарей и читателей.

СПИСОК СОКРАЩЕНИЙ

АБИС -автоматизированные библиотечно-информационные системы
ЭБС – электронно-библиотечная система
ИС – информационная система
ПО – программное обеспечение
БД – база данных
СУБД – система управления базой данных
ПК – персональный компьютер
API – application programming interface
MVC – model, view, controller
MS - microsoft
SQL - structured query language
LINQ - language-integrated query
EF – entity framework
ER – entity relationship
DI - dependency injection

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Фримен А. ASP.NET Core MVC 2 с примерами на C# для профессионалов 7-е изд. / А.Фримен – Киев: Диалектика, 2019. – 1008 с.: ил.
2. Рихтер Д. CLR via C#. Программирование на платформе Microsoft .NET Framework 4.5 на языке C#. 4-е изд. / Д.Рихтер – Санкт-Петербург: Питер, 2018. – 896 с.: ил.
3. Албахари Д., Албахари Б. C# 7.0. Карманный справочник – Санкт-Петербург: Диалектика, 2017 – 224 с.: ил.
4. Руководство по ASP.NET Core 2 [Электронный ресурс] / режим доступа: https://metanit.com/sharp/aspnet5/
5. Введение в ASP.NET Core [Электронный ресурс] / режим доступа: https://docs.microsoft.com/ru-ru/aspnet/core/
6. Руководство по Entity Framework Core [Электронный ресурс] / режим доступа: https://metanit.com/sharp/entityframeworkcore/
7. Соглашение по именованию / Методология / БЭМ [Электронный ресурс] / режим доступа: https://ru.bem.info/methodology/naming-convention/
8. Bootstrap 4 – официальная документация [Электронный ресурс] / режим доступа: https://bootstrap-4.ru
9. Руководства по веб-технологиям HTML, CSS документация [Электронный ресурс] / режим доступа: https://webref.ru/layout
10. ISBNdb API Documentation [Электронный ресурс] / режим доступа: https://isbndb.com/apidocs
11. Мезенцев К.Н. Автоматизированные информационные системы. – Москва: Академия, 2012. – 174 с.: ил.
12. Емельянова Н.З. Проектирование информационных систем. – Москва: Форум, 2010. – 432 с.: ил.
13. Гвоздева Т.В. Проектирование информационных систем. – Ростов-на-Дону: Феникс, 2010. – 512 с.: ил.
14. Руководство по CSS [Электронный ресурс]. / режим доступа: https://developer.mozilla.org/ru/docs/Web/CSS/Reference
15. СТО 4.2-07-2014 Система менеджмента качества. Общие требования к построению, изложению и оформлению документов учебной деятельности. – Введ. 9.01.2014. – Красноярск: ИПК СФУ, 2014. – 60 с.
16. Microsoft Docs [Электронный ресурс] / режим доступа: https://docs.microsoft.com/en-us/aspnet/core/security/authentication/?view=aspnetcore-2.1.

Федеральное государственное автономное
образовательное учреждение
высшего образования
«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»
Институт космических и информационных технологий
институт
Информационных систем
кафедра

УТВЕРЖДАЮ
Заведующий кафедрой ИС
_____ Дьячук П.П.
подпись фамилия, инициалы
« 21 » июля 2019 г.

БАКАЛАВРСКАЯ РАБОТА

09.03.02 Информационные системы и технологии

Автоматизация процесса учета книг в библиотеке

Руководитель _____ 21.06.19 канд. техн. наук, доцент каф. И.А. Легалов
подпись, дата ученая степень, должность инициалы, фамилия

Студент КИ15-13Б 031509269 _____ 21.06.19 С.А. Маркушин
номер группы номер зачетной книжки подпись, дата инициалы, фамилия

Нормоконтролер 21.06.19 _____ Ю.В. Шмагрис
подпись, дата инициалы, фамилия

Красноярск 2019