**МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ**

**УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ**

**ГОМЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ**

**УНИВЕРСИТЕТ ИМЕНИ П. О. СУХОГО**

Факультет автоматизированных и информационных систем

Кафедра «Информационные технологии»

Специальность 1-40 05 01-01 Информационные системы и технологии (в проектировании и производстве)

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовому проекту

по дисциплине «Разработка приложений баз данных для информационных систем»

**на тему: «Подписка на издания в почтовых отделениях города»**

Выполнил: студент гр. ИТП-31

Бондарев Е.Ю.  
Руководитель: доцент

Асенчик О. Д.

Дата проверки: ­­­­­­­ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_­­­­­­­\_­­­­­­­\_­­­­­­­\_­­­­­­­\_­­­­­­­\_­­­­­­­\_­­­­­­­\_­­­­­­­

Дата допуска к защите:­­­­­­­ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_­­­­­­­\_­­­­­­­\_­­­­­­­\_­­­­­­­\_­­­­­­­\_­­­­­­­\_­­­­­­­\_­­­­­­

Дата защиты: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_­­­­­­­\_­­­­­­­\_­­­­­­­\_­­­­­­­\_­­­­­­­\_­­­­­­­\_­­­­­­­\_

Оценка работы: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_­­­­­­­\_­­­­­­­\_­­­­­­­\_­­­­­­­\_­­­­­­­\_­­­­­­­\_­­­­­­­\_

Подписи членов комиссии

по защите курсового проекта: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Гомель 2023

**СОДЕРЖАНИЕ**

[Введение 5](#_Toc152518238)

[1 Логическая и физическая структура базы данных 7](#_Toc152518239)

[1.1 Информационно-логическая модель предметной области 7](#_Toc152518240)

[1.2 Физическая модель базы данных 10](#_Toc152518241)

[2 Аппаратное и программное обеспечение информационной системы 15](#_Toc152518242)

[2.1 Требования к системному и прикладному программному обеспечению на стороне сервера хранилища данных 15](#_Toc152518243)

[2.2 Требования к системному и прикладному программному обеспечению на стороне *web*-сервера 15](#_Toc152518244)

[2.3 Действия системного администратора при настройке и развертывании *web*-приложения 16](#_Toc152518245)

[3 Структура приложения 18](#_Toc152518246)

[3.1 Описание общей структуры приложения 18](#_Toc152518247)

[3.2 Описание компонента *domains* 18](#_Toc152518248)

[3.3 Описание компонента *repository* 21](#_Toc152518249)

[3.4 Описание компонента *service* 22](#_Toc152518250)

[3.5 Описание компонента *PostCity* 23](#_Toc152518251)

[4 Руководство пользователя 25](#_Toc152518252)

[4.1 Назначение, условие применения и функционал 25](#_Toc152518253)

[4.2 Графический интерфейс и его функционал 25](#_Toc152518254)

[5 Руководство программиста 31](#_Toc152518255)

[Заключение 32](#_Toc152518256)

[Список используемых источников 33](#_Toc152518257)

[Приложение А Листинг программы 34](#_Toc152518258)

[Приложение Б Чертеж структуры *web*-приложения 115](#_Toc152518260)

# ВВЕДЕНИЕ

В современном информационном обществе, где технологии активно влияют на наш повседневный образ жизни, вопросы организации эффективного распространения и получения информации становятся все более важными. Одним из ключевых аспектов этой проблематики является вопрос об удобстве получения печатных изданий, таких как газеты и журналы, через почтовые отделения. Актуальность данной темы обусловлена рядом обстоятельств, которые заслуживают внимания и исследования.

В первую очередь, следует отметить, что в современном мире число подписчиков на печатные издания неуклонно сокращается под воздействием цифровизации и развития онлайн-медиа. Вместе с тем, существует определенная категория читателей, предпочитающих традиционный формат и получение информации в виде бумажных изданий. Почтовые отделения могут стать эффективным каналом для доставки таких изданий, обеспечивая удобство и доступность для данной аудитории.

Дополнительной причиной актуальности темы является стремление издательств повысить эффективность распределения своих продуктов. Почтовые отделения предоставляют возможность централизованного и структурированного распределения изданий, что может способствовать оптимизации логистических процессов и снижению затрат на доставку.

Таким образом, работа посвящена актуальной проблеме организации подписки на печатные издания через почтовые отделения города, и представляет собой попытку исследования этой темы с целью выработки практических рекомендаций для улучшения процессов распределения и получения бумажных изданий.

Выбранная программная архитектура для разрабатываемой информационной системы (ИС) представляет собой *MVS* (*Model-View-Controller*), которая является широко используемой парадигмой для организации кода приложения. *MVS* разделяет приложение на три ключевые компоненты: *Model* (модель), *View* (представление) и *Controller* (контроллер), что способствует легкости поддержки и расширения приложения.

В контексте использования СУБД *MS SQL* (*Microsoft SQL Server*) для хранения данных, *MVS* позволяет ясно выделить модель данных (*Model*), представление данных (*View*) и логику управления данными (*Controller*). Модель данных отвечает за взаимодействие с базой данных, представление определяет, как данные будут представлены пользователю, а контроллер управляет логикой приложения, обрабатывая входные запросы и взаимодействуя с моделью и представлением.

Важной составляющей разрабатываемой информационной системы (ИС) является веб-интерфейс, предоставляющий пользователям возможность взаимодействия с системой. В данном контексте использование платформы *.NET* *Core* позволяет создать современный и удобный интерфейс, соответствующий требованиям современных веб-приложений.

*ASP.NET Core MVC* является выбранной технологией для реализации веб-интерфейса. Эта модель, основанная на принципах *Model-View-Controller* (*MVC*).

Для управления данными из базы данных *MS SQL* в модели данных (*Model*) используется *Entity Framework Core*. Этот инструмент обеспечивает высокоуровневый объектно-ориентированный доступ к данным, упрощая взаимодействие с базой данных и соответствуя принципам архитектуры *MVS*.

Для создания представления данных (*View*) веб-интерфейс использует технологии *Razor Pages*, что обеспечивает удобство генерации *HTML* на сервере. Это способствует легкости интеграции данных из контроллера в пользовательский интерфейс. Кроме того, применение *HTML*, *CSS* и *JavaScript* обеспечивает создание современных и отзывчивых веб-приложений, соответствующих ожиданиям пользователей.

Для обеспечения эстетичного и адаптивного дизайна веб-интерфейса используются фреймворки, такие как *Bootstrap*. Это облегчает создание интерфейса, который привлекателен и хорошо адаптирован к различным устройствам. Таким образом, выбор технологий .*NET Core, ASP.NET Core MVC*, *Entity Framework Core* и использование современных подходов к веб-разработке обеспечивают создание удобного, производительного и привлекательного веб-интерфейса для системы подписки на издания через почтовые отделения города.

В заключение, разработка информационной системы для управления подписками на издания через почтовые отделения города представляет собой актуальную задачу, объединяющую современные технологии и лучшие практики веб-разработки. Выбор программной архитектуры в сочетании с использованием платформы *.NET Core, ASP.NET Core MVC* и базы данных *MS SQL* с *Entity Framework Core* обеспечивает эффективное взаимодействие между компонентами системы, гарантируя их стабильность, масштабируемость и удобство сопровождения.

# 1 ЛОГИЧЕСКАЯ И ФИЗИЧЕСКАЯ СТРУКТУРА БАЗЫ ДАННЫХ

## 

## Информационно-логическая модель предметной области

Информационная система опирается на информационно-логическую модель предметной области. Эта модель представляет собой абстрактное отражение данных и их взаимосвязей в контексте организации, занимающейся распределением печатных изданий. Ниже представлена простая информационно-логическая модель, охватывающая ключевые сущности и их атрибуты в области управления подписками на издания через почтовые отделения.

*Employee* представляет сущность сотрудника в системе управления подписками на издания через почтовые отделения города. Oписание полей этой сущности:

*Id* – Уникальный идентификатор сотрудника. Это числовое поле используется для однозначной идентификации каждого сотрудника в базе данных.

*Name* – Имя сотрудника. Используется для хранения имени сотрудника.

*Middlename* – Отчество сотрудника. Используется для хранения отчества сотрудника.

*Surname* – Фамилия сотрудника.

*PositionId* ­– Идентификатор должности сотрудника. Это числовое поле, связанное с идентификатором должности, которую занимает сотрудник.

*OfficeId* ­– Идентификатор офиса, в котором работает сотрудник. Это числовое поле, связанное с идентификатором офиса.

*Office* – Ссылка на объект *Office*. Это отношение представляет связь между сотрудником и офисом, в котором он работает.

*Position* – Ссылка на объект *EmployeePosition*. Это отношение представляет связь между сотрудником и его должностью.

*FullName* ­­– Свойство, возвращающее полное имя сотрудника, состоящее из фамилии, имени и отчества.

Класс *EmployeePosition* представляет сущность должности сотрудника. Вот Свойств этой модели:

*Id* – Уникальный идентификатор должности. Это числовое поле используется для однозначной идентификации каждой должности в базе данных.

*Position* – Название должности. Используется для хранения названия конкретной должности.

*Employees* – Коллекция сотрудников, занимающих данную должность. Это виртуальное свойство представляет собой отношение "один ко многим" между должностью и сотрудниками, где одна должность может иметь много сотрудников.

Класс *Office* представляет сущность офиса в системе управления подписками. Описание полей и свойств этой модели:

*Id* – Уникальный идентификатор офиса. Это числовое поле используется для однозначной идентификации каждого офиса в базе данных.

*OwnerName* – Имя владельца офиса. Используется для хранения имени владельца офиса.

*OwnerMiddlename* – Отчество владельца офиса, используется для хранения отчества владельца офиса.

*OwnerSurname* – Фамилия владельца офиса.

*StreetName* – Название улицы, на которой расположен офис.

*MobilePhone* ­– Мобильный телефон офиса. Текстовое поле с ограничением максимальной длины в 15 символов, проверенное по шаблону телефонного номера.

*Email* –Адрес электронной почты офиса.

*Employees* – Коллекция сотрудников, работающих в данном офисе. Это виртуальное свойство представляет собой отношение "один ко многим" между офисом и сотрудниками.

*Subscriptions* – Коллекция подписок, оформленных через данный офис. Также виртуальное свойство, представляющее отношение "один ко многим" между офисом и подписками.

Класс *Publication* представляет сущность издания. Описание полей и свойств этой модели:

*Id* – Уникальный идентификатор издания. Это числовое поле используется для однозначной идентификации каждого издания в базе данных.

*TypeId* – Идентификатор типа издания. Это числовое поле, связанное с идентификатором типа издания.

*Name* – Название издания. Используется для хранения названия конкретного издания.

*Price* – Цена издания. Десятичное число с фиксированным количеством цифр до и после запятой, заданное с использованием атрибута *Column* для корректного представления в базе данных.

*Subscriptions* – Коллекция подписок, оформленных на данное издание. Виртуальное свойство представляет собой отношение "один ко многим" между изданием и подписками.

*Type* – Ссылка на объект "*PublicationType*". Виртуальное свойство представляет отношение "многие к одному" между изданием и его типом.

*PublicationType* представляет сущность типа издания. Описание полей и свойств этой модели:

*Id* – Уникальный идентификатор типа издания. Это числовое поле используется для однозначной идентификации каждого типа издания в базе данных.

*Type* – Название типа издания. Используется для хранения названия конкретного типа издания.

*Publications* ­– Коллекция изданий, относящихся к данному типу. Виртуальное свойство представляет собой отношение "один ко многим" между типом издания и соответствующими изданиями.

Класс *Recipient* представляет сущность получателя в системе управления подписками через почтовые отделения города. Описание полей и свойств этой модели:

*Id* – Уникальный идентификатор получателя. Это числовое поле используется для однозначной идентификации каждого получателя в базе данных.

*Name* – Имя получателя. Используется для хранения имени получателя.

*Middlename* – Отчество получателя. Используется для хранения отчества получателя.

*Surname* – Фамилия получателя. Используется для хранения фамилии получателя.

*AddressId* – Идентификатор адреса получателя. Это числовое поле, связанное с идентификатором адреса получателя.

*MobilePhone* – Мобильный телефон получателя. Текстовое поле с ограничением максимальной длины в 15 символов, проверенное по шаблону телефонного номера.

*Address* – Ссылка на объект *RecipientAddress*. Виртуальное свойство представляет собой отношение "один к одному" между получателем и его адресом.

*Subscriptions* – Коллекция подписок, оформленных на данного получателя. Виртуальное свойство представляет собой отношение "один ко многим" между получателем и его подписками.

*FullName* – Свойство, возвращающее полное имя получателя, состоящее из фамилии, имени и отчества.

*RecipientAddress* представляет сущность адреса получателя. Описание полей и свойств этой модели:

*Id* – Уникальный идентификатор адреса получателя. Это числовое поле используется для однозначной идентификации каждого адреса в базе данных.

*Street* – Название улицы, на которой расположен адрес. Используется для хранения названия улицы.

*House* – Номер дома по указанной улице. Это целочисленное поле, обязательное для заполнения.

*Apartment* – Номер квартиры в указанном доме.

*Recipients* – Коллекция получателей, проживающих по данному адресу. Виртуальное свойство представляет собой отношение "один ко многим" между адресом и соответствующими получателями.

*FullAddress* – Свойство, возвращающее полный адрес, состоящий из названия улицы, номера дома и номера квартиры (если указан).

Класс *Subscription* представляет сущность подписки в системе управления подписками через почтовые отделения города. Описание полей и свойств этой модели:

*Id* – Уникальный идентификатор подписки. Это числовое поле используется для однозначной идентификации каждой подписки в базе данных.

*RecipientId* – Идентификатор получателя, на которого оформлена подписка. Это числовое поле, связанное с идентификатором получателя.

*PublicationId* – Идентификатор издания, на которое оформлена подписка. Это числовое поле, связанное с идентификатором издания.

*Duration* – Длительность подписки в месяцах. Это числовое поле, обязательное для заполнения, с ограничением в пределах от 1 до максимального значения типа *int*.

*OfficeId* – Идентификатор офиса, через который оформлена подписка. Это числовое поле, связанное с идентификатором офиса.

*SubscriptionStartDate* – Дата начала подписки. Это поле типа *DateTime*, обязательное для заполнения, представляющее собой дату начала действия подписки.

*Office* – Ссылка на объект *Office*, через который оформлена подписка. Виртуальное свойство представляет отношение "многие к одному" между подпиской и офисом.

*Publication* – Ссылка на объект *Publication*, на которое оформлена подписка. Виртуальное свойство представляет отношение "многие к одному" между подпиской и изданием.

*Recipient* – Ссылка на объект *Recipient*, на которого оформлена подписка. Виртуальное свойство представляет отношение "многие к одному" между подпиской и получателем.

## 1.2 Физическая модель базы данных

В процессе преобразования логической модели в физическую были созданы таблицы, отражающие структуру данных в соответствии с ранее определенными отношениями. Каждая таблица содержит поля, соответствующие атрибутам сущностей, а связи между таблицами отражают установленные отношения между этими сущностями.

Были учтены особенности выбранной базы данных MS SQL Server, такие как типы данных, ограничения, индексы и другие параметры, необходимые для обеспечения эффективного хранения и обработки данных.

На полученной диаграмме базы данных представлены взаимосвязи между созданными таблицами, что позволяет легко воспринимать структуру базы данных и понимать, как связаны различные сущности в контексте приложения. Эта физическая модель базы данных является основой для эффективного хранения и управления информацией в рамках разработанной информационной системы., диаграмма базы данных и связи между сгенерированными таблицами представлены на рисунке 1.1.

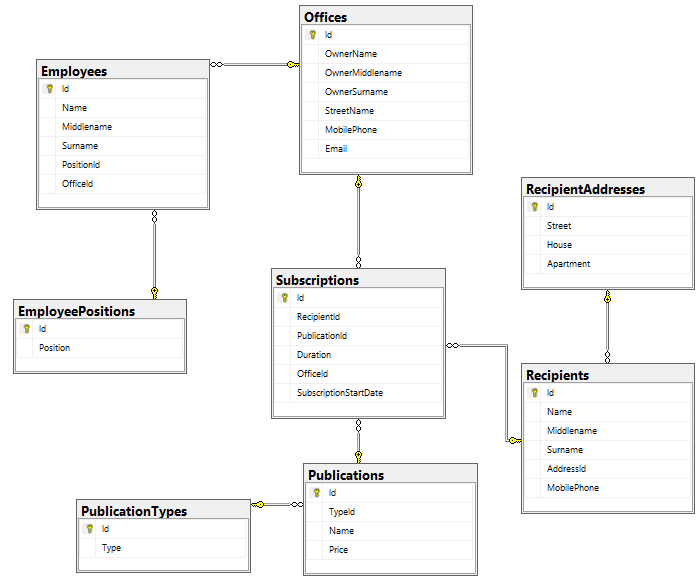


Рисунок 1.1 – Диаграмма базы данных

Для процесса преобразовании логической модели в физическую существует несколько правил:

– сущности становятся таблицами в физической базе данных;

– атрибуты становятся столбцами в физической базе данных, для каждого столбца необходимо определить подходящий тип данных.

В таблице 1.1 представлена информация о столбцах и типах данных для таблицы *EmployeePositions*.

Таблица 1.1 – Таблица *EmployeePositions*

|  |  |
| --- | --- |
| **Имя поля** | **Тип данных** |
| *Id* | *int* |
| *Position* | *nvarchar(100)* |

В таблице 1.2 представлена информация о столбцах и типах данных для таблицы *Employees*.

Таблица 1.2 – Таблица *Employees*

|  |  |
| --- | --- |
| **Имя поля** | **Тип данных** |
| *Id* | *int* |
| *Name* | *nvarchar(50)* |
| *Middlename* | *nvarchar(50)* |
| *Surname* | *nvarchar(50)* |
| *PositionId* | *int* |
| *OfficeId* | *int* |

В таблице 1.3 представлена информация о столбцах и типах данных для таблицы Offices.

Таблица 1.3 – Таблица Offices

|  |  |
| --- | --- |
| **Имя поля** | **Тип данных** |
| *Id* | *int* |
| *OwnerName* | *nvarchar(50)* |
| *OwnerMiddlename* | *nvarchar(50)* |
| *OwnerSurname* | *nvarchar(50)* |
| *StreetName* | *nvarchar(100)* |
| *MobilePhone* | *nvarchar(15)* |
| *Email* | *nvarchar(100)* |

В таблице 1.4 представлена подробная информация о столбцах и соответствующих типах данных для таблицы *Publications*.

Таблица 1.4 – Таблица *Publications*

|  |  |
| --- | --- |
| **Имя поля** | **Тип данных** |
| *Id* | *int* |
| *TypeId* | *int* |
| *Name* | *nvarchar(100)* |
| *Price* | *decimal(6, 2)* |

В таблице 1.5 представлена информация о столбцах и типах данных для таблицы *PublicationTypes*.

Таблица 1.5 – Таблица *PublicationTypes*

|  |  |
| --- | --- |
| **Имя поля** | **Тип данных** |
| *Id* | *int* |
| *Type* | *nvarchar(100)* |

В таблице 1.6 представлена информация о столбцах и типах данных для таблицы *RecipientAddresses*.

Таблица 1.6 – Таблица *RecipientAddresses*

|  |  |
| --- | --- |
| **Имя поля** | **Тип данных** |
| *Id* | *int* |
| *Street* | *nvarchar(255)* |
| *House* | *int* |
| *Apartment* | *int* |

В таблице 1.7 представлена информация о столбцах и типах данных для таблицы *InsuranceCases*.

Таблица 1.7 – Таблица *InsuranceCases*

|  |  |
| --- | --- |
| **Имя поля** | **Тип данных** |
| *Id* | *int* |
| *Date* | *datetime2(7)* |
| *Description* | *nvarchar(MAX)* |
| *SupportingDocumentId* | *int* |
| *InsurancePayment* | *decimal(18, 2)* |
| *ClientId* | *int* |

В таблице 1.8 представлена информация о столбцах и типах данных для таблицы *Recipients.*

Таблица 1.8 – Таблица *Recipients*

|  |  |
| --- | --- |
| **Имя поля** | **Тип данных** |
| *Id* | *int* |
| *Name* | *nvarchar(50)* |
| *Middlename* | *nvarchar(50))* |
| *Surname* | *nvarchar(50)* |
| *AddressId* | *int* |
| *MobilePhone* | *nvarchar(15)* |

В таблице 1.9 представлена информация о столбцах и типах данных для таблицы *Subscriptions.*

Таблица 1.9 – Таблица *Subscriptions*

|  |  |
| --- | --- |
| **Имя поля** | **Тип данных** |
| *Id* | *int* |
| *RecipientId* | *int* |
| *PublicationId* | *int* |
| *Duration* | *int* |
| *OfficeId* | *int* |

Продолжение таблицы 1.9

|  |  |
| --- | --- |
| *SubscriptionStartDate* | *datetime2(7)* |

Внедрение физической структуры базы данных, реализованное при помощи *Entity Framework* и языка программирования *C#*, не только предоставляет практическую реализацию логической модели, но и обеспечивает эффективное управление данными. Создание контекста данных, а также связывание классов приложения с таблицами базы данных, созданными через *Entity Framework,* предоставляет удобные средства взаимодействия с хранилищем данных.

Этот подход обеспечивает не только гибкость и эффективность при выполнении операций с данными, но также обеспечивает высокий уровень абстракции при работе с базой данных. Система страховой компании может легко адаптироваться к изменяющимся требованиям, а разработчики получают удобные инструменты для работы с информацией.

В целом, правильное построение и логической, и физической структуры базы данных выступает в качестве основы для эффективного функционирования информационной системы страховой компании. Такой тщательный дизайн способствует надежности, производительности и удобству использования системы, что становится критически важным в динамичной современной бизнес-среде.

# 2 АППАРАТНОЕ И ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ ИНФОРМАЦИОННОЙ СИСТЕМЫ

## 2.1 Требования к системному и прикладному программному обеспечению на стороне сервера хранилища данных

Жесткий диск: для *SQL Server* требуется как минимум 6 ГБ свободного места на диске. Требования к месту на диске определяются набором устанавливаемых компонентов *SQL Server*.

Монитор: для *SQL Server* требуется монитор *Super VGA* с разрешением 800x600 пикселей или более высоким.

Интернет: для поддержки функциональных средств Интернета требуется доступ к Интернету.

Память, минимальные:

– экспресс выпуски: 512 Мб;

– другие выпуски: 1 Гб.

Память, рекомендуется:

– экспресс выпуски: 1 Гб;

– другие выпуски: 4 Гб и выше, в зависимости от роста размеров БД.

Быстродействие процессора:

– минимум: процессор *x*64 с тактовой частотой 1,4 ГГц;

– рекомендуется: 2,0 ГГц и выше.

Тип процессора *x*64:

– *AMD Opteron*;

– *AMD Athlon* 64;

– *Intel Xeon* с поддержкой *Intel EM*64*T*;

– *Intel Pentium* IV с поддержкой *EM*64*T*.

Операционная система:

– *Windows* 10 *TH*1 1507 или более поздней версии;

– *Windows Server* 2016 или более поздней версии.

.*NET Framework*: минимальная версия операционной системы подразумевает минимальную версию платформы .*NET Framework*.

Сетевое программное обеспечение: поддерживаемые операционные системы для *SQL Server* содержат встроенное сетевое программное обеспечение. Именованные экземпляры и экземпляры по умолчанию изолированной установки поддерживают следующие сетевые протоколы: *Shared memory*, *Named Pipes* и *TCP*/*IP*.

## 2.2 Требования к системному и прикладному программному обеспечению на стороне *web*-сервера

Процессор, минимальные: *x*86- или *x*64-разрядный двухъядерный процессор с тактовой частотой 1,9 ГГц, с набором инструкций *SSE*2.

Процессор, рекомендуемые: *x*64-разрядный двухъядерный процессор с тактовой частотой 3,3 ГГц и выше, с набором инструкций *SSE*2.

Память, минимальные: ОЗУ объемом 2 Гб.

Память, рекомендуемые: ОЗУ объемом 4 Гб и более.

Дисплей: *Super VGA* с разрешением 1024 x 768.

Требования к сети:

Пропускная способность более 50 КБ/с (400 КБ/с);

Задержка менее 150 мс.

Веб-приложение может выполняться на следующих веб-браузерах, запущенных на соответствующих операционных системах:

*Microsoft Edge* (последняя открыто выпущенная версия) работает в *Windows* 11, *Windows* 10, *Windows* 8.1, *Windows* 8, *Windows* 7;

*Mozilla* *Firefox* (последняя публично выпущенная версия) работает в *Windows* 11, *Windows* 10, *Windows* 8.1, *Windows* 8 или *Windows* 7;

*Google* *Chrome* (последняя открыто выпущенная версия) работает в *Windows* 11, *Windows* 10, *Windows* 8.1, *Windows* 8, *Windows* 7, две последних публичных версии *Mac* *OS*;

*Apple* *Safari* (последняя общедоступная версия), работающая на двух последних публичных версиях *Mac* *OS* или на *Apple* *iPad*.

## 2.3 Действия системного администратора при настройке и развертывании *web*-приложения

Шаги системного администратора при конфигурации и развертывании веб-приложения включают в себя следующее:

Установка и настройка серверов:

– Установка операционной системы с последними обновлениями, предпочтительно использовать *Windows Server* для максимальной совместимости с экосистемой *Microsoft*.

– Загрузка необходимой версии *Microsoft SQL Server* с официального сайта, запуск процесса установки и последовательное следование инструкциям мастера. Выбор компонентов, таких как *SQL Server Database Engine*, и указание соответствующих параметров, таких как строки подключения и порты.

– Установка и настройка веб-сервера (например, *IIS*).

Развертывание веб-приложения:

– Загрузка и развертывание веб-приложения на веб-сервер, проверка успешности развертывания и убеждение в его корректной загрузке в браузере.

– Редактирование конфигурационных файлов веб-приложения, таких как web.config для платформы .*NET*, с учетом прав доступа. Внесение необходимых изменений в строки подключения к базе данных и проверка корректности подключения к предварительно настроенной СУБД.

Настройка безопасности:

– Установка и настройка системы мониторинга безопасности с определением правил для отслеживания событий безопасности.

– Настройка брандмауэра и других мер безопасности на сервере, включая использование встроенного брандмауэра операционной системы и определение правил, разрешающих необходимые сетевые соединения для веб-приложения.

Резервное копирование и мониторинг:

– Настройка системы резервного копирования данных.

– Внедрение инструментов мониторинга производительности и доступности, определение ключевых параметров и настройка системы уведомлений для реагирования на сбои или проблемы с производительностью.

Тестирование:

– Проведение функционального тестирования с отслеживанием соответствия каждой функции требованиям.

– Проверка работоспособности резервных копий и механизмов восстановления.

Документирование:

– Создание документации по настройке и администрированию системы.

– Фиксация конфигурации и параметров для будущего обслуживания.

# 3 СТРУКТУРА ПРИЛОЖЕНИЯ

## 3.1 Описание общей структуры приложения

В контексте разработки данной архитектуры был уделен особый акцент на принципы чистой архитектуры, предложенные Робертом Мартином. Основная идея заключалась в том, чтобы создать модульную систему, где компоненты могли бы взаимодействовать между собой, минимизируя связанность и обеспечивая высокую степень независимости.

Каждый компонент архитектуры является своеобразным "островом", предоставляющим интерфейс для взаимодействия с другими компонентами. Это обеспечивает удобство внесения изменений внутри конкретного компонента без необходимости пересмотра всей системы. Кроме того, такая структура упрощает тестирование и поддержку, поскольку каждый компонент может рассматриваться и изменяться как отдельная сущность.

Компонент *Domains* является хранилищем для всех классов моделей, предназначенных для взаимодействия с базой данных. В дополнение к этому, в компоненте содержатся вспомогательные модели с перечислениями, которые облегчают работу с данными. Этот компонент служит основным строительным блоком для представления и управления данными в системе.

Компонент *Repository* выступает в роли посредника между приложением и базой данных. В его функционал входит хранение контекста для доступа к базе данных, а также классы инициализации для обеспечения совместимости данных. Этот компонент обеспечивает эффективное взаимодействие с хранилищем данных и поддерживает целостность информации в базе.

Компонент *Service* представляет собой хранилище для классов, реализующих бизнес-правила и специфические функции, связанные с проектом. Здесь содержится логика, отвечающая за обработку данных, принятие решений и обеспечение необходимой функциональности системы.

Компонент *Web* представляет веб-приложение в его основной структуре. Включает в себя классы контроллеров, ответственных за обработку запросов, классы представлений, отвечающих за визуальное представление данных, а также классы промежуточного уровня (*Middleware*), обеспечивающих дополнительную логику обработки запросов перед передачей их контроллерам. Этот компонент обеспечивает пользовательский интерфейс и взаимодействие с конечными пользователями.

## 3.2 Описание компонента *Domains*

Класс *Employee* в слое *Domain*s представляет модель сотрудника в системе. Каждый сотрудник имеет уникальный идентификатор. Для обеспечения целостности данных, требуется указать обязательные атрибуты: имя, отчество и фамилия. Класс также содержит идентификаторы должности и офиса, к которым привязан сотрудник. Эти атрибуты представляют собой внешние ключи, связывающие сущность с другими моделями в системе.

Класс *EmployeePosition* представляет модель, описывающую должность сотрудника в системе. В его состав входят следующие атрибуты: уникальный идентификатор, который выступает в роли первичного ключа и используется для однозначной идентификации каждой конкретной должности и атрибут *Position* содержит название должности.

Класс также содержит навигационное свойство Employees, представляющее собой коллекцию сущностей типа Employee. Это свойство устанавливает связь между должностью и сотрудниками, занимающими данную позицию.

Класс *Office* в слое *Domains* представляет модель офиса в системе. Уникальный идентификатор идентифицирует каждый офис. Для обеспечения целостности данных обязательны атрибуты, такие как имя владельца, адрес офиса, мобильный телефон и электронная почта.

Класс также содержит навигационные свойства: *Employees* представляет коллекцию связанных сущностей типа *Employee*, представляющих сотрудников, привязанных к офису. *Subscriptions* представляет коллекцию связанных сущностей типа *Subscription*, представляющих абонементы, связанные с данным офисом.

Класс *Publication* представляет собой модель публикации в системе. Уникальный идентификатор служит для однозначной идентификации каждой публикации. Важными атрибутами этого класса являются *TypeId*, который обозначает идентификатор типа публикации, и *Name*, представляющий собой название публикации.

Дополнительно, класс содержит атрибут *Price*, который представляет собой цену публикации. Этот атрибут представлен в формате десятичного числа с точностью 6 знаков, два из которых идут после запятой.

Класс также включает навигационное свойство *Subscriptions*, представляющее собой коллекцию связанных сущностей типа *Subscription*. Эта коллекция связана с абонементами, которые соотносятся с данной публикацией.

Навигационное свойство *Type* указывает на связанную сущность типа *PublicationType*. Это свойство обогащает информацию о публикации, предоставляя дополнительные атрибуты типа.

Класс *PublicationType* в слое *Domains* представляет модель типа публикации в системе. Уникальный идентификатор служит для идентификации каждого типа публикации. Атрибут *Type* представляет собой название типа публикации.

Класс содержит навигационное свойство *Publications*, представляющее собой коллекцию связанных сущностей типа *Publication*. Эта коллекция связана с публикациями, которые соотносятся с данным типом.

Класс *Recipient* представляет модель получателя в системе. Уникальный идентификатор служит для идентификации каждого получателя. Обязательными атрибутами являются *Name*, *Middlename* и *Surname*.

Дополнительные атрибуты включают *AddressId*, представляющий идентификатор адреса получателя, и *MobilePhone*, представляющий контактный мобильный телефон получателя, ограниченный в 15 символов и прошедший валидацию на соответствие формату телефонного номера.

Класс также содержит навигационные свойства: *Address* представляет собой связанный адрес получателя, а *Subscriptions* представляет коллекцию связанных сущностей типа *Subscription*, обозначая абонементы, связанные с данным получателем.

Метод *ToString()* возвращает строковое представление получателя фамилии и имени, а свойство *FullName* представляет полное имя получателя, включая отчество.

Класс *RecipientAddress* в слое *Domains* представляет модель адреса получателя в системе. Уникальный идентификатор служит для однозначной идентификации каждого адреса. Обязательными атрибутами являются *Street* (название улицы) и *House* (номер дома), при этом *Street* ограничено в 255 символов. Дополнительным атрибутом является *Apartment* (номер квартиры).

Класс также включает навигационное свойство *Recipients*, представляющее собой коллекцию связанных сущностей типа *Recipient*. Эта коллекция связана с получателями, которые привязаны к данному адресу.

Свойство *FullAddress* представляет строковое представление полного адреса.

Класс *Subscription* в архитектуре *Domains* представляет модель подписки в системе. Уникальный идентификатор служит для уникальной идентификации каждой конкретной подписки. Важными атрибутами являются *RecipientId* (идентификатор получателя), *PublicationId* (идентификатор публикации), *Duration* (продолжительность подписки), *OfficeId* (идентификатор офиса) и *SubscriptionStartDate* (дата начала подписки).

Класс также включает в себя навигационные свойства: *Office*, связанное с офисом, к которому привязана подписка; *Publication*, связанное с публикацией, к которой относится подписка; и *Recipient*, связанное с получателем, оформившим данную подписку.

Интерфейс *IUser* в слое *Domains* предоставляет общие стандарты для моделей, представляющих сущности пользователей в системе. Он включает свойство *Id*, представляющее уникальный идентификатор пользователя, и свойство *FullName*, возвращающее полное имя пользователя. Этот интерфейс служит для обеспечения единообразия доступа к основным характеристикам пользователей в системе.

Класс *PostCityUser* в архитектуре *Domains* представляет модель пользователя с учетной записью пользователя в системе. Этот класс расширяет базовую модель *IdentityUser* из фреймворка *ASP.NET Core Identity* и добавляет дополнительное свойство *UserId*, которое является идентификатором пользователя, связывающим данный класс с конкретным пользователем в таблицах *Recipien* или *Employee*.

В директории *FilterModel* находятся классы, представляющие собой фильтры с набором полей, совпадающих с описаниями свойств соответствующих моделей.

В каталоге *Sort* размещены перечисления, предназначенные для сортировки каждой модели в двух вариантах – по возрастанию и убыванию.

## 3.3 Описание компонента *Repository*

Класс *PostCityContext* в рамках репозитория представляет собой контекст базы данных, ответственный за взаимодействие с различными сущностями системы. Этот контекст является наследником *IdentityDbContext<PostCityUser>*, включающего функциональность *ASP.NET Core Identity* для управления пользователями.

Класс содержит конструкторы для создания экземпляра контекста с параметрами по умолчанию и с пользовательскими опциями. Также включены виртуальные свойства для каждой сущности, представляющей таблицу в базе данных (например, *Employees*, *Offices*, и т.д.).

Класс *DbInitializer* представляет собой компонент для инициализации базы данных системы. Его целью является создание начальных данных, необходимых для корректной работы приложения. Класс включает метод *InitializeDb()*, который проверяет соединение с базой данных, мигрирует ее при необходимости, инициализирует роли, администратора и другие сущности.

Класс содержит набор методов, каждый из которых ответственен за инициализацию отдельной сущности (например, *InitializePublicationTypes()*, *InitializePublications()* и т.д.). Данные для инициализации генерируются случайным образом, что обеспечивает разнообразие и реалистичность данных. Метод *CreateAdmin()* создает администратора «*admin@gmail.com*» с паролем «*EQRu~ha+75hqIcr*». Метод *CreateEmployees()* создает пользователей с ролью «Employee» на основе данных сотрудников, а метод *CreateRecipient()* создает пользователей с ролью «*Recipient*» на основе данных получателей. Все созданные пользователи получают уникальные имена и пароли, соответствующие их ролям и идентификаторам. Такая инициализация обеспечивает создание пользователей различных ролей в системе управления подписками и присвоение им соответствующих ролей для обеспечения правильного функционирования системы.

Директория *Migrations* в проекте, использующем *Entity Framework Core*, предназначена для хранения файлов миграций. Эти файлы содержат описания изменений в структуре базы данных, необходимых для соответствия изменениям в моделях данных.

Миграции выполняют функции хранения, отслеживания и применения изменений в базе данных. Они также обеспечивают возможность отката изменений при необходимости вернуть базу данных к предыдущему состоянию.

## 3.4 Описание компонента *Service*

Папка *Cache* в компоненте *Service* включает в себя классы, ответственные за кэширование данных в приложении. В этой папке присутствует абстрактный класс *AppCache<T>*, который является базовым для всех классов кэширования в системе.

Абстрактный класс *AppCache<T>* использует интерфейс кэширования *IMemoryCache* из *Microsoft.Extensions.Caching.Memory* и предоставляет общие методы и поля для управления кэшем.

Ключевые элементы класса: Конструктор принимает объект контекста базы данных *PostCityContext* и интерфейс кэширования *IMemoryCache*. Инициализирует соответствующие поля класса для взаимодействия с базой данных и кэшем. Поле *\_saveTime* определяет время жизни объектов в кэше. Абстрактные методы *Get()*, *Set()*, *Set(IEnumerable<T> values)* предоставляют основные операции для работы с кэшем. Эти методы оставлены для конкретных реализаций в производных классах. Метод *Update()* вызывает метод *Set()* для обновления данных в кэше.

Этот абстрактный класс предоставляет общие методы для управления кэшированием и создает структуру для конкретных реализаций кэширования данных в системе.

Класс *CookiesManager* в компоненте *Service* предоставляет методы для работы с куками в контексте веб-приложения. Метод *SaveToCookies<T>* принимает объект *IResponseCookies*, ключ (*key*) и данные (*filterData*). Данные сериализуются в формат *JSON* с использованием библиотеки *Newtonsoft.Json* и добавляются в куки. Метод *GetFromCookies<T>* извлекает данные определенного типа *T* из кук, принимая объект *IRequestCookieCollection* и ключ (*key*). Если куки содержат данные для указанного ключа, происходит их десериализация из *JSON* в объект типа *T*. В случае отсутствия данных возвращается значение по умолчанию для типа *T*.

Класс *FilterBy<T>* в компоненте *Utils* предоставляет удобные методы для фильтрации коллекций объектов по различным критериям. Метод *FilterByInt* осуществляет фильтрацию по целочисленному свойству объектов. Если значение фильтра не равно *null*, возвращаются объекты, у которых указанное свойство совпадает с значением фильтра. В противном случае возвращается исходная коллекция. Аналогичные подходы используются в методах *FilterByDate*, *FilterByString*, *FilterByDecimal* и *FilterByPeriod*, но применяются к свойствам с типами *DateTime*, *string*, *decimal* и *DateTime* соответственно.

Этот класс предоставляет модульные и гибкие методы для фильтрации коллекций в зависимости от различных критериев.

Класс *UserRegistrationManager* в компоненте *Utils* предоставляет удобный способ регистрации пользователя с назначением ему роли в системе. Конструктор класса принимает экземпляры *UserManager<PostCityUser>* и *RoleManager<IdentityRole>*, что позволяет взаимодействовать с функциональностью управления пользователями и ролями в *ASP.NET Identity*.

Метод *RegisterUserWithRole* принимает модель *PostCityUserModel*, содержащую данные о пользователе, такие как электронная почта, пароль и роль. Сначала он проверяет, существует ли указанная роль, и, если нет, создает ее с использованием *RoleManager*. Затем создается новый экземпляр *PostCityUser* на основе переданных данных, и пользователь регистрируется с использованием *UserManager*. Если регистрация прошла успешно, пользователь добавляется в указанную роль.

Этот класс обеспечивает единообразный и простой способ регистрации пользователей с назначением им ролей в системе *ASP.NET Identity.*

## 3.5 Описание компонента *PostCity*

В этом компоненте хранятся *Views* и *Controllers* слои данного *MVC* приложения.

Компонент *Views* отвечает за отображение данных пользователю и представление пользовательского интерфейса. Это то, что видит пользователь в браузере. *Views* отображают информацию из модели в удобной для восприятия форме.

Компонент *Controllers* управляет потоком приложения и обрабатывает пользовательские запросы. Контроллеры принимают ввод от пользователя, взаимодействуют с моделью для получения необходимых данных и передают эти данные в компонент *Views* для отображения.

Основной класс для настройки веб-приложения, класс *Program*, играет важную роль в определении параметров, необходимых для его функционирования. В частности, этот класс отвечает за установку ключевых параметров, таких как строка подключения к базе данных. Но его функционал не ограничивается только этим – класс *Program* также предоставляет механизм внедрения необходимых классов в приложение, используя механизм *Dependency Injection* (*DI*).

Пример использования этого класса включает в себя следующие действия. В первую очередь, происходит создание строителя приложения (*WebApplication*) и определение необходимых сервисов. Затем, используя строку подключения из конфигурационных файлов, настраивается контекст базы данных с использованием *Entity Framework Core* для взаимодействия с базой данных *SQL Server*.

Далее, осуществляется настройка *Identity* для обеспечения аутентификации и авторизации пользователей, а также управления ролями. Дополнительно, регистрируется класс *UserRegistrationManager* как *transient* сервис, который будет использоваться в дальнейшем.

Для обеспечения работы с кэшем в памяти, сервисы *AddMemoryCache*, *AddDistributedMemoryCache* и *AddSession* добавляют соответствующую функциональность. Также регистрируются различные сервисы, отвечающие за управление кэшем для различных сущностей, таких как пользователи, офисы, публикации и другие.

Далее идет конфигурация различных сервисов и компонентов, таких как фильтры, менеджеры куки, инициализатор базы данных и другие. Наконец, приложение запускается и применяются необходимые промежуточные обработчики (*middleware*) для обеспечения правильной работы приложения, включая управление сессиями, обработку ошибок и другие аспекты.

Был внедрен механизм *Middleware. Middleware* (промежуточное программное обеспечение) представляет собой компонент или слой программы, который выполняет функции обработки запросов или операций между клиентским и серверным программным обеспечением. *Middleware* располагается между операционной системой и прикладным программным обеспечением, обеспечивая различные службы, такие как обработка запросов, аутентификация, авторизация, логирование и другие аспекты управления потоком данных.

Для реализации этого механизма, был использован класс *DbInitializerMiddleware*, который представляет собой *middleware* для инициализации базы данных. В конструкторе этого класса передается делегат *RequestDelegate* для обработки следующего запроса в цепочке *middleware*.

Метод *Invoke(DbInitializerMiddleware)* выполняет необходимые действия при обработке запроса. В данном случае, происходит проверка наличия ключа «*starting*» в сессии. Если ключ отсутствует, то инициализируется база данных при помощи класса *DbInitialize*r, который получает доступ к контексту базы данных, менеджерам ролей и пользователей. После успешной инициализации, устанавливается ключ «*starting*» в сессии для предотвращения повторной инициализации.

# 4 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

## 4.1 Назначение, условие применения и функционал

Приложение предназначено для управления подписками на издания почтовых отделений города.

К основным функциям клиента относятся:

– Регистрация и авторизация с указанием соответствующей информации.

– Просмотр списка изданий и подписок.

– Фильтрация данных в таблицах по содержанию полей.

– Сортировка данных в столбцах таблиц.

­ Просмотр личной информации пользователя.

– Возможность выхода из учетной записи и создание новой.

## 4.2 Графический интерфейс и его функционал

При запуске приложения на экране появляется основная страница, которая изменяется в зависимости от роли пользователя. Если пользователь посетил сайт, но не прошел авторизацию, то на главной странице будет представлена общая информация о приложении. Примеры главной страницы сайта для незарегистрированного пользователя указан на рисунке 4.1.

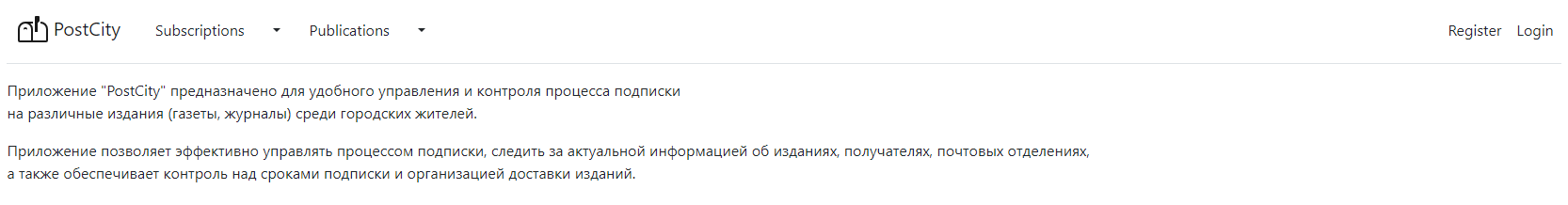


Рисунок 4.1 – Главная страница приложения для незарегистрированного пользователя

Если пользователь вошел в систему, выбрав роль работника или подписчика, на главной странице будет отображаться персонализированная информация о нем. Примеры главной страницы сайта для авторизованного пользователя указан на рисунке 4.2.

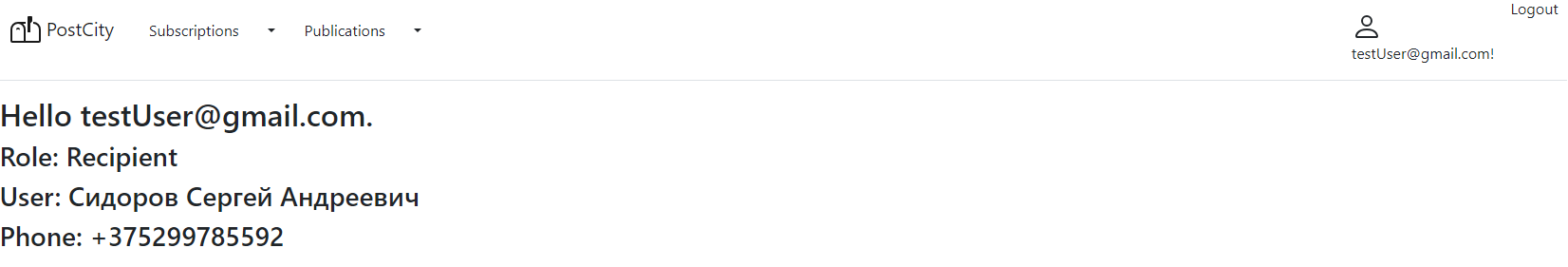


Рисунок 4.2 – Главная страница приложения для авторизованного пользователя

В случае авторизации под учетной записью администратора, на главной странице появится таблица с журналом операций, проведенных на сайте. Примеры главной страницы сайта для администратора указан на рисунке 4.2.

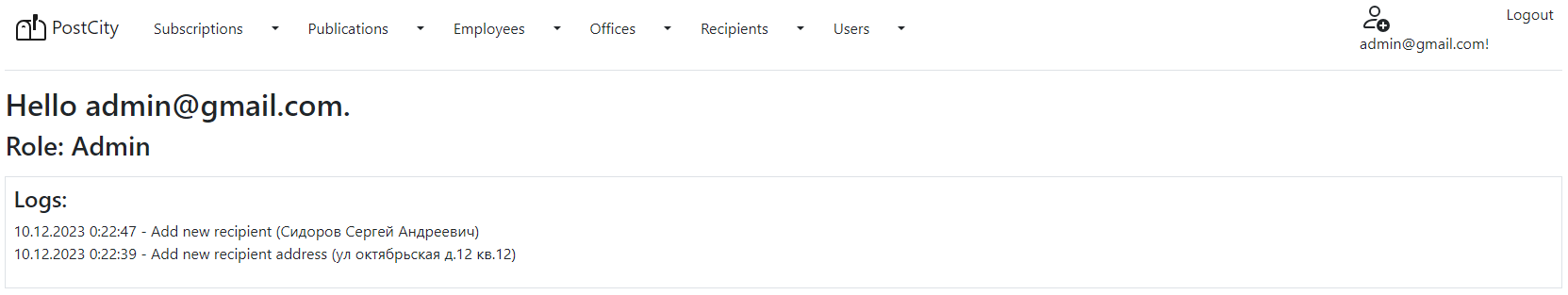


Рисунок 4.3 – Главная страница приложения для авторизованного пользователя

Для входа в систему была реализована страница, на которой пользователю необходимо ввести данные своей учетной записи. Пример страницы для входа в систему указан на рисунке 4.4

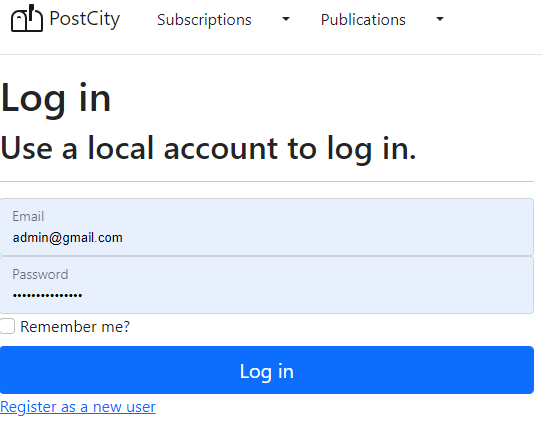


Рисунок 4.4 – Страница для входа в систему

Также на сайте была реализована страница для регистрации пользователя. На этой страницы пользователь должен ввести данные своей новой учетной записи. Пример страницы регистрации для пользователя указан на рисунке 4.5.

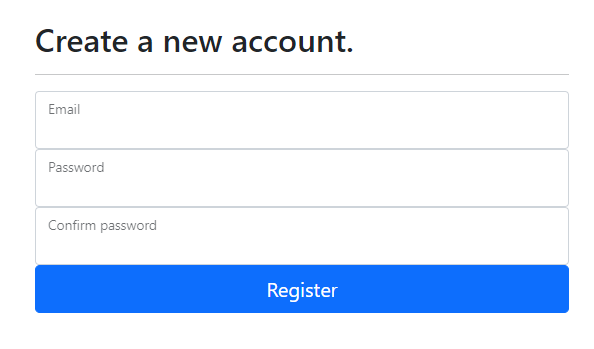


Рисунок 4.5 – Страница для регистрации пользователя

Далее пользователь переходит на страницу ввода личной информации, такой как имя, фамилия, отчество, мобильный телефон и адрес. Пример страницы для ввода личных данных представлен на рисунке 4.6.

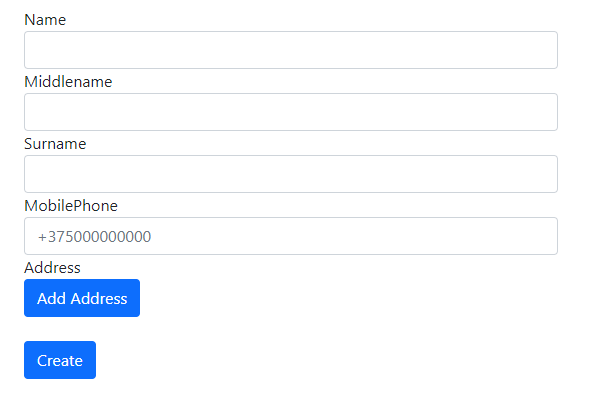


Рисунок 4.6 – Страница для регистрации пользователя

Пример страницы для ввода адреса пользователя указан на рисунке 4.7.

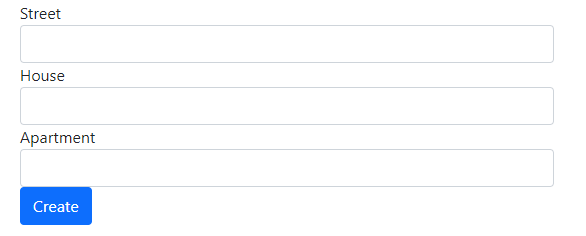


Рисунок 4.7 – Страница для регистрации пользователя

В приложении реализовано разделение доступной функциональности в зависимости от ролей пользователей, включая доступные для просмотра таблицы. Пример списка таблиц, доступных пользователю, отображается в панели навигации, представленной на рисунке 4.8.

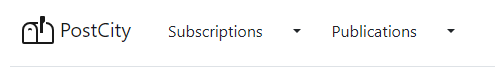


Рисунок 4.8 – Панель навигации пользователя

Администратор обладает большей функциональностью, пример панели навигации администратора указан на рисунке 4.9.



Рисунок 4.9 – Панель навигации администратора

Далее пользователь может просматривать информацию о конкретных сущностях. Для каждой сущности были созданы страницы, на которых отображается список всех сущностей, а также кнопки для добавления новой сущности, удаления, редактирования и получения деталей. Кроме того, для обеспечения удобства и эффективности работы сайта была реализована система разбиения всех сущностей на страницы. Рассмотрим пример работы с таблицами для сущности *Subscriptions*. Пример страницы для просмотра сущностей указана на рисунке 4.10.

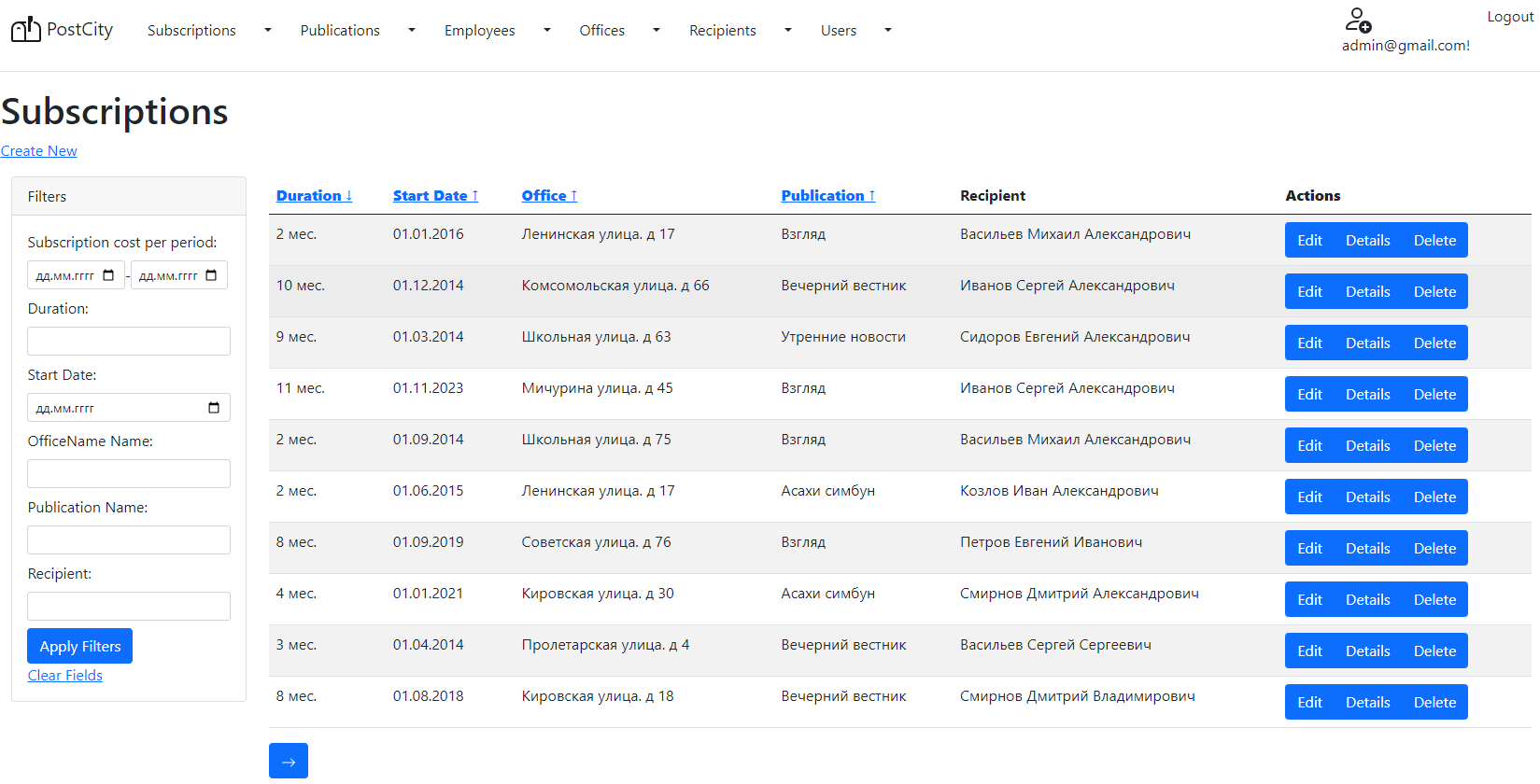


Рисунок 4.10 – Страница просмотра сущностей

Также для поиска нужного элемента в списке была внедрена система фильтров и сортировки. Для этого создано отдельное меню и кнопка подтверждения фильтрации. Пример фильтрации указан на рисунке 4.11.

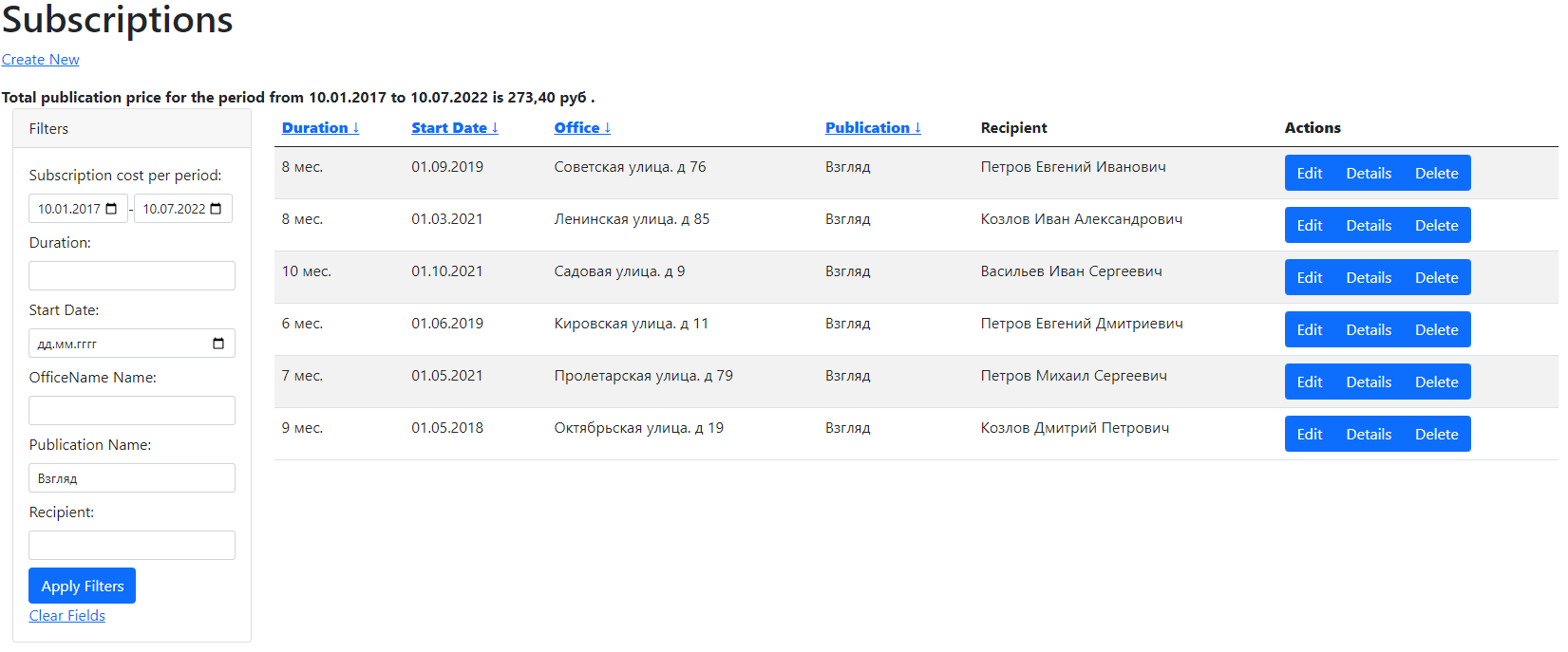


Рисунок 4.11 – Страница просмотра сущностей с фильтрацией

При добавлении нового элемента в базу данных открывается страница, на которой пользователю необходимо заполнить все поля. Чтобы гарантировать правильность и логичность введенных данных, были установлены ограничения на каждое из полей. Аналогичная страница также реализована для редактирования сущности. Пример страницы для добавления новой сущности указан на рисунке 4.12.

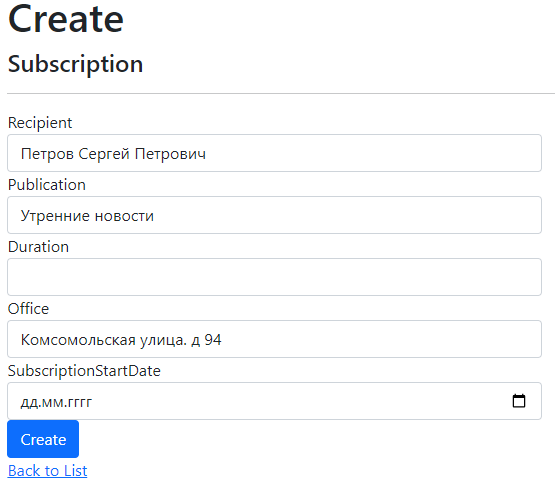


Рисунок 4.12 – Страница для добавления или обновления сущности

Пользователь также имеет возможность удалить необходимую сущность. Для этого он должен выбрать соответствующий элемент и, нажав на кнопку удаления, увидеть информацию об этой сущности. Также на экране отображаются кнопки для подтверждения удаления и отмены операции. Пример этой страницы указана на рисунке 4.13.

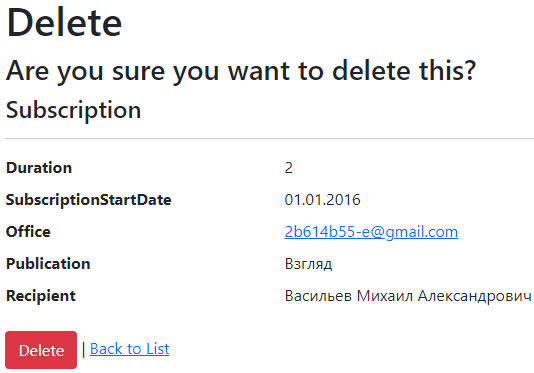


Рисунок 4.13 – Страница для удаления сущности

Также была создана страница для получения подробной информации о каждой сущности. На этой странице отображается детальная информация о сущности. Пример страницы, которая выводит детали указана на рисунке 4.14.

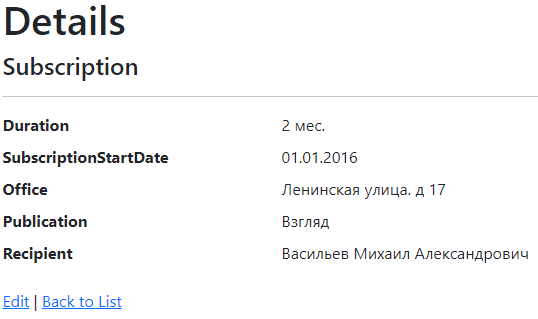


Рисунок 4.14 – Страница для получения подробной информации о сущности

# 5 РУКОВОДСТВО ПРОГРАММИСТА

1. Назначение и условия применения

Приложение было разработано с целью предоставления информации о подписках на издания в почтовых отделениях города. Оно выступает в роли инструмента для анализа и управления различными аспектами в сфере почтовых подписок. Среди основных функциональных возможностей:

– Управление записями в базе данных предоставляет пользователям возможность добавлять, изменять, просматривать и удалять записи, обеспечивая актуальность и точность данных.

– Многопользовательский доступ обеспечивает возможность одновременного использования приложения несколькими пользователями, эффективно согласовывая работу персонала.

– Развертывание функционала управления пользователями для администраторов включает в себя создание, редактирование и удаление учетных записей, а также установку прав доступа.

– Предоставление данных в удобном формате обеспечивает удобные и интуитивно понятные средства просмотра информации, что позволяет пользователям оперативно анализировать и визуализировать данные.

1. Характеристики программы

– Язык программирования и среда разработки: Разработка приложения осуществлена с использованием языка программирования *C#* в интегрированной среде разработки *Visual Studio 2022*. Этот подход обеспечивает удобство разработки и поддержки кода благодаря мощным инструментам и возможностям, предоставляемым *Visual Studio 2022.*

– СУБД и *Entity Framework*: Для хранения и управления данными применяется база данных *MS SQL Server*. Взаимодействие с базой данных реализовано с использованием *Entity Framework*, что обеспечивает эффективное взаимодействие приложения с хранилищем данных. *Entity Framework* обеспечивает удобство в работе с базой данных, предоставляя объектно-ориентированный подход к взаимодействию с данными.

– Серверная часть: Реализация серверной части осуществлена в виде *ASP.NET* приложения, способного обрабатывать запросы по протоколу *HTTP*. Это обеспечивает надежное функционирование и взаимодействие с клиентскими устройствами через стандартный протокол, что делает приложение доступным для использования на различных платформах.

1. Обращение к программе

– Для запуска приложения необходима установленная платформа .*NET* *Core* на сервере. Также требуется предварительная конфигурация параметров для соединения с базой данных. Для успешного подключения к базе данных необходимо настроить строку подключения в конфигурационном файле приложения *appsetting.json*. Для обеспечения соединения, СУБД и веб-приложение должны находиться в одной сети.

# ЗАКЛЮЧЕНИЕ

В процессе разработки курсового проекта, направленного на создание информационной системы для управления системой подписок на издания в почтовых отделениях города, на базе технологии *ASP.NET MVC* с использованием *Microsoft SQL Server*, были достигнуты значительные результаты. Проект предоставляет решение для управления процессами распространения подписок. Это включает в себя систему регистрации и авторизации, а также возможность просмотра записей таблиц.

Использование архитектурного шаблона *MVC* способствовало четкому разделению логики представления, бизнес-логики и управления пользовательским вводом. Это обеспечило легкость поддержки, возможность расширения функциональности и проведение тестирования приложения. Применение *ASP.NET MVC* и *.NET Core* дало системе высокую производительность, масштабируемость и кроссплатформенность.

Важным элементом разработанной информационной системы является *Microsoft SQL Server*, обеспечивающая надежное и безопасное хранение данных. Взаимодействие с базой данных реализовано с применением *Entity Framework*, что упростило взаимодействие с данными и обеспечило высокий уровень абстракции при работе с БД.

Предложенный проект не только предоставляет возможность получения подробной информации о клиентах, сотрудниках и подписках, но также обеспечивает систематизацию фильтрации, сортировки и внесения изменений в данных, что делает его комплексным решением для управления системой распространения подписок на почтовые издания.

# Список используемых источников

1. Албахари, Дж. С# 7.0. Карманный справочник / Дж. Албахари, Б. Албахари. – пер. с английского – СПб.: ООО «Альфа-книга», 2017. – 224 с.

2.Metanit [Электронный ресурс] – Режим доступа: https://metanit.com/sharp/aspnet5/16.1.php – Дата доступа: 20.04.2023.

3. Шилдт Г. C# 4.0: полное руководство: учебное пособие / Г. Шилдт. – пер. с англ.: – ООО «И.Д. Вильямс», 2011. – 1056 с.

4. Habr [Электронный ресурс] – Чистая Архитектура для веб-приложений – Режим доступа: https://habr.com/ru/articles/493430/ – Дата доступа: 12.11.2023

5. Роберт Мартин Чистая архитектура. Искусство разработки программного обеспечения / Роберт Сесил Мартин – пер. с англ.: – Санкт-Петербург, издательство «Библиотека программиста (Питер)», 2019. – 401 с.

6. Microsoft.com [Электронный ресурс] – Документация по ASP.NET – Режим доступа: https://learn.microsoft.com/ru-ru/aspnet/core/?view=aspnetcore-8.0 – Дата доступа: 09.11.2023

7. Metanit [Электронный ресурс] – Контроллеры и их действия – Режим доступа: https://metanit.com/sharp/aspnet5/5.1.php – Дата доступа: 10.11.2023

# ПРИЛОЖЕНИЕ А

(обязательное)

# Листинг программы

Листинг класса *PostCityUser*.*cs*:

using Microsoft.AspNetCore.Identity;

using System.ComponentModel.DataAnnotations;

namespace Domains.Models

{

public class PostCityUser : IdentityUser

{

[Required]

public int UserId { get; set; }

}

}

Листинг класса *PostCityUserModel.cs*:

namespace Domains.Models.Identity

{

public class PostCityUserModel

{

public string Role { get; set; }

public string Email { get; set; }

public string Password { get; set; }

public int UserId { get; set; }

}

}

Листинг класса *ErrorViewModel.cs*:

namespace Domains.Models.Standard

{

public class ErrorViewModel

{

public string? RequestId { get; set; }

public bool ShowRequestId => !string.IsNullOrEmpty(RequestId);

}

}

Листинг класса *Employee.cs*:

using System.ComponentModel.DataAnnotations;

namespace Domains.Models

{

public class Employee : IUser

{

public int Id { get; set; }

[Required]

[MaxLength(50)]

public string Name { get; set; }

[Required]

[MaxLength(50)]

public string Middlename { get; set; }

[Required]

[MaxLength(50)]

public string Surname { get; set; }

public int PositionId { get; set; }

public int OfficeId { get; set; }

public Office? Office { get; set; }

public EmployeePosition? Position { get; set; }

public override string ToString()

{

return $"{Surname} {Name}";

}

public string FullName

{

get { return $"{Surname} {Name} {Middlename}"; }

}

}

}

Листинг класса *EmployeePosition*.*cs*:

using System.ComponentModel.DataAnnotations;

namespace Domains.Models

{

public class EmployeePosition

{

public int Id { get; set; }

[Required]

[MaxLength(100)]

public string? Position { get; set; }

public virtual ICollection<Employee> Employees { get; set; } = new List<Employee>();

}

}

Листинг класса *IUser*.*cs*:

namespace Domains.Models

{

public interface IUser

{

public int Id { get; set; }

public string FullName { get;}

}

}

Листинг класса *Office*.*cs*:

using System.ComponentModel.DataAnnotations;

namespace Domains.Models

{

public class Office

{

public int Id { get; set; }

[Required]

[MaxLength(50)]

public string OwnerName { get; set; }

[Required]

[MaxLength(50)]

public string OwnerMiddlename { get; set; }

[Required]

[MaxLength(50)]

public string OwnerSurname { get; set; }

[Required]

[MaxLength(100)]

public string StreetName { get; set; }

[Required]

[MaxLength(15)]

[Phone]

public string MobilePhone { get; set; }

[Required]

[MaxLength(100)]

[EmailAddress]

public string Email { get; set; }

public virtual ICollection<Employee> Employees { get; set; } = new List<Employee>();

public virtual ICollection<Subscription> Subscriptions { get; set; } = new List<Subscription>();

}

}

Листинг класса *Publication*.*cs*:

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

using System.ComponentModel.DataAnnotations.Schema;

namespace Domains.Models

{

public class Publication

{

public int Id { get; set; }

public int TypeId { get; set; }

[Required]

[MaxLength(100)]

public string Name { get; set; } = null!;

[Required]

[Column(TypeName = "decimal(6, 2)")]

public decimal Price { get; set; }

public virtual ICollection<Subscription> Subscriptions { get; set; } = new List<Subscription>();

public virtual PublicationType? Type { get; set; } = null!;

public override string ToString()

{

return $"{Name}\t{Type.Type}\t{Price} руб.";

}

}

}

Листинг класса *PublicationType*.*cs*:

using System.ComponentModel.DataAnnotations;

namespace Domains.Models

{

public class PublicationType

{

public int Id { get; set; }

[Required]

[MaxLength(100)]

public string Type { get; set; } = null!;

public virtual ICollection<Publication> Publications { get; set; } = new List<Publication>();

}

}

Листинг класса *Recipient.cs*:

using System.ComponentModel.DataAnnotations;

namespace Domains.Models

{

public class Recipient : IUser

{

public int Id { get; set; }

[Required]

[MaxLength(50)]

public string Name { get; set; }

[Required]

[MaxLength(50)]

public string Middlename { get; set; }

[Required]

[MaxLength(50)]

public string Surname { get; set; }

public int? AddressId { get; set; }

[Required]

[MaxLength(15)]

[Phone]

public string MobilePhone { get; set; }

public virtual RecipientAddress? Address { get; set; }

public virtual ICollection<Subscription> Subscriptions { get; set; } = new List<Subscription>();

public override string ToString()

{

return $"{Surname} {Name}";

}

public string FullName

{

get { return $"{Surname} {Name} {Middlename}"; }

}

}

}

Листинг класса *ClientFilterModel*.*cs*:

using System.ComponentModel.DataAnnotations;

namespace Domains.Models

{

public class RecipientAddress

{

public int Id { get; set; }

[Required]

[MaxLength(255)]

public string Street { get; set; }

[Required]

public int House { get; set; }

public int Apartment { get; set; }

public virtual ICollection<Recipient> Recipients { get; set; } = new List<Recipient>();

public string FulAddress

{

get { return $"{Street} д.{House} кв.{Apartment}"; }

}

}

}

Листинг класса *Subscription*.*cs*:

using System.ComponentModel.DataAnnotations;

namespace Domains.Models

{

public class Subscription

{

public int Id { get; set; }

[Required]

public int RecipientId { get; set; }

[Required]

public int PublicationId { get; set; }

[Required]

[Range(1, int.MaxValue)]

public int Duration { get; set; }

[Required]

public int OfficeId { get; set; }

[Required]

[DataType(DataType.Date)]

public DateTime SubscriptionStartDate { get; set; }

public Office? Office { get; set; } = null!;

public Publication? Publication { get; set; } = null!;

public Recipient? Recipient { get; set; } = null!;

}

}

Листинг класса *EmployeeFilterModel*.*cs*:

namespace Domains.ViewModels.Filters.FilterModel

{

public class EmployeeFilterModel : ITableFilterModel

{

public string? Name { get; set; }

public string? Middlename { get; set; }

public string? Surname { get; set; }

public string? Office { get; set; }

public string? Position { get; set; }

}

}

Листинг класса *ITableFilterModel*.*cs*:

namespace Domains.ViewModels.Filters.FilterModel

{

public interface ITableFilterModel

{

}

}

Листинг класса *OfficeFilterModel*.*cs*:

namespace Domains.ViewModels.Filters.FilterModel

{

public class OfficeFilterModel : ITableFilterModel

{

public string OwnerName { get; set; }

public string OwnerMiddlename { get; set; }

public string OwnerSurname { get; set; }

public string StreetName { get; set; }

public string MobilePhone { get; set; }

public string Email { get; set; }

}

}

Листинг класса *PublicationFilterModel*.*cs*:

namespace Domains.ViewModels.Filters.FilterModel

{

public class PublicationFilterModel

{

public string? Name { get; set; }

public string? Type { get; set; }

public decimal? Price { get; set; }

}

}

Листинг класса *RecipientFilterModel*.*cs*:

namespace Domains.ViewModels.Filters.FilterModel

{

public class RecipientFilterModel : ITableFilterModel

{

public string? Name { get; set; }

public string? Middlename { get; set; }

public string? Surname { get; set; }

public string? MobilePhone { get; set; }

public string? Address { get; set; }

}

}

Листинг класса *SubscriptionFilterModel*.*cs*:

namespace Domains.ViewModels.Filters.FilterModel

{

public class SubscriptionFilterModel : ITableFilterModel

{

public int? Duration { get; set; }

public DateTime? StartDate { get; set; }

public string? OfficeName { get; set; }

public string? PublicationName { get; set; }

public string? RecipientName { get; set; }

public string? EmployeeName { get; set; }

public DateTime? StartPeriod { get; set; }

public DateTime? EndPeriod { get; set; }

}

}

Листинг класса *UserFilterModel*.*cs*:

namespace Domains.ViewModels.Filters.FilterModel

{

public class UserFilterModel : ITableFilterModel

{

public string? Name { get; set; }

public string? Email { get; set; }

public string? Role { get; set; }

}

}

Листинг класса *EmployeeSortState.cs*:

namespace Domains.ViewModels.Sort

{

public enum EmployeeSortState

{

StandardState,

NameAsc,

NameDesc,

MiddlenameAsc,

MiddlenameDesc,

SurnameAsc,

SurnameDesc,

OfficeAsc,

OfficeDesc,

PositionAsc,

PositionDesc

}

}

Листинг класса *OfficeSortState cs*:

namespace Domains.ViewModels.Sort

{

public enum OfficeSortState

{

StandardState,

OwnerNameAsc,

OwnerNameDesc,

OwnerMiddlenameAsc,

OwnerMiddlenameDecs,

OwnerSurnameAsc,

OwnerSurnameDesc,

StreetNameAsc,

StreetNameDesc,

}

}

Листинг класса *PublicationSortState*.*cs*:

namespace Domains.ViewModels.Sort

{

public enum PublicationSortState

{

StandardState,

NameAsc,

NameDesc,

TypeAsc,

TypeDesc,

PriceAsc,

PriceDesc

}

}

Листинг класса *RecipientSortState*.*cs*:

namespace Domains.ViewModels.Sort

{

public enum RecipientSortState

{

StandardState,

NameAsc,

NameDesc,

MiddlenameAsc,

MiddlenameDesc,

SurnameAsc,

SurnameDesc,

AddressAsc,

AddressDesc,

}

}

Листинг класса *SubscriptionSortState cs*:

namespace Domains.ViewModels.Sort

{

public enum SubscriptionSortState

{

StandardState,

DurationAsc,

DurationDesc,

DateAsc,

DateDesc,

SubscriptionNameAsc,

SubscriptionNameDesc,

PublicationNameAsc,

PublicationNameDesc,

OfficeNameAsc,

OfficeNameDesc

}

}

Листинг класса *UserSortState*.*cs*:

namespace Domains.ViewModels.Sort

{

public enum UserSortState

{

StandardState,

RoleAsc,

RoleDesc

}

Листинг класса *ChangeRoleViewModel.cs*:

namespace Domains.Models

{

public class ChangeRoleViewModel

{

public string UserId { get; set; }

public string UserEmail { get; set; }

public IEnumerable<string> UserRoles { get; set; }

public IEnumerable<Microsoft.AspNetCore.Identity.IdentityRole> AllRoles { get; set; }

}

}

Листинг класса *PageViewModel*.*cs*:

namespace Domains.ViewModels

{

public class PageViewModel<T, K>

{

public string Info { get; set; }

public int PageNumber { get; }

public int TotalPages { get; }

public bool HasPreviousPage => PageNumber > 1;

public bool HasNextPage => PageNumber < TotalPages;

private IEnumerable<T> \_items;

private K \_filterModel;

public PageViewModel(IEnumerable<T> pageDate, int page, int pageSize, K filterModel )

{

var count = pageDate.Count();

\_items = pageDate.Skip((page - 1) \* pageSize).Take(pageSize);

PageNumber = page;

TotalPages = (int)Math.Ceiling(count / (double)pageSize);

\_filterModel = filterModel;

}

public IEnumerable<T> Items { get { return \_items; } }

public K FilterModel { get { return \_filterModel; } }

}

}

Листинг класса *InsuranceCompanyCache*.*cs*:

using Microsoft.Extensions.Caching.Memory;

using Repository;

namespace Service {

public class InsuranceCompanyCache {

private IMemoryCache \_cache;

private InsuranceCompanyContext \_db;

private const int SAVE\_DURATION = 300;

public InsuranceCompanyCache(InsuranceCompanyContext db, IMemoryCache memoryCache) {

\_db = db;

\_cache = memoryCache;

}

public IEnumerable<T> GetEntity<T>() {

var entityName = typeof(T).Name;

\_cache.TryGetValue(entityName, out IEnumerable<T>? entities);

if (entities is null) {

entities = SetEntity<T>();

}

return entities;

}

public IEnumerable<T> SetEntity<T>() {

var entityName = typeof(T).Name;

var entities = InsuranceCompanyFactory.GetEnites<T>(entityName, \_db);

\_cache.Set(entityName, entities, new MemoryCacheEntryOptions().SetAbsoluteExpiration(TimeSpan.FromSeconds(SAVE\_DURATION)));

return entities;

}

}

}

Листинг класса *UserViewModel*.*cs*:

using Domains.Models;

namespace Domains.ViewModels

{

public class UserViewModel

{

public string Id { get; set; }

public IUser UserObj { get; set; }

public int UserId { get; set; }

public string Email { get; set; }

public string Role { get; set; }

}

}

Листинг класса *EmployeePositionsController*.*cs*:

using Microsoft.AspNetCore.Mvc;

using Microsoft.EntityFrameworkCore;

using Microsoft.AspNetCore.Authorization;

using Repository.Models;

using Utils;

using Domains.Models;

namespace PostCity.Controllers

{

[Authorize(Roles = "Admin")]

public class EmployeePositionsController : Controller

{

private readonly PostCityContext \_context;

private readonly SessionLogger \_logger;

public EmployeePositionsController(PostCityContext context, SessionLogger logger)

{

\_context = context;

\_logger = logger;

}

// GET: EmployeePositions

public async Task<IActionResult> Index()

{

return \_context.EmployeePositions != null ?

View(await \_context.EmployeePositions.ToListAsync()) :

Problem("Entity set 'PostCityContext.EmployeePositions' is null.");

}

public async Task<IActionResult> Details(int? id)

{

if (id == null || \_context.EmployeePositions == null)

{

return NotFound();

}

var employeePosition = await \_context.EmployeePositions

.FirstOrDefaultAsync(m => m.Id == id);

if (employeePosition == null)

{

return NotFound();

}

return View(employeePosition);

}

public IActionResult Create()

{

return View();

}

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<IActionResult> Create([Bind("Id,Position")] EmployeePosition employeePosition)

{

if (ModelState.IsValid)

{

\_context.Add(employeePosition);

await \_context.SaveChangesAsync();

\_logger.LogInformation($"Add new employee position ({employeePosition.Position})");

return RedirectToAction(nameof(Index));

}

return View(employeePosition);

}

public async Task<IActionResult> Edit(int? id)

{

if (id == null || \_context.EmployeePositions == null)

{

return NotFound();

}

var employeePosition = await \_context.EmployeePositions.FindAsync(id);

if (employeePosition == null)

{

return NotFound();

}

return View(employeePosition);

}

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<IActionResult> Edit(int id, [Bind("Id,Position")] EmployeePosition employeePosition)

{

if (id != employeePosition.Id)

{

return NotFound();

}

if (ModelState.IsValid)

{

try

{

\_context.Update(employeePosition);

\_logger.LogInformation($"Edit employee position ({employeePosition.Position})");

await \_context.SaveChangesAsync();

}

catch (DbUpdateConcurrencyException)

{

if (!EmployeePositionExists(employeePosition.Id))

{

return NotFound();

}

else

{

throw;

}

}

return RedirectToAction(nameof(Index));

}

return View(employeePosition);

}

public async Task<IActionResult> Delete(int? id)

{

if (id == null || \_context.EmployeePositions == null)

{

return NotFound();

}

var employeePosition = await \_context.EmployeePositions

.FirstOrDefaultAsync(m => m.Id == id);

if (employeePosition == null)

{

return NotFound();

}

return View(employeePosition);

}

[HttpPost, ActionName("Delete")]

[ValidateAntiForgeryToken]

public async Task<IActionResult> DeleteConfirmed(int id)

{

if (\_context.EmployeePositions == null)

{

return Problem("Entity set 'PostCityContext.EmployeePositions' is null.");

}

var employeePosition = await \_context.EmployeePositions.FindAsync(id);

if (employeePosition != null)

{

\_context.EmployeePositions.Remove(employeePosition);

\_logger.LogInformation($"Delete employee position ({employeePosition.Position})");

}

await \_context.SaveChangesAsync();

return RedirectToAction(nameof(Index));

}

private bool EmployeePositionExists(int id)

{

return (\_context.EmployeePositions?.Any(e => e.Id == id)).GetValueOrDefault();

}

}

}

Листинг класса *EmployeesController*.*cs*:

using System;

using System.Collections.Generic;

using System.Data;

using System.Linq;

using System.Threading.Tasks;

using Domains.Models;

using Domains.Models.Identity;

using Domains.ViewModels;

using Domains.ViewModels.Filters.FilterModel;

using Domains.ViewModels.Sort;

using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.Identity;

using Microsoft.AspNetCore.Mvc;

using Microsoft.AspNetCore.Mvc.Rendering;

using Microsoft.EntityFrameworkCore;

using Newtonsoft.Json;

using Repository.Models;

using Service.Data.Cache;

using Service.Data.Cookies;

using Utils;

namespace PostCity.Controllers

{

[Authorize(Roles = "Admin")]

public class EmployeesController : Controller, ISortOrderController<Employee, EmployeeSortState>

{

private readonly PostCityContext \_context;

private readonly EmployeeCache \_cache;

private readonly CookiesManeger \_cookies;

private readonly FilterBy<Employee> \_filter;

private readonly UserRegistrationManager \_userRegistrationManager;

private readonly UserManager<PostCityUser> \_userManager;

private readonly CacheUpdater \_cacheUpdater;

private readonly SessionLogger \_logger;

public EmployeesController(PostCityContext context,

EmployeeCache employeeCache,

CookiesManeger cookiesManeger,

FilterBy<Employee> filter,

UserRegistrationManager userRegistrationManager,

UserManager<PostCityUser> userManager,

CacheUpdater cacheUpdater,

SessionLogger logger)

{

\_context = context;

\_cache = employeeCache;

\_cookies = cookiesManeger;

\_filter = filter;

\_userRegistrationManager = userRegistrationManager;

\_userManager = userManager;

\_cacheUpdater = cacheUpdater;

\_logger = logger;

}

// GET: Employees

public async Task<IActionResult> Index(EmployeeSortState sortOrder = EmployeeSortState.StandardState, int page = 1)

{

var postCityContext = \_cache.Get();

EmployeeFilterModel filterData = \_cookies.GetFromCookies<EmployeeFilterModel>(Request.Cookies, "EmployeeFilterData");

SetSortOrderViewData(sortOrder);

postCityContext = ApplySortOrder(postCityContext, sortOrder).ToList();

int pageSize = 10;

var pageViewModel = new PageViewModel<Employee, EmployeeFilterModel>(postCityContext, page, pageSize, filterData);

return View(pageViewModel);

}

[HttpPost]

public async Task<IActionResult> Index(EmployeeFilterModel filterData, int page = 1)

{

\_cache.Update();

\_cookies.SaveToCookies(Response.Cookies, "EmployeeFilterData", filterData);

var data = \_cache.Get();

data = \_filter.FilterByString(data, pn => pn.Name, filterData.Name);

data = \_filter.FilterByString(data, pn => pn.Middlename, filterData.Middlename);

data = \_filter.FilterByString(data, pn => pn.Surname, filterData.Surname);

data = \_filter.FilterByString(data, pn => pn.Office.StreetName, filterData.Office);

data = \_filter.FilterByString(data, pn => pn.Position.Position, filterData.Position);

int pageSize = 10;

\_cache.Set(data);

var pageViewModel = new PageViewModel<Employee, EmployeeFilterModel>(data, page, pageSize, filterData);

return View(pageViewModel);

}

// GET: Employees/Details/5

public async Task<IActionResult> Details(int? id)

{

if (id == null || \_context.Employees == null)

{

return NotFound();

}

var employee = \_cache.Get().FirstOrDefault(m => m.Id == id);

if (employee == null)

{

return NotFound();

}

return View(employee);

}

// GET: Employees/Create

[Authorize(Roles = "Admin")]

public IActionResult Create()

{

ViewData["OfficeId"] = new SelectList(\_context.Offices, "Id", "StreetName");

ViewData["PositionId"] = new SelectList(\_context.EmployeePositions, "Id", "Position");

return View();

}

// POST: Employees/Create

// To protect from overposting attacks, enable the specific properties you want to bind to.

// For more details, see http://go.microsoft.com/fwlink/?LinkId=317598.

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<IActionResult> Create([Bind("Id,Name,Middlename,Surname,PositionId,OfficeId")] Employee employee)

{

if (ModelState.IsValid)

{

if (Request.Cookies.TryGetValue("EmployeeCredentials", out string credentialsJson))

{

var credentials = JsonConvert.DeserializeAnonymousType(credentialsJson, new { Email = "", Password = "" });

using var transaction = await \_context.Database.BeginTransactionAsync();

try

{

\_context.Add(employee);

await \_context.SaveChangesAsync();

var result = await \_userRegistrationManager.RegisterUserWithRole(new PostCityUserModel()

{

Role = "Employee",

Email = credentials.Email,

Password = credentials.Password,

UserId = employee.Id

});

if (result.Succeeded)

{

transaction.Commit();

\_cacheUpdater.Update(\_cache);

\_logger.LogInformation($"Add new employee ({employee.FullName})");

return Redirect("/User/Index");

}

else

{

transaction.Rollback();

\_context.Employees.Remove(employee);

foreach (var error in result.Errors)

{

ViewData["OfficeId"] = new SelectList(\_context.Offices, "Id", "StreetName");

ViewData["PositionId"] = new SelectList(\_context.EmployeePositions, "Id", "Position");

return View(employee);

}

}

}

catch (Exception)

{

transaction.Rollback();

}

}

}

ViewData["OfficeId"] = new SelectList(\_context.Offices, "Id", "StreetName");

ViewData["PositionId"] = new SelectList(\_context.EmployeePositions, "Id", "Position");

return View(employee);

}

// GET: Employees/Edit/5

[Authorize(Roles = "Admin")]

public async Task<IActionResult> Edit(int? id)

{

if (id == null || \_context.Employees == null)

{

return NotFound();

}

var employee = await \_context.Employees.FindAsync(id);

if (employee == null)

{

return NotFound();

}

ViewData["OfficeId"] = new SelectList(\_context.Offices, "Id", "StreetName", employee.OfficeId);

ViewData["PositionId"] = new SelectList(\_context.EmployeePositions, "Id", "Position", employee.PositionId);

return View(employee);

}

// POST: Employees/Edit/5

// To protect from overposting attacks, enable the specific properties you want to bind to.

// For more details, see http://go.microsoft.com/fwlink/?LinkId=317598.

[HttpPost]

[ValidateAntiForgeryToken]

[Authorize(Roles = "Admin")]

public async Task<IActionResult> Edit(int id, [Bind("Id,Name,Middlename,Surname,PositionId,OfficeId")] Employee employee)

{

if (id != employee.Id)

{

return NotFound();

}

if (ModelState.IsValid)

{

try

{

\_context.Update(employee);

await \_context.SaveChangesAsync();

\_cacheUpdater.Update(\_cache);

\_logger.LogInformation($"Edit employee ({employee.FullName})");

}

catch (DbUpdateConcurrencyException)

{

if (!EmployeeExists(employee.Id))

{

return NotFound();

}

else

{

throw;

}

}

return RedirectToAction(nameof(Index));

}

ViewData["OfficeId"] = new SelectList(\_context.Offices, "Id", "StreetName", employee.OfficeId);

ViewData["PositionId"] = new SelectList(\_context.EmployeePositions, "Id", "Position", employee.PositionId);

return View(employee);

}

// GET: Employees/Delete/5

[Authorize(Roles = "Admin")]

public async Task<IActionResult> Delete(int? id)

{

if (id == null || \_context.Employees == null)

{

return NotFound();

}

var employee =\_cache.Get().FirstOrDefault(m => m.Id == id);

if (employee == null)

{

return NotFound();

}

return View(employee);

}

// POST: Employees/Delete/5

[HttpPost, ActionName("Delete")]

[ValidateAntiForgeryToken]

public async Task<IActionResult> DeleteConfirmed(int id)

{

if (\_context.Employees == null)

{

return Problem("Entity set 'PostCityContext.Employees' is null.");

}

var employee = await \_context.Employees.FindAsync(id);

if (employee != null)

{

\_context.Employees.Remove(employee);

\_logger.LogInformation($"Delete employee ({employee.FullName})");

}

int isDelete = await \_context.SaveChangesAsync();

if (isDelete == 1)

{

PostCityUser user = \_context.FindUserByUserId(employee.Id);

IdentityResult result = await \_userManager.DeleteAsync(user);

}

\_cacheUpdater.Update(\_cache);

return RedirectToAction(nameof(Index));

}

private bool EmployeeExists(int id)

{

return (\_context.Employees?.Any(e => e.Id == id)).GetValueOrDefault();

}

public void SetSortOrderViewData(EmployeeSortState sortOrder)

{

ViewData["NameSort"] = sortOrder == EmployeeSortState.NameAsc

? EmployeeSortState.NameDesc

: EmployeeSortState.NameAsc;

ViewData["MiddlenameSort"] = sortOrder == EmployeeSortState.MiddlenameAsc

? EmployeeSortState.MiddlenameDesc

: EmployeeSortState.MiddlenameAsc;

ViewData["SurnameSort"] = sortOrder == EmployeeSortState.SurnameAsc

? EmployeeSortState.SurnameDesc

: EmployeeSortState.SurnameAsc;

ViewData["OfficeSort"] = sortOrder == EmployeeSortState.OfficeAsc

? EmployeeSortState.OfficeDesc

: EmployeeSortState.OfficeAsc;

ViewData["PositionSort"] = sortOrder == EmployeeSortState.PositionAsc

? EmployeeSortState.PositionDesc

: EmployeeSortState.PositionAsc;

}

public IEnumerable<Employee> ApplySortOrder(IEnumerable<Employee> postCityContext, EmployeeSortState sortOrder)

{

return sortOrder switch

{

EmployeeSortState.NameDesc => postCityContext.OrderByDescending(n => n.Name),

EmployeeSortState.NameAsc => postCityContext.OrderBy(n => n.Name),

EmployeeSortState.MiddlenameDesc => postCityContext.OrderByDescending(m => m.Middlename),

EmployeeSortState.MiddlenameAsc => postCityContext.OrderBy(m => m.Middlename),

EmployeeSortState.SurnameDesc => postCityContext.OrderByDescending(s => s.Surname),

EmployeeSortState.SurnameAsc => postCityContext.OrderBy(s => s.Surname),

EmployeeSortState.OfficeDesc => postCityContext.OrderByDescending(o => o.Office.StreetName),

EmployeeSortState.OfficeAsc => postCityContext.OrderBy(o => o.Office.StreetName),

EmployeeSortState.PositionDesc => postCityContext.OrderByDescending(p => p.Position.Position),

EmployeeSortState.PositionAsc => postCityContext.OrderBy(p => p.Position.Position),

EmployeeSortState.StandardState => postCityContext.ToList()

};

}

}

}

Листинг класса *HomeController*.*cs*:

using Domains.Models;

using Microsoft.AspNetCore.Identity;

using Microsoft.AspNetCore.Mvc;

using Microsoft.CodeAnalysis;

using Repository.Models;

using System.Diagnostics;

namespace PostCity.Controllers

{

public class HomeController : Controller

{

private readonly UserManager<PostCityUser> \_userManager;

private readonly PostCityContext \_context;

private readonly SessionLogger \_logger;

public HomeController(UserManager<PostCityUser> userManager, PostCityContext context, SessionLogger logger)

{

\_userManager = userManager;

\_context = context;

\_logger = logger;

}

public IActionResult Index()

{

PostCityUser currentUser = \_userManager.GetUserAsync(User).Result;

if (currentUser != null)

{

ViewData["Email"] = currentUser.Email;

var roles = \_userManager.GetRolesAsync(currentUser).Result;

var userRole = roles.FirstOrDefault();

ViewData["Role"] = userRole;

ViewData["UserId"] = currentUser.UserId;

if (userRole == "Employee")

{

Employee employee = \_context.Employees.Find(currentUser.UserId);

if (employee != null)

{

ViewData["Name"] = employee.FullName;

ViewData["Office"] = employee.OfficeId;

ViewData["Position"] = employee.Position;

}

}

else if (userRole == "Recipient")

{

Recipient recipient = \_context.Recipients.Find(currentUser.UserId);

if (recipient != null)

{

ViewData["Name"] = recipient.FullName;

ViewData["Phone"] = recipient.MobilePhone;

}

}

else if (userRole == "Admin")

{

}

}

List<string> logs = \_logger.GetSessionLogs();

return View(logs);

}

public IActionResult Privacy()

{

return View();

}

[ResponseCache(Duration = 0, Location = ResponseCacheLocation.None, NoStore = true)]

public IActionResult Error()

{

return View(new ErrorViewModel { RequestId = Activity.Current?.Id ?? HttpContext.TraceIdentifier });

}

}

}

Листинг класса *ISortOrderController*.*cs*:

namespace PostCity.Controllers

{

public interface ISortOrderController<T, K>

{

public void SetSortOrderViewData(K sortSatate);

public IEnumerable<T> ApplySortOrder(IEnumerable<T> data, K sortSatate);

}

}

Листинг класса *OfficesController*.*cs*:

using Microsoft.AspNetCore.Mvc;

using Microsoft.EntityFrameworkCore;

using Microsoft.AspNetCore.Authorization;

using System.Data;

using Domains.Models;

using Domains.ViewModels.Sort;

using Repository.Models;

using Service.Data.Cache;

using Service.Data.Cookies;

using Utils;

using Domains.ViewModels.Filters.FilterModel;

using Domains.ViewModels;

namespace PostCity.Controllers

{

public class OfficesController : Controller, ISortOrderController<Office, OfficeSortState>

{

private readonly PostCityContext \_context;

private readonly OfficeCache \_cache;

private readonly CookiesManeger \_cookies;

private readonly FilterBy<Office> \_filter;

private readonly CacheUpdater \_cacheUpdater;

private readonly SessionLogger \_logger;

public OfficesController(PostCityContext context,

OfficeCache officeCache,

CookiesManeger cookiesManeger,

FilterBy<Office> filterBy,

CacheUpdater cacheUpdater,

SessionLogger logger)

{

\_context = context;

\_cache = officeCache;

\_cookies = cookiesManeger;

\_filter = filterBy;

\_cacheUpdater = cacheUpdater;

\_logger = logger;

}

// GET: Offices

public async Task<IActionResult> Index(OfficeSortState sortOrder = OfficeSortState.StandardState, int page = 1)

{

var postCityContext = \_cache.Get();

OfficeFilterModel filterData = \_cookies.GetFromCookies<OfficeFilterModel>(Request.Cookies, "OfficeFilterData");

SetSortOrderViewData(sortOrder);

postCityContext = ApplySortOrder(postCityContext, sortOrder);

int pageSize = 10;

\_cache.Set(postCityContext);

var pageViewModel = new PageViewModel<Office, OfficeFilterModel>(postCityContext, page, pageSize, filterData);

return View(pageViewModel);

}

[HttpPost]

public async Task<IActionResult> Index(OfficeFilterModel filterData, int page = 1)

{

\_cache.Update();

\_cookies.SaveToCookies(Response.Cookies, "OfficeFilterData", filterData);

var data = \_cache.Get();

data = \_filter.FilterByString(data, pn => pn.OwnerName, filterData.OwnerName);

data = \_filter.FilterByString(data, pn => pn.OwnerSurname, filterData.OwnerSurname);

data = \_filter.FilterByString(data, pn => pn.OwnerMiddlename, filterData.OwnerMiddlename);

data = \_filter.FilterByString(data, pn => pn.StreetName, filterData.StreetName);

data = \_filter.FilterByString(data, pn => pn.MobilePhone, filterData.MobilePhone);

data = \_filter.FilterByString(data, pn => pn.Email, filterData.Email);

int pageSize = 10;

\_cache.Set(data);

var pageViewModel = new PageViewModel<Office, OfficeFilterModel>(data, page, pageSize, filterData);

return View(pageViewModel);

}

// GET: Offices/Details/5

public async Task<IActionResult> Details(int? id)

{

if (id == null || \_context.Offices == null)

{

return NotFound();

}

var office = \_cache.Get().FirstOrDefault(m => m.Id == id);

if (office == null)

{

return NotFound();

}

return View(office);

}

// GET: Offices/Create

public IActionResult Create()

{

return View();

}

// POST: Offices/Create

// To protect from overposting attacks, enable the specific properties you want to bind to.

// For more details, see http://go.microsoft.com/fwlink/?LinkId=317598.

[HttpPost]

[ValidateAntiForgeryToken]

[Authorize(Roles = "Admin")]

public async Task<IActionResult> Create([Bind("Id,OwnerName,OwnerMiddlename,OwnerSurname,StreetName,MobilePhone,Email")] Office office)

{

if (ModelState.IsValid)

{

\_context.Add(office);

await \_context.SaveChangesAsync();

\_cacheUpdater.Update(\_cache);

\_logger.LogInformation($"Add new office ({office.StreetName})");

return RedirectToAction(nameof(Index));

}

return View(office);

}

// GET: Offices/Edit/5

[Authorize(Roles = "Admin")]

public async Task<IActionResult> Edit(int? id)

{

if (id == null || \_context.Offices == null)

{

return NotFound();

}

var office = await \_context.Offices.FindAsync(id);

if (office == null)

{

return NotFound();

}

return View(office);

}

// POST: Offices/Edit/5

// To protect from overposting attacks, enable the specific properties you want to bind to.

// For more details, see http://go.microsoft.com/fwlink/?LinkId=317598.

[HttpPost]

[ValidateAntiForgeryToken]

[Authorize(Roles = "Admin")]

public async Task<IActionResult> Edit(int id, [Bind("Id,OwnerName,OwnerMiddlename,OwnerSurname,StreetName,MobilePhone,Email")] Office office)

{

if (id != office.Id)

{

return NotFound();

}

if (ModelState.IsValid)

{

try

{

\_context.Update(office);

await \_context.SaveChangesAsync();

\_cacheUpdater.Update(\_cache);

\_logger.LogInformation($"Edit office ({office.StreetName})");

}

catch (DbUpdateConcurrencyException)

{

if (!OfficeExists(office.Id))

{

return NotFound();

}

else

{

throw;

}

}

return RedirectToAction(nameof(Index));

}

return View(office);

}

// GET: Offices/Delete/5

[Authorize(Roles = "Admin")]

public async Task<IActionResult> Delete(int? id)

{

if (id == null || \_context.Offices == null)

{

return NotFound();

}

var office = \_cache.Get().FirstOrDefault(m => m.Id == id);

if (office == null)

{

return NotFound();

}

return View(office);

}

// POST: Offices/Delete/5

[HttpPost, ActionName("Delete")]

[ValidateAntiForgeryToken]

public async Task<IActionResult> DeleteConfirmed(int id)

{

if (\_context.Offices == null)

{

return Problem("Entity set 'PostCityContext.Offices' is null.");

}

var office = await \_context.Offices.FindAsync(id);

if (office != null)

{

\_context.Offices.Remove(office);

\_logger.LogInformation($"Delete office ({office.StreetName})");

}

await \_context.SaveChangesAsync();

\_cacheUpdater.Update(\_cache);

return RedirectToAction(nameof(Index));

}

private bool OfficeExists(int id)

{

return (\_context.Offices?.Any(e => e.Id == id)).GetValueOrDefault();

}

public void SetSortOrderViewData(OfficeSortState sortOrder)

{

ViewData["OwnerNameSort"] = sortOrder == OfficeSortState.OwnerNameAsc

? OfficeSortState.OwnerNameDesc

: OfficeSortState.OwnerNameAsc;

ViewData["OwnerMiddlenameSort"] = sortOrder == OfficeSortState.OwnerMiddlenameAsc

? OfficeSortState.OwnerMiddlenameDecs

: OfficeSortState.OwnerMiddlenameAsc;

ViewData["OwnerSurnameSort"] = sortOrder == OfficeSortState.OwnerSurnameAsc

? OfficeSortState.OwnerSurnameDesc

: OfficeSortState.OwnerSurnameAsc;

ViewData["StreetNameSort"] = sortOrder == OfficeSortState.StreetNameAsc

? OfficeSortState.StreetNameDesc

: OfficeSortState.StreetNameAsc;

}

public IEnumerable<Office> ApplySortOrder(IEnumerable<Office> postCityContext, OfficeSortState sortOrder)

{

return sortOrder switch

{

OfficeSortState.OwnerNameDesc => postCityContext.OrderByDescending(n => n.OwnerName),

OfficeSortState.OwnerNameAsc => postCityContext.OrderBy(n => n.OwnerName),

OfficeSortState.OwnerSurnameAsc => postCityContext.OrderByDescending(s => s.OwnerSurname),

OfficeSortState.OwnerSurnameDesc => postCityContext.OrderBy(s => s.OwnerSurname),

OfficeSortState.OwnerMiddlenameAsc => postCityContext.OrderByDescending(m => m.OwnerMiddlename),

OfficeSortState.OwnerMiddlenameDecs => postCityContext.OrderBy(m => m.OwnerMiddlename),

OfficeSortState.StreetNameAsc => postCityContext.OrderByDescending(s => s.StreetName),

OfficeSortState.StreetNameDesc => postCityContext.OrderBy(s => s.StreetName),

OfficeSortState.StandardState => postCityContext.ToList()

};

}

}

}

Листинг класса *PublicationsController*.*cs*:

using Microsoft.AspNetCore.Mvc;

using Microsoft.AspNetCore.Mvc.Rendering;

using Microsoft.EntityFrameworkCore;

using Microsoft.AspNetCore.Authorization;

using System.Data;

using PostCity.Controllers;

using Domains.Models;

using Domains.ViewModels.Sort;

using Repository.Models;

using Service.Data.Cache;

using Service.Data.Cookies;

using Utils;

using Domains.ViewModels.Filters.FilterModel;

using Domains.ViewModels;

namespace PostCity.Controllers

{

public class PublicationsController : Controller, ISortOrderController<Publication, PublicationSortState>

{

private readonly PostCityContext \_context;

private readonly PublicationCache \_cache;

private readonly CookiesManeger \_cookies;

private readonly FilterBy<Publication> \_filter;

private readonly CacheUpdater \_cacheUpdater;

private readonly SessionLogger \_logger;

public PublicationsController(PostCityContext context,

PublicationCache publicationCache,

CookiesManeger cookies,

FilterBy<Publication> filter,

CacheUpdater cacheUpdater,

SessionLogger sessionLogger)

{

\_context = context;

\_cache = publicationCache;

\_cookies = cookies;

\_filter = filter;

\_cacheUpdater = cacheUpdater;

\_logger = sessionLogger;

}

public async Task<IActionResult> Index(PublicationSortState sortOrder = PublicationSortState.StandardState, int page = 1)

{

var postCityContext = \_cache.Get();

PublicationFilterModel filterData = \_cookies.GetFromCookies<PublicationFilterModel>(Request.Cookies, "PublicationFilterData");

SetSortOrderViewData(sortOrder);

postCityContext = ApplySortOrder(postCityContext, sortOrder);

int pageSize = 10;

\_cache.Set(postCityContext);

var pageViewModel = new PageViewModel<Publication, PublicationFilterModel>(postCityContext, page, pageSize, filterData);

return View(pageViewModel);

}

[HttpPost]

public async Task<IActionResult> Index(PublicationFilterModel filterData, int page = 1)

{

\_cache.Update();

\_cookies.SaveToCookies(Response.Cookies, "PublicationFilterData", filterData);

var data = \_cache.Get();

data = \_filter.FilterByString(data, pn => pn.Name, filterData.Name);

data = \_filter.FilterByString(data, pn => pn.Type.Type, filterData.Type);

data = \_filter.FilterByDecimal(data, pn => pn.Price, filterData.Price);

int pageSize = 10;

\_cache.Set(data);

var pageViewModel = new PageViewModel<Publication, PublicationFilterModel>(data, page, pageSize, filterData);

return View(pageViewModel);

}

public async Task<IActionResult> Details(int? id)

{

if (id == null || \_context.Publications == null)

{

return NotFound();

}

var publication = \_cache.Get().FirstOrDefault(m => m.Id == id);

if (publication == null)

{

return NotFound();

}

return View(publication);

}

[Authorize(Roles = "Admin")]

public IActionResult Create()

{

ViewData["TypeId"] = new SelectList(\_context.PublicationTypes, "Id", "Type");

return View();

}

[Authorize(Roles = "Admin")]

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<IActionResult> Create([Bind("Id,TypeId,Name,Price")] Publication publication)

{

if (ModelState.IsValid)

{

\_context.Add(publication);

await \_context.SaveChangesAsync();

\_cacheUpdater.Update(\_cache);

\_logger.LogInformation($"Add new publication ({publication.Name})");

return RedirectToAction(nameof(Index));

}

ViewData["TypeId"] = new SelectList(\_context.PublicationTypes, "Id", "Type", publication.TypeId);

return View(publication);

}

[Authorize(Roles = "Admin")]

public async Task<IActionResult> Edit(int? id)

{

if (id == null || \_context.Publications == null)

{

return NotFound();

}

var publication = await \_context.Publications.FindAsync(id);

if (publication == null)

{

return NotFound();

}

ViewData["TypeId"] = new SelectList(\_context.PublicationTypes, "Id", "Type", publication.TypeId);

return View(publication);

}

[Authorize(Roles = "Admin")]

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<IActionResult> Edit(int id, [Bind("Id, TypeId,Name,Price")] Publication publication)

{

if (id != publication.Id)

{

return NotFound();

}

if (ModelState.IsValid)

{

try

{

\_context.Update(publication);

await \_context.SaveChangesAsync();

\_cacheUpdater.Update(\_cache);

\_logger.LogInformation($"Edit publication ({publication.Name})");

}

catch (DbUpdateConcurrencyException)

{

if (!PublicationExists(publication.Id))

{

return NotFound();

}

else

{

throw;

}

}

return RedirectToAction(nameof(Index));

}

ViewData["TypeId"] = new SelectList(\_context.PublicationTypes, "Id", "Type", publication.TypeId);

return View(publication);

}

[Authorize(Roles = "Admin")]

public async Task<IActionResult> Delete(int? id)

{

if (id == null || \_context.Publications == null)

{

return NotFound();

}

var publication = \_cache.Get().FirstOrDefault(m => m.Id == id);

if (publication == null)

{

return NotFound();

}

return View(publication);

}

[Authorize(Roles = "Admin")]

[HttpPost, ActionName("Delete")]

[ValidateAntiForgeryToken]

public async Task<IActionResult> DeleteConfirmed(int id)

{

if (\_context.Publications == null)

{

return Problem("Entity set 'PostCityContext.Publications' is null.");

}

var publication = await \_context.Publications.FindAsync(id);

if (publication != null)

{

\_context.Publications.Remove(publication);

\_logger.LogInformation($"Delete publication ({publication.Name})");

}

await \_context.SaveChangesAsync();

\_cacheUpdater.Update(\_cache);

return RedirectToAction(nameof(Index));

}

private bool PublicationExists(int id)

{

return (\_context.Publications?.Any(e => e.Id == id)).GetValueOrDefault();

}

public void SetSortOrderViewData(PublicationSortState sortOrder)

{

ViewData["NameSort"] = sortOrder == PublicationSortState.NameAsc

? PublicationSortState.NameDesc

: PublicationSortState.NameAsc;

ViewData["TypeSort"] = sortOrder == PublicationSortState.TypeAsc

? PublicationSortState.TypeDesc

: PublicationSortState.TypeAsc;

ViewData["PriceSort"] = sortOrder == PublicationSortState.PriceAsc

? PublicationSortState.PriceDesc

: PublicationSortState.PriceAsc;

}

public IEnumerable<Publication> ApplySortOrder(IEnumerable<Publication> postCityContext, PublicationSortState sortOrder)

{

return sortOrder switch

{

PublicationSortState.NameDesc => postCityContext.OrderByDescending(n => n.Name),

PublicationSortState.NameAsc => postCityContext.OrderBy(n => n.Name),

PublicationSortState.TypeDesc => postCityContext.OrderByDescending(t => t.Type.Type),

PublicationSortState.TypeAsc => postCityContext.OrderBy(t => t.Type.Type),

PublicationSortState.PriceDesc => postCityContext.OrderByDescending(p => p.Price),

PublicationSortState.PriceAsc => postCityContext.OrderBy(p => p.Price),

PublicationSortState.StandardState => postCityContext

};

}

}

}

Листинг класса *PublicationTypesController*.*cs*:

using Microsoft.AspNetCore.Mvc;

using Microsoft.EntityFrameworkCore;

using Microsoft.AspNetCore.Authorization;

using Repository.Models;

using Service.Data.Cache;

using Utils;

using Domains.Models;

namespace PostCity.Controllers

{

[Authorize(Roles = "Admin")]

public class PublicationTypesController : Controller

{

private readonly PostCityContext \_context;

private readonly PublicationCache \_publicationCache;

private readonly SessionLogger \_logger;

public PublicationTypesController(PostCityContext context, PublicationCache publicationCache, SessionLogger logger)

{

\_context = context;

\_publicationCache = publicationCache;

\_logger = logger;

}

// GET: PublicationTypes

public async Task<IActionResult> Index()

{

return \_context.PublicationTypes != null ?

View(await \_context.PublicationTypes.ToListAsync()) :

Problem("Entity set 'PostCityContext.PublicationTypes' is null.");

}

// GET: PublicationTypes/Details/5

public async Task<IActionResult> Details(int? id)

{

if (id == null || \_context.PublicationTypes == null)

{

return NotFound();

}

var publicationType = await \_context.PublicationTypes

.FirstOrDefaultAsync(m => m.Id == id);

if (publicationType == null)

{

return NotFound();

}

return View(publicationType);

}

// GET: PublicationTypes/Create

public IActionResult Create()

{

return View();

}

// POST: PublicationTypes/Create

// To protect from overposting attacks, enable the specific properties you want to bind to.

// For more details, see http://go.microsoft.com/fwlink/?LinkId=317598.

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<IActionResult> Create([Bind("Id,Type")] PublicationType publicationType)

{

if (ModelState.IsValid)

{

\_context.Add(publicationType);

await \_context.SaveChangesAsync();

\_logger.LogInformation($"Add new publication type ({publicationType.Type})");

return RedirectToAction(nameof(Index));

}

return View(publicationType);

}

// GET: PublicationTypes/Edit/5

public async Task<IActionResult> Edit(int? id)

{

if (id == null || \_context.PublicationTypes == null)

{

return NotFound();

}

var publicationType = await \_context.PublicationTypes.FindAsync(id);

if (publicationType == null)

{

return NotFound();

}

return View(publicationType);

}

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<IActionResult> Edit(int id, [Bind("Id,Type")] PublicationType publicationType)

{

if (id != publicationType.Id)

{

return NotFound();

}

if (ModelState.IsValid)

{

try

{

\_context.Update(publicationType);

\_publicationCache.Update();

\_logger.LogInformation($"Edit publication type ({publicationType.Type})");

await \_context.SaveChangesAsync();

}

catch (DbUpdateConcurrencyException)

{

if (!PublicationTypeExists(publicationType.Id))

{

return NotFound();

}

else

{

throw;

}

}

return RedirectToAction(nameof(Index));

}

return View(publicationType);

}

// GET: PublicationTypes/Delete/5

public async Task<IActionResult> Delete(int? id)

{

if (id == null || \_context.PublicationTypes == null)

{

return NotFound();

}

var publicationType = await \_context.PublicationTypes

.FirstOrDefaultAsync(m => m.Id == id);

if (publicationType == null)

{

return NotFound();

}

return View(publicationType);

}

// POST: PublicationTypes/Delete/5

[HttpPost, ActionName("Delete")]

[ValidateAntiForgeryToken]

public async Task<IActionResult> DeleteConfirmed(int id)

{

if (\_context.PublicationTypes == null)

{

return Problem("Entity set 'PostCityContext.PublicationTypes' is null.");

}

var publicationType = await \_context.PublicationTypes.FindAsync(id);

if (publicationType != null)

{

\_context.PublicationTypes.Remove(publicationType);

\_publicationCache.Update();

\_logger.LogInformation($"Delete publication type ({publicationType.Type})");

}

await \_context.SaveChangesAsync();

return RedirectToAction(nameof(Index));

}

private bool PublicationTypeExists(int id)

{

return (\_context.PublicationTypes?.Any(e => e.Id == id)).GetValueOrDefault();

}

}

}

Листинг класса *RecipientAddressesController*.*cs*:

using Microsoft.AspNetCore.Mvc;

using Microsoft.EntityFrameworkCore;

using Newtonsoft.Json;

using Microsoft.AspNetCore.Authorization;

using Repository.Models;

using Utils;

using Domains.Models;

namespace PostCity.Controllers

{

public class RecipientAddressesController : Controller

{

private readonly PostCityContext \_context;

private readonly SessionLogger \_logger;

public RecipientAddressesController(PostCityContext context, SessionLogger logger)

{

\_context = context;

\_logger = logger;

}

[Authorize(Roles = "Admin")]

public async Task<IActionResult> Details(int? id)

{

if (id == null || \_context.RecipientAddresses == null)

{

return NotFound();

}

var recipientAddress = await \_context.RecipientAddresses

.FirstOrDefaultAsync(m => m.Id == id);

if (recipientAddress == null)

{

return NotFound();

}

return View(recipientAddress);

}

public IActionResult Create()

{

return View();

}

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<IActionResult> Create([Bind("Id,Street,House,Apartment")] RecipientAddress recipientAddress)

{

if (ModelState.IsValid)

{

\_context.Add(recipientAddress);

\_logger.LogInformation($"Add new recipient address ({recipientAddress.FulAddress})");

int result = await \_context.SaveChangesAsync();

if(result == 1)

{

if (Request.Cookies.TryGetValue("UserCredentials", out string credentialsJson))

{

var credentials = JsonConvert.DeserializeAnonymousType(credentialsJson, new { Email = "",

Password = "",

Address = new RecipientAddress() });

var updatedCredentials = new { Email = credentials.Email, Password = credentials.Password, Address = recipientAddress };

var updatedCredentialsJson = JsonConvert.SerializeObject(updatedCredentials);

Response.Cookies.Append("UserCredentials", updatedCredentialsJson);

}

return RedirectToAction("Register", "Recipients");

}

}

return View(recipientAddress);

}

[Authorize(Roles = "Admin")]

public async Task<IActionResult> Edit(int? id)

{

if (id == null || \_context.RecipientAddresses == null)

{

return NotFound();

}

var recipientAddress = await \_context.RecipientAddresses.FindAsync(id);

if (recipientAddress == null)

{

return NotFound();

}

return View(recipientAddress);

}

[Authorize(Roles = "Admin")]

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<IActionResult> Edit(int id, [Bind("Id,Street,House,Apartment")] RecipientAddress recipientAddress)

{

if (id != recipientAddress.Id)

{

return NotFound();

}

if (ModelState.IsValid)

{

try

{

\_context.Update(recipientAddress);

\_logger.LogInformation($"Edit recipient address ({recipientAddress.FulAddress})");

await \_context.SaveChangesAsync();

}

catch (DbUpdateConcurrencyException)

{

if (!RecipientAddressExists(recipientAddress.Id))

{

return NotFound();

}

else

{

throw;

}

}

return RedirectToAction(nameof(Index));

}

return View(recipientAddress);

}

[Authorize(Roles = "Admin")]

public async Task<IActionResult> Delete(int? id)

{

if (id == null || \_context.RecipientAddresses == null)

{

return NotFound();

}

var recipientAddress = await \_context.RecipientAddresses

.FirstOrDefaultAsync(m => m.Id == id);

if (recipientAddress == null)

{

return NotFound();

}

return View(recipientAddress);

}

[Authorize(Roles = "Admin")]

[HttpPost, ActionName("Delete")]

[ValidateAntiForgeryToken]

public async Task<IActionResult> DeleteConfirmed(int id)

{

if (\_context.RecipientAddresses == null)

{

return Problem("Entity set 'PostCityContext.RecipientAddresses' is null.");

}

var recipientAddress = await \_context.RecipientAddresses.FindAsync(id);

if (recipientAddress != null)

{

\_context.RecipientAddresses.Remove(recipientAddress);

\_logger.LogInformation($"Delete recipient address ({recipientAddress.FulAddress})");

}

await \_context.SaveChangesAsync();

return RedirectToAction(nameof(Index));

}

private bool RecipientAddressExists(int id)

{

return (\_context.RecipientAddresses?.Any(e => e.Id == id)).GetValueOrDefault();

}

}

}

Листинг класса *RecipientsController*.*cs*:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using Microsoft.AspNetCore.Mvc;

using Microsoft.AspNetCore.Mvc.Rendering;

using Microsoft.EntityFrameworkCore;

using Newtonsoft.Json;

using Microsoft.AspNetCore.Identity;

using Microsoft.AspNetCore.Authorization;

using PostCity.Controllers;

using Domains.Models;

using Domains.ViewModels.Sort;

using Repository.Models;

using Service.Data.Cache;

using Service.Data.Cookies;

using Utils;

using Domains.ViewModels.Filters.FilterModel;

using Domains.ViewModels;

using Domains.Models.Identity;

namespace PostCity.Controllers

{

public class RecipientsController : Controller, ISortOrderController<Recipient, RecipientSortState>

{

private readonly PostCityContext \_context;

private readonly RecipientCache \_cache;

private readonly CookiesManeger \_cookies;

private readonly FilterBy<Recipient> \_filter;

private readonly UserRegistrationManager \_userRegistrationManager;

private readonly UserManager<PostCityUser> \_userManager;

private readonly CacheUpdater \_cacheUpdater;

private readonly SessionLogger \_logger;

public RecipientsController(PostCityContext context,

RecipientCache recipientCache,

CookiesManeger cookiesManeger,

FilterBy<Recipient> filterBy,

UserRegistrationManager userRegistrationManager,

UserManager<PostCityUser> userManager,

UserCache userCache,

CacheUpdater cacheUpdater,

SessionLogger logger)

{

\_context = context;

\_cache = recipientCache;

\_cookies = cookiesManeger;

\_filter = filterBy;

\_userRegistrationManager = userRegistrationManager;

\_userManager = userManager;

\_cacheUpdater = cacheUpdater;

\_logger = logger;

}

[Authorize(Roles = "Admin")]

public async Task<IActionResult> Index(RecipientSortState sortOrder = RecipientSortState.StandardState, int page = 1)

{

var postCityContext = \_cache.Get();

RecipientFilterModel filterData = \_cookies.GetFromCookies<RecipientFilterModel>(Request.Cookies, "RecipientFilterData");

SetSortOrderViewData(sortOrder);

postCityContext = ApplySortOrder(postCityContext, sortOrder);

int pageSize = 10;

\_cache.Set(postCityContext);

var pageViewModel = new PageViewModel<Recipient, RecipientFilterModel>(postCityContext, page, pageSize, filterData);

return View(pageViewModel);

}

[Authorize(Roles = "Admin")]

[HttpPost]

public async Task<IActionResult> Index(RecipientFilterModel filterData, int page = 1)

{

\_cache.Update();

\_cookies.SaveToCookies(Response.Cookies, "RecipientFilterData", filterData);

var data = \_cache.Get();

data = \_filter.FilterByString(data, n => n.Name, filterData.Name);

data = \_filter.FilterByString(data, s => s.Surname, filterData.Surname);

data = \_filter.FilterByString(data, m => m.Middlename, filterData.Middlename);

data = \_filter.FilterByString(data, m => m.MobilePhone, filterData.MobilePhone);

data = \_filter.FilterByString(data, a => a.Address.FulAddress, filterData.Address);

int pageSize = 10;

\_cache.Set(data);

var pageViewModel = new PageViewModel<Recipient, RecipientFilterModel>(data, page, pageSize, filterData);

return View(pageViewModel);

}

[Authorize(Roles = "Admin")]

public async Task<IActionResult> Details(int? id)

{

if (id == null || \_context.Recipients == null)

{

return NotFound();

}

var recipient = \_cache.Get().FirstOrDefault(m => m.Id == id);

if (recipient == null)

{

return NotFound();

}

return View(recipient);

}

public IActionResult Register()

{

if (Request.Cookies.TryGetValue("RecipientFields", out string recipientFieldsJson))

{

var recipientFields = JsonConvert.DeserializeAnonymousType(recipientFieldsJson, new

{

Name = "",

Middlename = "",

Surname = "",

MobilePhone = ""

});

ViewData["Name"] = recipientFields.Name;

ViewData["Middlename"] = recipientFields.Middlename;

ViewData["Surname"] = recipientFields.Surname;

ViewData["MobilePhone"] = recipientFields.MobilePhone;

}

if (Request.Cookies.TryGetValue("UserCredentials", out string userCredentialsJson))

{

var userCredentials = JsonConvert.DeserializeAnonymousType(userCredentialsJson, new

{

Email = "",

Password = "",

Address = new RecipientAddress()

});

if (userCredentials.Address.Street != null)

{

ViewData["Address"] = userCredentials.Address.FulAddress;

}

}

return View();

}

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<IActionResult> SaveFields([Bind("Id,Name,Middlename,Surname, MobilePhone")] Recipient recipient)

{

if (ModelState.IsValid)

{

var credentials = new { Name=recipient.Name,

Middlename=recipient.Middlename,

Surname=recipient.Surname,

MobilePhone = recipient.MobilePhone,

};

var credentialsJson = JsonConvert.SerializeObject(credentials);

Response.Cookies.Append("RecipientFields", credentialsJson);

return RedirectToAction("Create", "RecipientAddresses");

}

ViewData["AddressId"] = new SelectList(\_context.RecipientAddresses, "Id", "FulAddress", recipient.AddressId);

return View(recipient);

}

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<IActionResult> Register([Bind("Id,Name,Middlename,Surname, MobilePhone")] Recipient recipient)

{

if (ModelState.IsValid)

{

if (Request.Cookies.TryGetValue("UserCredentials", out string credentialsJson))

{

var credentials = JsonConvert.DeserializeAnonymousType(credentialsJson, new { Email = "",

Password = "",

Address = new RecipientAddress() });

using var transaction = await \_context.Database.BeginTransactionAsync();

if(credentials.Address.Street != null)

{

try

{

var newRecipient = new Recipient()

{

Name = recipient.Name,

Middlename = recipient.Middlename,

Surname = recipient.Surname,

MobilePhone = recipient.MobilePhone,

AddressId = credentials.Address.Id

};

\_context.Add(newRecipient);

await \_context.SaveChangesAsync();

var result = await \_userRegistrationManager.RegisterUserWithRole(new PostCityUserModel()

{

Role = "Recipient",

Email = credentials.Email,

Password = credentials.Password,

UserId = newRecipient.Id

});

if (result.Succeeded)

{

transaction.Commit();

\_cacheUpdater.Update(\_cache);

Response.Cookies.Delete("RecipientFields");

Response.Cookies.Delete("UserCredentials");

\_logger.LogInformation($"Add new recipient ({newRecipient.FullName})");

return Redirect("/Identity/Account/Login");

}

else

{

transaction.Rollback();

\_context.Recipients.Remove(recipient);

\_context.RecipientAddresses.Remove(credentials.Address);

foreach (var error in result.Errors)

{

ViewData["AddressId"] = new SelectList(\_context.RecipientAddresses, "Id", "FulAddress", recipient.AddressId);

return View(recipient);

}

}

}

catch (Exception)

{

transaction.Rollback();

\_context.Recipients.Remove(recipient);

\_context.RecipientAddresses.Remove(credentials.Address);

}

}

else

{

ViewData["ErrorMessage"] = "Address is required. Please add an address before registering.";

return View(recipient);

}

}

}

ViewData["AddressId"] = new SelectList(\_context.RecipientAddresses, "Id", "FulAddress", recipient.AddressId);

return View(recipient);

}

[Authorize(Roles = "Admin")]

public async Task<IActionResult> Edit(int? id)

{

if (id == null || \_context.Recipients == null)

{

return NotFound();

}

var recipient = await \_context.Recipients.FindAsync(id);

if (recipient == null)

{

return NotFound();

}

ViewData["AddressId"] = new SelectList(\_context.RecipientAddresses, "Id", "FulAddress", recipient.AddressId);

return View(recipient);

}

[Authorize(Roles = "Admin")]

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<IActionResult> Edit(int id, [Bind("Id,Name,Middlename,Surname,AddressId,MobilePhone")] Recipient recipient)

{

if (id != recipient.Id)

{

return NotFound();

}

if (ModelState.IsValid)

{

try

{

await \_context.SaveChangesAsync();

\_cacheUpdater.Update(\_cache);

\_logger.LogInformation($"Edit recipient ({recipient.FullName})");

}

catch (DbUpdateConcurrencyException)

{

if (!RecipientExists(recipient.Id))

{

return NotFound();

}

else

{

throw;

}

}

return RedirectToAction(nameof(Index));

}

ViewData["AddressId"] = new SelectList(\_context.RecipientAddresses, "Id", "FulAddress", recipient.AddressId);

return View(recipient);

}

[Authorize(Roles = "Admin")]

public async Task<IActionResult> Delete(int? id)

{

if (id == null || \_context.Recipients == null)

{

return NotFound();

}

var recipient = \_cache.Get().FirstOrDefault(m => m.Id == id);

if (recipient == null)

{

return NotFound();

}

return View(recipient);

}

[Authorize(Roles = "Admin")]

[HttpPost, ActionName("Delete")]

[ValidateAntiForgeryToken]

public async Task<IActionResult> DeleteConfirmed(int id)

{

if (\_context.Recipients == null)

{

return Problem("Entity set 'SubsCityContext.Recipients' is null.");

}

var recipient = await \_context.Recipients.FindAsync(id);

if (recipient != null)

{

\_context.Recipients.Remove(recipient);

\_logger.LogInformation($"Delete recipient ({recipient.FullName})");

}

int isDelete = await \_context.SaveChangesAsync();

if (isDelete == 1)

{

PostCityUser user = \_context.FindUserByUserId(recipient.Id);

IdentityResult result = await \_userManager.DeleteAsync(user);

}

\_cacheUpdater.Update(\_cache);

return RedirectToAction(nameof(Index));

}

private bool RecipientExists(int id)

{

return (\_context.Recipients?.Any(e => e.Id == id)).GetValueOrDefault();

}

public void SetSortOrderViewData(RecipientSortState sortOrder)

{

ViewData["NameSort"] = sortOrder == RecipientSortState.NameAsc

? RecipientSortState.NameDesc

: RecipientSortState.NameAsc;

ViewData["MiddlenameSort"] = sortOrder == RecipientSortState.MiddlenameAsc

? RecipientSortState.MiddlenameDesc

: RecipientSortState.MiddlenameAsc;

ViewData["SurnameSort"] = sortOrder == RecipientSortState.SurnameAsc

? RecipientSortState.SurnameDesc

: RecipientSortState.SurnameAsc;

ViewData["AddressSort"] = sortOrder == RecipientSortState.AddressAsc

? RecipientSortState.AddressDesc

: RecipientSortState.AddressAsc;

}

public IEnumerable<Recipient> ApplySortOrder(IEnumerable<Recipient> postCityContext, RecipientSortState sortOrder)

{

return sortOrder switch

{

RecipientSortState.NameAsc => postCityContext.OrderByDescending(n => n.Name),

RecipientSortState.NameDesc => postCityContext.OrderBy(n => n.Name),

RecipientSortState.MiddlenameAsc => postCityContext.OrderByDescending(m => m.Middlename),

RecipientSortState.MiddlenameDesc => postCityContext.OrderBy(m => m.Middlename),

RecipientSortState.SurnameAsc => postCityContext.OrderByDescending(s => s.Surname),

RecipientSortState.SurnameDesc => postCityContext.OrderBy(s => s.Surname),

RecipientSortState.AddressAsc => postCityContext.OrderByDescending(a => a.Address.Street),

RecipientSortState.AddressDesc => postCityContext.OrderBy(a => a.Address.Street),

RecipientSortState.StandardState => postCityContext

};

}

}

}

Листинг класса *SubscriptionsController*.*cs*:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using Microsoft.AspNetCore.Mvc;

using Microsoft.AspNetCore.Mvc.Rendering;

using Microsoft.EntityFrameworkCore;

using Microsoft.AspNetCore.Authorization;

using System.Data;

using Newtonsoft.Json.Linq;

using Domains.Models;

using Domains.ViewModels.Sort;

using Repository.Models;

using Service.Data.Cache;

using Service.Data.Cookies;

using Utils;

using Domains.ViewModels;

using Domains.ViewModels.Filters.FilterModel;

namespace PostCity.Controllers

{

public class SubscriptionsController : Controller, ISortOrderController<Subscription, SubscriptionSortState>

{

private readonly PostCityContext \_context;

private readonly SubscriptionCache \_cache;

private readonly CookiesManeger \_cookies;

private readonly FilterBy<Subscription> \_filter;

private readonly SessionLogger \_logger;

public SubscriptionsController(PostCityContext context,

SubscriptionCache cache,

CookiesManeger cookiesManeger,

FilterBy<Subscription> filter,

SessionLogger logger)

{

\_context = context;

\_cache = cache;

\_cookies = cookiesManeger;

\_filter = filter;

\_logger = logger;

}

// GET: Subscriptions

public IActionResult Index(SubscriptionSortState sortOrder = SubscriptionSortState.StandardState, int page = 1)

{

var postCityContext = \_cache.Get();

SubscriptionFilterModel filterData = \_cookies.GetFromCookies<SubscriptionFilterModel>(Request.Cookies, "SubscriptionFilterData");

SetSortOrderViewData(sortOrder);

postCityContext = ApplySortOrder(postCityContext, sortOrder);

int pageSize = 10;

\_cache.Set(postCityContext);

var pageViewModel = new PageViewModel<Subscription, SubscriptionFilterModel>(postCityContext, page, pageSize, filterData);

return View(pageViewModel);

}

[HttpPost]

public async Task<IActionResult> Index(SubscriptionFilterModel filterData, int page = 1)

{

\_cache.Update();

\_cookies.SaveToCookies(Response.Cookies, "SubscriptionFilterData", filterData);

IEnumerable<Subscription> data = \_cache.Get();

data = \_filter.FilterByInt(data, d => d.Duration, filterData.Duration);

data = \_filter.FilterByDate(data, sb => sb.SubscriptionStartDate, filterData.StartDate);

data = \_filter.FilterByString(data, pn => pn.Office.StreetName, filterData.OfficeName);

data = \_filter.FilterByString(data, pn => pn.Publication.Name, filterData.PublicationName);

data = \_filter.FilterByString(data, pn => pn.Recipient.FullName, filterData.RecipientName);

data = \_filter.FilterByPeriod(data, pn => pn.SubscriptionStartDate, filterData.StartPeriod, filterData.EndPeriod);

var publicationPriceSum = data.Sum(subscription => subscription.Publication.Price);

int pageSize = 10;

\_cache.Set(data);

var pageViewModel = new PageViewModel<Subscription, SubscriptionFilterModel>(data, page, pageSize, filterData);

pageViewModel.Info = GetInfo(filterData, publicationPriceSum);

return View(pageViewModel);

}

public string GetInfo(SubscriptionFilterModel filterData, decimal publicationPriceSum)

{

string info = "Total publication price ";

if(filterData.StartPeriod != null && filterData.EndPeriod != null)

{

info += $"for the period from {filterData.StartPeriod.Value.ToString("dd.MM.yyyy")} to {filterData.EndPeriod.Value.ToString("dd.MM.yyyy")} ";

}

else if(filterData.StartPeriod != null && filterData.EndPeriod == null)

{

info += $"starting from {filterData.StartPeriod.Value.ToString("dd.MM.yyyy")} ";

}

else if (filterData.StartPeriod == null && filterData.EndPeriod != null)

{

info += $"ending from {filterData.EndPeriod.Value.ToString("dd.MM.yyyy")} ";

}

info += $"is {publicationPriceSum} руб ";

if(filterData.OfficeName != null)

{

info += $"for an office '{filterData.OfficeName}'";

}

info += ".";

return info;

}

public async Task<IActionResult> Details(int? id)

{

if (id == null || \_context.Subscriptions == null)

{

return NotFound();

}

var subscription = \_cache.Get().FirstOrDefault(m => m.Id == id);

if (subscription == null)

{

return NotFound();

}

return View(subscription);

}

[Authorize(Roles = "Admin")]

public IActionResult Create()

{

ViewData["EmployeeId"] = new SelectList(\_context.Employees, "Id", "FullName");

ViewData["OfficeId"] = new SelectList(\_context.Offices, "Id", "StreetName");

ViewData["PublicationId"] = new SelectList(\_context.Publications, "Id", "Name");

ViewData["RecipientId"] = new SelectList(\_context.Recipients, "Id", "FullName");

return View();

}

[Authorize(Roles = "Admin")]

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<IActionResult> Create([Bind("Id,RecipientId,PublicationId,Duration,OfficeId,SubscriptionStartDate")] Subscription subscription)

{

if (ModelState.IsValid)

{

\_context.Add(subscription);

await \_context.SaveChangesAsync();

\_cache.Update();

\_logger.LogInformation($"Add new subscription ({subscription.Publication.Name} / {subscription.Duration} мес.)");

return RedirectToAction(nameof(Index));

}

ViewData["OfficeId"] = new SelectList(\_context.Offices, "Id", "StreetName", subscription.OfficeId);

ViewData["PublicationId"] = new SelectList(\_context.Publications, "Id", "Name", subscription.PublicationId);

ViewData["RecipientId"] = new SelectList(\_context.Recipients, "Id", "FullName", subscription.RecipientId);

return View(subscription);

}

[Authorize(Roles = "Admin")]

public async Task<IActionResult> Edit(int? id)

{

if (id == null || \_context.Subscriptions == null)

{

return NotFound();

}

var subscription = await \_context.Subscriptions.FindAsync(id);

if (subscription == null)

{

return NotFound();

}

ViewData["OfficeId"] = new SelectList(\_context.Offices, "Id", "StreetName", subscription.OfficeId);

ViewData["PublicationId"] = new SelectList(\_context.Publications, "Id", "Name", subscription.PublicationId);

ViewData["RecipientId"] = new SelectList(\_context.Recipients, "Id", "FullName", subscription.RecipientId);

return View(subscription);

}

[HttpPost]

[ValidateAntiForgeryToken]

[Authorize(Roles = "Admin")]

public async Task<IActionResult> Edit(int id, [Bind("Id,RecipientId,PublicationId,Duration,OfficeId,SubscriptionStartDate")] Subscription subscription)

{

if (id != subscription.Id)

{

return NotFound();

}

if (ModelState.IsValid)

{

try

{

\_context.Update(subscription);

await \_context.SaveChangesAsync();

\_cache.Update();

\_logger.LogInformation($"Edit subscription ({subscription.Publication.Name} / {subscription.Duration} мес.)");

}

catch (DbUpdateConcurrencyException)

{

if (!SubscriptionExists(subscription.Id))

{

return NotFound();

}

else

{

throw;

}

}

return RedirectToAction(nameof(Index));

}

ViewData["OfficeId"] = new SelectList(\_context.Offices, "Id", "StreetName", subscription.OfficeId);

ViewData["PublicationId"] = new SelectList(\_context.Publications, "Id", "Name", subscription.PublicationId);

ViewData["RecipientId"] = new SelectList(\_context.Recipients, "Id", "Email", subscription.RecipientId);

return View(subscription);

}

[Authorize(Roles = "Admin")]

public async Task<IActionResult> Delete(int? id)

{

if (id == null || \_context.Subscriptions == null)

{

return NotFound();

}

var subscription = \_cache.Get().FirstOrDefault(m => m.Id == id);

if (subscription == null)

{

return NotFound();

}

return View(subscription);

}

[Authorize(Roles = "Admin")]

[HttpPost, ActionName("Delete")]

[ValidateAntiForgeryToken]

public async Task<IActionResult> DeleteConfirmed(int id)

{

if (\_context.Subscriptions == null)

{

return Problem("Entity set 'PostCityContext.Subscriptions' is null.");

}

var subscription = await \_context.Subscriptions.Include(s => s.Publication).FirstOrDefaultAsync(s => s.Id == id);

if (subscription != null)

{

\_logger.LogInformation($"Delete subscription ({subscription.Publication.Name} / {subscription.Duration} мес.)");

\_context.Subscriptions.Remove(subscription);

}

await \_context.SaveChangesAsync();

\_cache.Update();

return RedirectToAction(nameof(Index));

}

private bool SubscriptionExists(int id)

{

return (\_context.Subscriptions?.Any(e => e.Id == id)).GetValueOrDefault();

}

public void SetSortOrderViewData(SubscriptionSortState sortOrder)

{

ViewData["DurationSort"] = sortOrder == SubscriptionSortState.DurationAsc

? SubscriptionSortState.DurationDesc

: SubscriptionSortState.DurationAsc;

ViewData["DateSort"] = sortOrder == SubscriptionSortState.DateAsc

? SubscriptionSortState.DateDesc

: SubscriptionSortState.DateAsc;

ViewData["OfficeSort"] = sortOrder == SubscriptionSortState.OfficeNameAsc

? SubscriptionSortState.OfficeNameDesc

: SubscriptionSortState.OfficeNameAsc;

ViewData["PublicationSort"] = sortOrder == SubscriptionSortState.PublicationNameAsc

? SubscriptionSortState.PublicationNameDesc

: SubscriptionSortState.PublicationNameAsc;

}

public IEnumerable<Subscription> ApplySortOrder(IEnumerable<Subscription> postCityContext, SubscriptionSortState sortOrder)

{

return sortOrder switch

{

SubscriptionSortState.DurationDesc => postCityContext.OrderByDescending(d => d.Duration),

SubscriptionSortState.DurationAsc => postCityContext.OrderBy(d => d.Duration),

SubscriptionSortState.DateDesc => postCityContext.OrderByDescending(d => d.SubscriptionStartDate),

SubscriptionSortState.DateAsc => postCityContext.OrderBy(d => d.SubscriptionStartDate),

SubscriptionSortState.OfficeNameDesc => postCityContext.OrderByDescending(o => o.Office.StreetName),

SubscriptionSortState.OfficeNameAsc => postCityContext.OrderBy(o => o.Office.StreetName),

SubscriptionSortState.PublicationNameDesc => postCityContext.OrderByDescending(n => n.Publication.Name),

SubscriptionSortState.PublicationNameAsc => postCityContext.OrderBy(n => n.Publication.Name),

SubscriptionSortState.StandardState => postCityContext

};

}

}

}

Листинг класса *UserController*.*cs*:

using Domains.Models;

using Domains.ViewModels;

using Domains.ViewModels.Filters.FilterModel;

using Domains.ViewModels.Sort;

using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.Identity;

using Microsoft.AspNetCore.Mvc;

using PostCity.Controllers;

using Repository.Models;

using Service.Data.Cache;

using Service.Data.Cookies;

using Utils;

namespace PostCity.Controllers

{

[Authorize(Roles = "Admin")]

public class UserController : Controller, ISortOrderController<UserViewModel, UserSortState>

{

private readonly RoleManager<IdentityRole> \_roleManager;

private readonly UserManager<PostCityUser> \_userManager;

private readonly UserCache \_cache;

private readonly PostCityContext \_context;

private readonly CookiesManeger \_cookies;

private readonly FilterBy<UserViewModel> \_filter;

private readonly CacheUpdater \_cacheUpdater;

private readonly SessionLogger \_logger;

public UserController(RoleManager<IdentityRole> roleManager,

UserManager<PostCityUser> userManager,

PostCityContext context,

UserCache cache,

CookiesManeger cookiesManeger,

FilterBy<UserViewModel> filter,

CacheUpdater cacheUpdater,

SessionLogger sessionLogger

)

{

\_roleManager = roleManager;

\_userManager = userManager;

\_context = context;

\_cache = cache;

\_cookies = cookiesManeger;

\_filter = filter;

\_cacheUpdater = cacheUpdater;

\_logger = sessionLogger;

}

public async Task<IActionResult> Index(UserSortState sortOrder = UserSortState.StandardState, int page = 1)

{

var users = \_cache.Get();

UserFilterModel filterData = \_cookies.GetFromCookies<UserFilterModel>(Request.Cookies, "UserFilterData");

SetSortOrderViewData(sortOrder);

users = ApplySortOrder(users, sortOrder);

int pageSize = 10;

\_cache.Set(users);

var pageViewModel = new PageViewModel<UserViewModel, UserFilterModel>(users, page, pageSize, filterData);

return View(pageViewModel);

}

[HttpPost]

public async Task<IActionResult> Index(UserFilterModel filterData, int page = 1)

{

\_cache.Update();

\_cookies.SaveToCookies(Response.Cookies, "UserFilterData", filterData);

var users = \_cache.Get();

users = \_filter.FilterByString(users, fn => fn.UserObj != null ? fn.UserObj.FullName : "", filterData.Name).ToList();

users = \_filter.FilterByString(users, e => e.Email, filterData.Email).ToList();

users = \_filter.FilterByString(users, r => r.Role, filterData.Role).ToList();

int pageSize = 10;

\_cache.Set(users);

var pageViewModel = new PageViewModel<UserViewModel, UserFilterModel>(users, page, pageSize, filterData);

return View(pageViewModel);

}

[HttpGet]

public IActionResult Create()

{

return View();

}

[HttpPost]

public async Task<IActionResult> Create(string name)

{

if (!string.IsNullOrEmpty(name))

{

IdentityResult result = await \_roleManager.CreateAsync(new IdentityRole(name));

if (result.Succeeded)

{

\_cacheUpdater.Update(\_cache);

\_logger.LogInformation($"Add new user ({name})");

return RedirectToAction("Index");

}

else

{

foreach (var error in result.Errors)

{

ModelState.AddModelError(string.Empty, error.Description);

}

}

}

return View(name);

}

[HttpPost]

public async Task<IActionResult> DeleteUser(string id)

{

PostCityUser user = await \_userManager.FindByIdAsync(id);

if (user != null)

{

var roles = await \_userManager.GetRolesAsync(user);

var userRole = roles.FirstOrDefault();

if (userRole == "Employee")

{

var employee = await \_context.Employees.FindAsync(user.UserId);

if (employee != null)

{

\_context.Employees.Remove(employee);

}

}

else if(userRole == "Recipient")

{

var recipient = await \_context.Recipients.FindAsync(user.UserId);

if (recipient != null)

{

\_context.Recipients.Remove(recipient);

}

}

int isDelete = await \_context.SaveChangesAsync();

if(isDelete == 1)

{

await \_userManager.DeleteAsync(user);

}

\_cacheUpdater.Update(\_cache);

\_logger.LogInformation($"Delete user ({user.Email})");

}

return RedirectToAction("Index");

}

public IActionResult Roles() => View(\_roleManager.Roles);

public void SetSortOrderViewData(UserSortState sortOrder)

{

ViewData["RoleSort"] = sortOrder == UserSortState.RoleAsc

? UserSortState.RoleDesc

: UserSortState.RoleAsc;

}

public IEnumerable<UserViewModel> ApplySortOrder(IEnumerable<UserViewModel> postCityContext, UserSortState sortOrder)

{

return sortOrder switch

{

UserSortState.RoleAsc => postCityContext.OrderByDescending(r => r.Role),

UserSortState.RoleDesc => postCityContext.OrderBy(r => r.Role),

UserSortState.StandardState => postCityContext,

};

}

}

}

Листинг класса *DatabaseSaveFilter*.*cs*:

using Microsoft.AspNetCore.Mvc.Filters;

using Service.Data.Cache;

namespace PostCity.Infrastructure.Filters

{

public class DatabaseSaveFilter: IActionFilter

{

private readonly SubscriptionCache \_appCache;

public DatabaseSaveFilter(SubscriptionCache appCache)

{

\_appCache = appCache;

}

public void OnActionExecuting(ActionExecutingContext context)

{

}

public void OnActionExecuted(ActionExecutedContext context)

{

\_appCache.Set();

}

}

}

Листинг класса *DbInitializerMiddleware*.*cs*:

using Domains.Models;

using Microsoft.AspNetCore.Identity;

using Repository.Data;

using Repository.Models;

namespace PostCity.Middleware

{

public class DbInitializerMiddleware

{

private readonly RequestDelegate \_next;

public DbInitializerMiddleware(RequestDelegate next)

{

\_next = next;

}

public Task Invoke(HttpContext httpContext, PostCityContext db,

RoleManager<IdentityRole> roleManager, UserManager<PostCityUser> userManager)

{

if (!(httpContext.Session.Keys.Contains("starting")))

{

DbInitializer dbInitializer = new DbInitializer(db, roleManager, userManager);

dbInitializer.InitializeDb();

httpContext.Session.SetString("starting", "Yes");

}

return \_next.Invoke(httpContext);

}

}

public static class DbInitializerMiddlewareExtensions

{

public static IApplicationBuilder UseDbInitializerMiddleware(this IApplicationBuilder builder)

{

return builder.UseMiddleware<DbInitializerMiddleware>();

}

}

}

Листинг класса *DbInitializer*.*cs*:

using Domains.Models;

using Microsoft.AspNetCore.Identity;

using Microsoft.EntityFrameworkCore;

using Repository.Models;

using System.Globalization;

namespace Repository.Data

{

public class DbInitializer

{

private readonly PostCityContext \_context;

private readonly Random \_random;

private readonly RoleManager<IdentityRole> \_roleManager;

private readonly UserManager<PostCityUser> \_userManager;

public DbInitializer(PostCityContext context,

RoleManager<IdentityRole> roleManager,

UserManager<PostCityUser> userManager)

{

\_context = context;

\_random = new Random();

\_roleManager = roleManager;

\_userManager = userManager;

}

public void InitializeDb()

{

if (!\_context.Database.CanConnect())

{

\_context.Database.Migrate();

}

if (!\_roleManager.Roles.Any())

{

CreateRoles().Wait();

}

if (!\_userManager.Users.Any())

{

CreateAdmin().Wait();

};

if (!\_context.PublicationTypes.Any())

{

InitializePublicationTypes();

}

if (!\_context.Publications.Any())

{

InitializePublications();

}

if (!\_context.RecipientAddresses.Any())

{

InitializeRecipientAddresses();

}

if (!\_context.Recipients.Any())

{

InitializeRecipients();

CreateRecipient().Wait();

}

if (!\_context.Offices.Any())

{

InitializeOffices();

}

if (!\_context.EmployeePositions.Any())

{

InitializeEmployeePositions();

}

if (!\_context.Employees.Any())

{

InitializeEmployees();

CreateEmployees().Wait();

}

if (!\_context.Subscriptions.Any())

{

InitializeSubscriptions();

}

\_context.SaveChanges();

}

string[] lastNames = {

"Иванов",

"Петров",

"Смирнов",

"Сидоров",

"Козлов",

"Морозов",

"Васильев",

};

string[] firstNames = {

"Александр",

"Иван",

"Михаил",

"Евгений",

"Сергей",

"Дмитрий"

};

string[] middleNames = {

"Иванович",

"Александрович",

"Петрович",

"Сергеевич",

"Владимирович",

"Игоревна",

"Дмитриевич"

};

string[] streetNames = {

"Пролетарская улица",

"Ленинская улица",

"Гагарина улица",

"Советская улица",

"Пушкинская улица",

"Московская улица",

"Кировская улица",

"Парковая улица",

"Садовая улица",

"Комсомольская улица",

"Школьная улица",

"Жукова улица",

"Мичурина улица",

"Свердлова улица",

"Октябрьская улица",

"Горького улица"

};

private void InitializePublicationTypes()

{

if (!\_context.PublicationTypes.Any())

{

\_context.PublicationTypes.AddRange(

new PublicationType { Type = "газета" },

new PublicationType { Type = "журнал" }

);

\_context.SaveChanges();

}

}

private void InitializePublications()

{

if (!\_context.Publications.Any())

{

var publicationNames = new List<string>

{

"Утренние новости",

"Вечерний вестник",

"Асахи симбун",

"Комсомольская правда",

"Взгляд"

};

var publicationTypes = \_context.PublicationTypes.ToList();

for (int i = 0; i <= 100; i++)

{

var randomType = publicationTypes[\_random.Next(publicationTypes.Count())];

var randomName = publicationNames[\_random.Next(publicationNames.Count())];

var price = (decimal)(\_random.NextDouble() \* 100);

\_context.Publications.Add(new Publication

{

Type = randomType,

Name = randomName,

Price = price

});

}

\_context.SaveChanges();

}

}

private void InitializeEmployeePositions()

{

if (!\_context.EmployeePositions.Any())

{

\_context.EmployeePositions.AddRange(

new EmployeePosition { Position = "Почтальон" },

new EmployeePosition { Position = "Кассир" },

new EmployeePosition { Position = "Сортировщик" },

new EmployeePosition { Position = "Менеджер по доставке" }

);

\_context.SaveChanges();

}

}

private void InitializeEmployees()

{

if (!\_context.Employees.Any())

{

var employeePositions = \_context.EmployeePositions.ToList();

var offices = \_context.Offices.ToList();

for (int i = 0; i <= 100; i++)

{

var randomPosition = employeePositions[\_random.Next(employeePositions.Count())];

var randomOffice = offices[\_random.Next(offices.Count())];

\_context.Employees.Add(new Employee

{

Name = firstNames[\_random.Next(firstNames.Length)],

Middlename = middleNames[\_random.Next(middleNames.Length)],

Surname = lastNames[\_random.Next(lastNames.Length)],

Position = randomPosition,

Office = randomOffice

});

}

\_context.SaveChanges();

}

}

private void InitializeOffices()

{

if (!\_context.Offices.Any())

{

for (int i = 0; i <= 100; i++)

{

var randomStreet = streetNames[\_random.Next(streetNames.Length)];

var house = \_random.Next(1, 101);

\_context.Offices.Add(new Office

{

OwnerName = firstNames[\_random.Next(firstNames.Length)],

OwnerMiddlename = middleNames[\_random.Next(middleNames.Length)],

OwnerSurname = lastNames[\_random.Next(lastNames.Length)],

StreetName = $"{randomStreet}. д {house}",

MobilePhone = $"+375{\_random.Next(100000000, 999999999)}",

Email = $"{Guid.NewGuid().ToString().Substring(0, 10)}@gmail.com"

});

}

\_context.SaveChanges();

}

}

private void InitializeRecipientAddresses()

{

if (!\_context.RecipientAddresses.Any())

{

for (int i = 0; i <= 100; i++)

{

var randomStreet = streetNames[\_random.Next(streetNames.Length)];

var house = \_random.Next(1, 101);

var apartment = \_random.Next(1, 21);

\_context.RecipientAddresses.Add(new RecipientAddress

{

Street = randomStreet,

House = house,

Apartment = apartment

});

}

\_context.SaveChanges();

}

}

private void InitializeRecipients()

{

if (!\_context.Recipients.Any())

{

var recipientAddresses = \_context.RecipientAddresses.ToList();

for (int i = 0; i <= 100; i++)

{

var randomAddress = recipientAddresses[\_random.Next(recipientAddresses.Count())];

\_context.Recipients.Add(new Recipient

{

Name = firstNames[\_random.Next(firstNames.Length)],

Middlename = middleNames[\_random.Next(middleNames.Length)],

Surname = lastNames[\_random.Next(lastNames.Length)],

Address = randomAddress,

MobilePhone = $"+375{\_random.Next(100000000, 999999999)}",

});

}

\_context.SaveChanges();

}

}

private void InitializeSubscriptions()

{

if (!\_context.Subscriptions.Any())

{

var recipients = \_context.Recipients.ToList();

var publications = \_context.Publications.ToList();

var offices = \_context.Offices.ToList();

var employees = \_context.Employees.ToList();

for (int i = 0; i < 100; i++)

{

var randomRecipient = recipients[\_random.Next(recipients.Count())];

var randomPublication = publications[\_random.Next(publications.Count())];

var randomEmployee = employees[\_random.Next(employees.Count())];

var randomOffice = offices[\_random.Next(offices.Count())];

var duration = \_random.Next(1, 13);

var dateStr = $"{\_random.Next(1, 13).ToString().PadLeft(2, '0')}.{\_random.Next(2014, 2024)}";

DateTime startDate;

if (DateTime.TryParseExact(dateStr, "MM.yyyy", CultureInfo.InvariantCulture, DateTimeStyles.None, out startDate))

{

\_context.Subscriptions.Add(new Subscription

{

Recipient = randomRecipient,

Publication = randomPublication,

Duration = duration,

Office = randomOffice,

SubscriptionStartDate = startDate

});

}

}

\_context.SaveChanges();

}

}

//Identity

private async Task CreateRoles()

{

string[] roleNames = { "Admin", "Employee", "Recipient" };

foreach (var roleName in roleNames)

{

var roleExist = await \_roleManager.RoleExistsAsync(roleName);

if (!roleExist)

{

IdentityResult result = await \_roleManager.CreateAsync(new IdentityRole(roleName));

}

}

}

private async Task CreateAdmin()

{

if (\_userManager.FindByNameAsync("admin@gmail.com").Result == null)

{

PostCityUser user = new PostCityUser

{

UserId = 1,

UserName = "admin@gmail.com",

Email = "admin@gmail.com"

};

IdentityResult result = await \_userManager.CreateAsync(user, "EQRu~ha+75hqIcr");

if (result.Succeeded)

{

await \_userManager.AddToRoleAsync(user, "Admin");

}

}

}

public async Task CreateEmployees()

{

var employeeToDelete = await \_userManager.GetUsersInRoleAsync("Employee");

foreach (var recipient in employeeToDelete)

{

var result = await \_userManager.DeleteAsync(recipient);

}

var employees = await \_context.Employees.ToListAsync();

foreach (var employee in employees)

{

var existingUser = await \_userManager.FindByEmailAsync($"Employee{employee.Id}@gmail.com");

if (existingUser == null)

{

PostCityUser user = new PostCityUser

{

UserId = employee.Id,

UserName = $"Employee{employee.Id}@gmail.com",

Email = $"Employee{employee.Id}@gmail.com"

};

var password = $"Employee\_{employee.Id}";

IdentityResult result = await \_userManager.CreateAsync(user, password);

if (result.Succeeded)

{

await \_userManager.AddToRoleAsync(user, "Employee");

}

}

}

}

public async Task CreateRecipient()

{

var recipientsToDelete = await \_userManager.GetUsersInRoleAsync("Recipient");

foreach (var recipient in recipientsToDelete)

{

var result = await \_userManager.DeleteAsync(recipient);

}

var recipients = await \_context.Recipients.ToListAsync();

foreach (var recipient in recipients)

{

var existingUser = await \_userManager.FindByEmailAsync($"Recipient{recipient.Id}@gmail.com");

if (existingUser == null)

{

PostCityUser user = new PostCityUser

{

UserId = recipient.Id,

UserName = $"Recipient{recipient.Id}@gmail.com",

Email = $"Recipient{recipient.Id}@gmail.com"

};

var password = $"Recipient\_{recipient.Id}";

IdentityResult result = await \_userManager.CreateAsync(user, password);

if (result.Succeeded)

{

await \_userManager.AddToRoleAsync(user, "Recipient");

}

}

}

}

}

}

Листинг класса *PostCityContext.cs*:

using Domains.Models;

using Microsoft.AspNetCore.Identity.EntityFrameworkCore;

using Microsoft.EntityFrameworkCore;

namespace Repository.Models;

public partial class PostCityContext : IdentityDbContext<PostCityUser>

{

public PostCityContext()

{

}

public PostCityContext(DbContextOptions<PostCityContext> options)

: base(options)

{

}

public virtual DbSet<Employee> Employees { get; set; }

public virtual DbSet<EmployeePosition> EmployeePositions { get; set; }

public virtual DbSet<Office> Offices { get; set; }

public virtual DbSet<Publication> Publications { get; set; }

public virtual DbSet<PublicationType> PublicationTypes { get; set; }

public virtual DbSet<Recipient> Recipients { get; set; }

public virtual DbSet<RecipientAddress> RecipientAddresses { get; set; }

public virtual DbSet<Subscription> Subscriptions { get; set; }

public virtual DbSet<PostCityUser> PostCityUsers { get; set; }

public PostCityUser FindUserByUserId(int userId)

{

return Users.FirstOrDefault(u => u.UserId == userId);

}

}

Листинг класса *AppCache*.*cs*:

using Microsoft.Extensions.Caching.Memory;

using Repository.Models;

namespace Service.Data.Cache

{

public abstract class AppCache<T> : IAppCache

{

protected readonly IMemoryCache \_cache;

protected readonly PostCityContext \_db;

internal readonly int \_saveTime = 2 \* 2 \* 240;

public AppCache(PostCityContext db, IMemoryCache memoryCache)

{

\_db = db;

\_cache = memoryCache;

}

public abstract IEnumerable<T> Get();

public abstract IEnumerable<T> Set();

public abstract IEnumerable<T> Set(IEnumerable<T> values);

public void Update()

{

Set();

}

}

}

Листинг класса *CacheUpdater.cs*:

namespace Service.Data.Cache

{

public class CacheUpdater

{

private EmployeeCache \_employeeCache;

private OfficeCache \_officeCache;

private PublicationCache \_publicationCache;

private RecipientCache \_recipientCache;

private SubscriptionCache \_subscriptionCache;

private UserCache \_userCache;

public CacheUpdater(RecipientCache recipientCache, SubscriptionCache subscriptionCache, UserCache userCache,

EmployeeCache employeeCache, PublicationCache publicationCache, OfficeCache officeCache)

{

\_recipientCache = recipientCache;

\_subscriptionCache = subscriptionCache;

\_userCache = userCache;

\_employeeCache = employeeCache;

\_officeCache = officeCache;

\_publicationCache = publicationCache;

}

public void Update(IAppCache cacheEntity)

{

cacheEntity.Update();

if (typeof(OfficeCache).IsAssignableFrom(cacheEntity.GetType()))

{

\_employeeCache.Update();

\_subscriptionCache.Update();

}

if (typeof(EmployeeCache).IsAssignableFrom(cacheEntity.GetType()))

{

\_officeCache.Update();

\_subscriptionCache.Update();

\_userCache.Update();

}

if (typeof(PublicationCache).IsAssignableFrom(cacheEntity.GetType()))

{

\_subscriptionCache.Update();

}

if (typeof(RecipientCache).IsAssignableFrom(cacheEntity.GetType()))

{

\_subscriptionCache.Update();

\_userCache.Update();

}

if (typeof(UserCache).IsAssignableFrom(cacheEntity.GetType()))

{

\_employeeCache.Update();

\_recipientCache.Update();

}

}

}

}

Листинг класса *EmployeeCache.cs*:

using Domains.Models;

using Microsoft.EntityFrameworkCore;

using Microsoft.Extensions.Caching.Memory;

using Repository.Models;

namespace Service.Data.Cache

{

public class EmployeeCache : AppCache<Employee>

{

public EmployeeCache(PostCityContext db, IMemoryCache memoryCache) : base(db, memoryCache)

{

}

public override IEnumerable<Employee> Get()

{

\_cache.TryGetValue("Employees", out IEnumerable<Employee>? emploees);

if (emploees is null)

{

emploees = Set();

}

return emploees;

}

public override IEnumerable<Employee> Set()

{

var emploees = \_db.Employees

.Include(e => e.Position)

.Include(e => e.Office)

.ToList();

\_cache.Set("Employees", emploees, new MemoryCacheEntryOptions().SetAbsoluteExpiration(TimeSpan.FromSeconds(\_saveTime)));

return emploees;

}

public override IEnumerable<Employee> Set(IEnumerable<Employee> emploees)

{

\_cache.Set("Employees", emploees, new MemoryCacheEntryOptions().SetAbsoluteExpiration(TimeSpan.FromSeconds(\_saveTime)));

return emploees;

}

}

}

Листинг класса *IAppCache.cs*:

namespace Service.Data.Cache

{

public interface IAppCache

{

public void Update();

}

}

Листинг класса *OfficeCache.cs*:

using Domains.Models;

using Microsoft.Extensions.Caching.Memory;

using Repository.Models;

namespace Service.Data.Cache

{

public class OfficeCache : AppCache<Office>

{

public OfficeCache(PostCityContext db, IMemoryCache memoryCache) : base(db, memoryCache)

{

}

public override IEnumerable<Office> Get()

{

\_cache.TryGetValue("Offices", out IEnumerable<Office>? offices);

if (offices is null)

{

offices = Set();

}

return offices;

}

public override IEnumerable<Office> Set()

{

var offices = \_db.Offices.ToList();

\_cache.Set("Offices", offices, new MemoryCacheEntryOptions().SetAbsoluteExpiration(TimeSpan.FromSeconds(\_saveTime)));

return offices;

}

public override IEnumerable<Office> Set(IEnumerable<Office> offices)

{

\_cache.Set("Offices", offices, new MemoryCacheEntryOptions().SetAbsoluteExpiration(TimeSpan.FromSeconds(\_saveTime)));

return offices;

}

}

}

Листинг класса *PublicationCache.cs*:

using Domains.Models;

using Microsoft.EntityFrameworkCore;

using Microsoft.Extensions.Caching.Memory;

using Repository.Models;

namespace Service.Data.Cache

{

public class PublicationCache : AppCache<Publication>

{

public PublicationCache(PostCityContext db, IMemoryCache memoryCache) : base(db, memoryCache)

{

}

public override IEnumerable<Publication> Get()

{

\_cache.TryGetValue("Publication", out IEnumerable<Publication>? publications);

if (publications is null)

{

publications = Set();

}

return publications;

}

public override IEnumerable<Publication> Set()

{

var subscriptions = \_db.Publications.Include(t => t.Type).ToList();

\_cache.Set("Publication", subscriptions, new MemoryCacheEntryOptions().SetAbsoluteExpiration(TimeSpan.FromSeconds(\_saveTime)));

return subscriptions;

}

public override IEnumerable<Publication> Set(IEnumerable<Publication> values)

{

\_cache.Set("Publication", values, new MemoryCacheEntryOptions().SetAbsoluteExpiration(TimeSpan.FromSeconds(\_saveTime)));

return values;

}

}

}

Листинг класса *RecipientCache.cs*:

using Domains.Models;

using Microsoft.EntityFrameworkCore;

using Microsoft.Extensions.Caching.Memory;

using Repository.Models;

namespace Service.Data.Cache

{

public class RecipientCache : AppCache<Recipient>

{

public RecipientCache(PostCityContext db, IMemoryCache memoryCache) : base(db, memoryCache)

{

}

public override IEnumerable<Recipient> Get()

{

\_cache.TryGetValue("Recipient", out IEnumerable<Recipient>? emploees);

if (emploees is null)

{

emploees = Set();

}

return emploees;

}

public override IEnumerable<Recipient> Set()

{

var recipients = \_db.Recipients.Include(a => a.Address).Include(s => s.Subscriptions).ThenInclude(s => s.Publication).ToList();

\_cache.Set("Recipient", recipients, new MemoryCacheEntryOptions().SetAbsoluteExpiration(TimeSpan.FromSeconds(\_saveTime)));

return recipients;

}

public override IEnumerable<Recipient> Set(IEnumerable<Recipient> values)

{

\_cache.Set("Recipient", values, new MemoryCacheEntryOptions().SetAbsoluteExpiration(TimeSpan.FromSeconds(\_saveTime)));

return values;

}

}

}

Листинг класса *SubscriptionCache.cs*:

using Domains.Models;

using Microsoft.EntityFrameworkCore;

using Microsoft.Extensions.Caching.Memory;

using Repository.Models;

namespace Service.Data.Cache

{

public class SubscriptionCache : AppCache<Subscription>

{

public SubscriptionCache(PostCityContext db, IMemoryCache memoryCache) : base(db, memoryCache)

{

}

public override IEnumerable<Subscription> Get()

{

\_cache.TryGetValue("Subscriptions", out IEnumerable<Subscription>? subscriptions);

if (subscriptions is null)

{

subscriptions = Set();

}

return subscriptions;

}

public override IEnumerable<Subscription> Set()

{

var subscriptions = \_db.Subscriptions.Include(s => s.Office)

.Include(s => s.Publication)

.Include(s => s.Recipient).ToList();

\_cache.Set("Subscriptions", subscriptions, new MemoryCacheEntryOptions().SetAbsoluteExpiration(TimeSpan.FromSeconds(\_saveTime)));

return subscriptions;

}

public override IEnumerable<Subscription> Set(IEnumerable<Subscription> subscriptions)

{

\_cache.Set("Subscriptions", subscriptions, new MemoryCacheEntryOptions().SetAbsoluteExpiration(TimeSpan.FromSeconds(\_saveTime)));

return subscriptions;

}

}

}

Листинг класса *UserCache.cs*:

using Domains.Models;

using Domains.ViewModels;

using Microsoft.AspNetCore.Identity;

using Microsoft.Extensions.Caching.Memory;

using Repository.Models;

namespace Service.Data.Cache

{

public class UserCache : IAppCache

{

private readonly UserManager<PostCityUser> \_userManager;

private readonly RoleManager<IdentityRole> \_roleManager;

private readonly int \_saveTime = 2 \* 2 \* 240;

private readonly PostCityContext \_context;

private readonly IMemoryCache \_cache;

public UserCache(PostCityContext db, IMemoryCache memoryCache,

UserManager<PostCityUser> userManager, RoleManager<IdentityRole> roleManager)

{

\_context = db;

\_cache = memoryCache;

\_userManager = userManager;

\_roleManager = roleManager;

\_roleManager = roleManager;

}

public IEnumerable<UserViewModel> Get()

{

\_cache.TryGetValue("Users", out IEnumerable<UserViewModel>? users);

if (users is null)

{

users = Set();

}

return users;

}

public IEnumerable<UserViewModel> Set()

{

var users = \_userManager.Users.ToList();

var usersList = new List<UserViewModel>();

foreach (var user in users)

{

var roles = \_userManager.GetRolesAsync(user).GetAwaiter().GetResult();

var roleToDisplay = roles.FirstOrDefault();

IUser userObj = null;

if (roleToDisplay == "Recipient")

{

userObj = \_context.Recipients.Find(user.UserId);

}

else if (roleToDisplay == "Employee")

{

userObj = \_context.Employees.Find(user.UserId);

}

usersList.Add(new UserViewModel()

{

Id = user.Id,

UserObj = userObj,

UserId = user.UserId,

Role = roleToDisplay,

Email = user.Email,

});

}

\_cache.Set("Users", usersList, new MemoryCacheEntryOptions().SetAbsoluteExpiration(TimeSpan.FromSeconds(\_saveTime)));

return usersList;

}

public IEnumerable<UserViewModel> Set(IEnumerable<UserViewModel> users)

{

\_cache.Set("Users", users, new MemoryCacheEntryOptions().SetAbsoluteExpiration(TimeSpan.FromSeconds(\_saveTime)));

return users;

}

public void Update()

{

Set();

}

}

}

Листинг класса *CookiesManeger.cs*:

using Microsoft.AspNetCore.Http;

using Newtonsoft.Json;

namespace Service.Data.Cookies

{

public class CookiesManeger

{

public void SaveToCookies<T>(IResponseCookies response,string key, T filterData)

{

string filterDataJson = JsonConvert.SerializeObject(filterData);

response.Append(key, filterDataJson);

}

public T GetFromCookies<T>(IRequestCookieCollection request, string key)

{

if (request.TryGetValue(key, out string filterValue))

{

T filterData = JsonConvert.DeserializeObject<T>(filterValue);

return filterData;

}

return default(T);

}

}

}

Листинг класса *FilterBy.cs*:

namespace Utils

{

public class FilterBy<T>

{

public IEnumerable<T> FilterByInt(

IEnumerable<T> data,

Func<T, int> propertySelector,

int? filterValue)

{

if(filterValue != null)

return data.Where(item => propertySelector(item) == filterValue);

else

{

return data;

}

}

public IEnumerable<T> FilterByDate(

IEnumerable<T> data,

Func<T, DateTime> propertySelector,

DateTime? filterDate)

{

if(filterDate != null)

{

return data.Where(item => propertySelector(item) == filterDate);

}

else return data;

}

public IEnumerable<T> FilterByString(

IEnumerable<T> data,

Func<T, string> propertySelector,

string filterValue)

{

return !string.IsNullOrEmpty(filterValue)

? data.Where(item => propertySelector(item).Contains(filterValue ?? ""))

: data;

}

public IEnumerable<T> FilterByDecimal(

IEnumerable<T> data,

Func<T, decimal> propertySelector,

decimal? filterValue,

decimal tolerance = 1)

{

if (filterValue != null)

{

return data.Where(item => Math.Abs(propertySelector(item) - filterValue.Value) <= tolerance);

}

else

{

return data;

}

}

public IEnumerable<T> FilterByPeriod(

IEnumerable<T> data,

Func<T, DateTime> propertySelector,

DateTime? startDate,

DateTime? endDate)

{

if (startDate != null && endDate != null)

{

return data.Where(item => propertySelector(item) >= startDate && propertySelector(item) <= endDate);

}

else if (startDate != null)

{

return data.Where(item => propertySelector(item) >= startDate);

}

else if (endDate != null)

{

return data.Where(item => propertySelector(item) <= endDate);

}

else

{

return data;

}

}

}

}

Листинг класса *UserRegistrationManager.cs*:

using Domains.Models;

using Domains.Models.Identity;

using Microsoft.AspNetCore.Identity;

namespace Utils

{

public class UserRegistrationManager

{

private readonly UserManager<PostCityUser> \_userManager;

private readonly RoleManager<IdentityRole> \_roleManager;

public UserRegistrationManager(UserManager<PostCityUser> userManager, RoleManager<IdentityRole> roleManager)

{

\_userManager = userManager;

\_roleManager = roleManager;

}

public async Task<IdentityResult> RegisterUserWithRole(PostCityUserModel postCityUserModel)

{

var roleExists = await \_roleManager.RoleExistsAsync(postCityUserModel.Role);

if (!roleExists)

{

await \_roleManager.CreateAsync(new IdentityRole(postCityUserModel.Role));

}

var user = new PostCityUser { UserName = postCityUserModel.Email,

Email = postCityUserModel.Email,

UserId = postCityUserModel.UserId };

var result = await \_userManager.CreateAsync(user, postCityUserModel.Password);

if (result.Succeeded)

{

await \_userManager.AddToRoleAsync(user, postCityUserModel.Role);

}

return result;

}

}

}

# ПРИЛОЖЕНИЕ Б

(обязательное)

**Чертёж структуры *web*-приложения**