**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 9](#_Toc152518241)

[2 Аппаратное и программное обеспечение информационной системы 14](#_Toc152518242)

[2.1 Требования к системному и прикладному программному обеспечению на стороне сервера хранилища данных 14](#_Toc152518243)

[2.2 Требования к системному и прикладному программному обеспечению на стороне *web*-сервера 14](#_Toc152518244)

[2.3 Действия системного администратора при настройке и развертывании *web*-приложения 15](#_Toc152518245)

[3 Структура приложения 17](#_Toc152518246)

[3.1 Описание общей структуры приложения 17](#_Toc152518247)

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

[3.3 Описание компонента *repository* 19](#_Toc152518249)

[3.4 Описание компонента *service* 20](#_Toc152518250)

[3.5 Описание компонента *web* 21](#_Toc152518251)

[4 Руководство пользователя 24](#_Toc152518252)

[4.1 Назначение, условие применения и функционал 24](#_Toc152518253)

[4.2 Графический интерфейс и его функционал 24](#_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 101](#_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

## 

## 1.1 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.

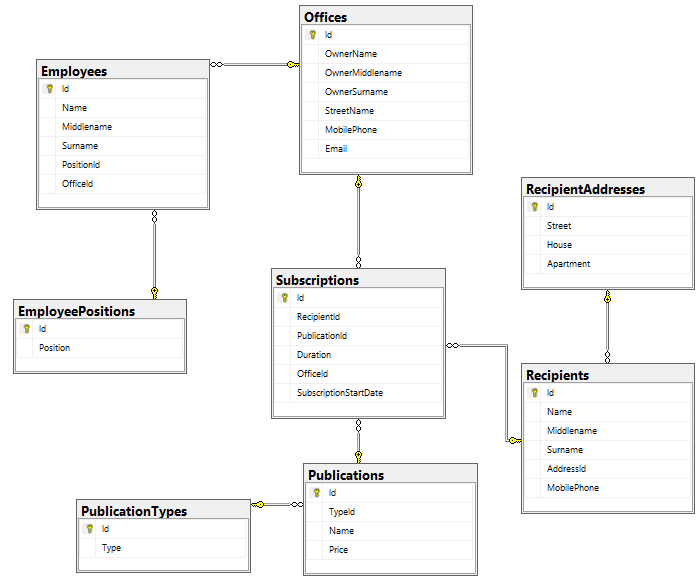


Рисунок 1.1 – Диаграмма базы данных

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

Данное приложение предназначено для управления страховой компанией. Приложение поддерживает две роли: клиента и страхового агента.

К основным функциям клиента относятся:

– Просмотр полисов, которые клиент оформил. Оформление новых полисов, которые представлены в списке. Отказ от оформленного полиса. Получения деталей по каждому из оформленных полисов.

– Просмотр, создание, удаление и редактирование страховых случаев для данного клиента.

– Получение подробной информации о текущем клиенте с возможностью редактирования информации о себе и смены пароля.

К основным функциям страхового агента относятся:

– Возможность просмотра, создания, редактирования, удаления и получения подробной информации для сущности тип страховки.

– Возможность просмотра, создания, редактирования, удаления и получения подробной информации для сущности контракт.

– Возможность просмотра, создания, редактирования, удаления и получения подробной информации для сущности подтверждающий документ.

– Возможность просмотра, создания, редактирования, удаления и получения подробной информации для сущности полис.

– Возможность просмотра, создания, редактирования, удаления и получения подробной информации для сущности страховой случай.

– Возможность просмотра, создания, редактирования, удаления и получения подробной информации для сущности страховой агент.

– Возможность просмотра, создания, редактирования, удаления и получения подробной информации для сущности клиент.

– Получение подробной информации о текущем страховом агенте с возможностью редактирования информации о себе и смены пароля.

## 4.2 Графический интерфейс и его функционал

При запуске приложение на экран выводится главная страница приложения. В зависимости от роли под которой зашел пользователь содержимое главной страницы будет отличатся. Если пользователь открыл сайт и не вошел в систему, то будет выведена кнопка зарегистрироваться на сайте. Если пользователь зашел в свой аккаунт, то будет выведено приветствие данного пользователя. Пример главной страницы сайта указан на рисунке 4.1.

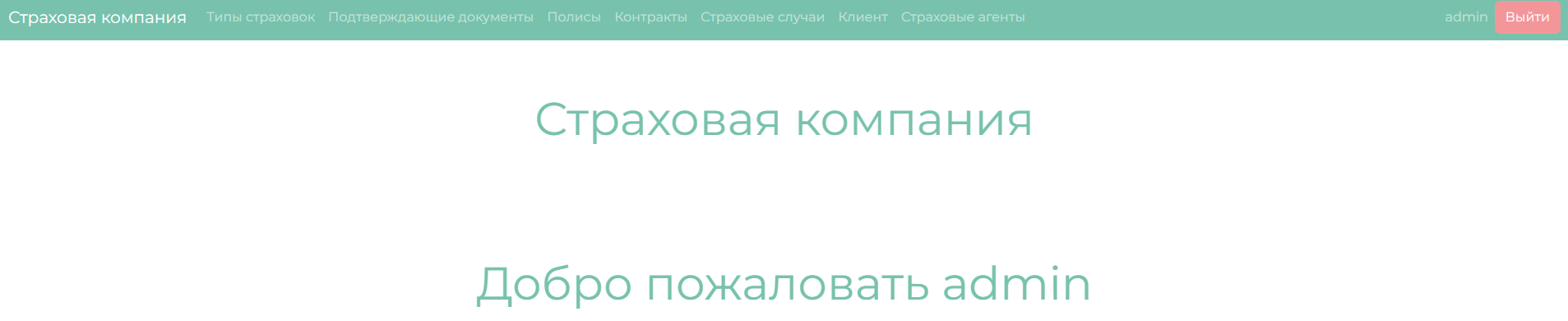


Рисунок 4.1 – Главная страница приложения

Для входа в систему была реализована страница, на которой пользователю необходимо ввести свои имя, фамилию, отчество и пароль. Пример страницы для входа в систему указан на рисунке 4.2

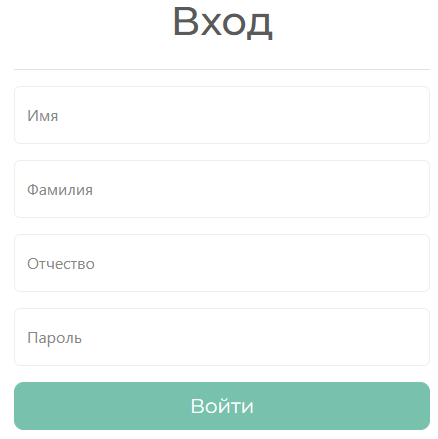


Рисунок 4.2 – Страница для входа в систему

Также на сайте была реализована страница для регистрации пользователя. На этой страницы пользователь должен ввести данные общие для любой роли: имя, фамилию, отчество и пароль. И если пользователь выбрал роль страхового агента ему необходимо выбрать тип агента (совместитель или штатный работник) и контракт, по которому он будет работать. Пример страницы регистрации для страхового агента указан на рисунке 4.3.

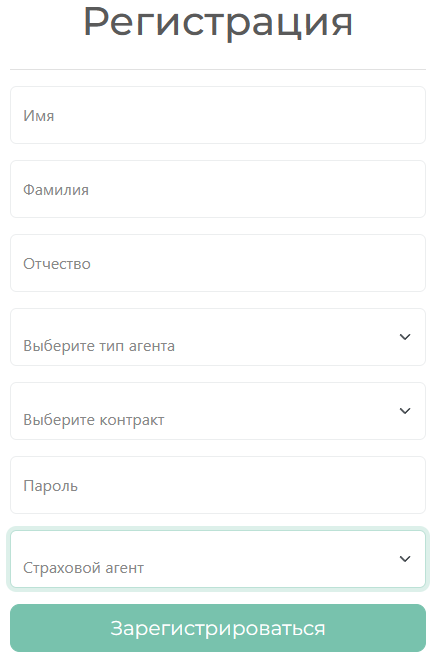


Рисунок 4.3 – Страница для регистрации страхового агента

Если пользователь при регистрации выбрал в качестве роль клиента ему необходимо будет ввести дополнительные данные: дата рождения, номер телефона, номер паспорта, адрес и дату выдачи паспорта. Пример страницы регистрации для клиента указан на рисунке 4.4.

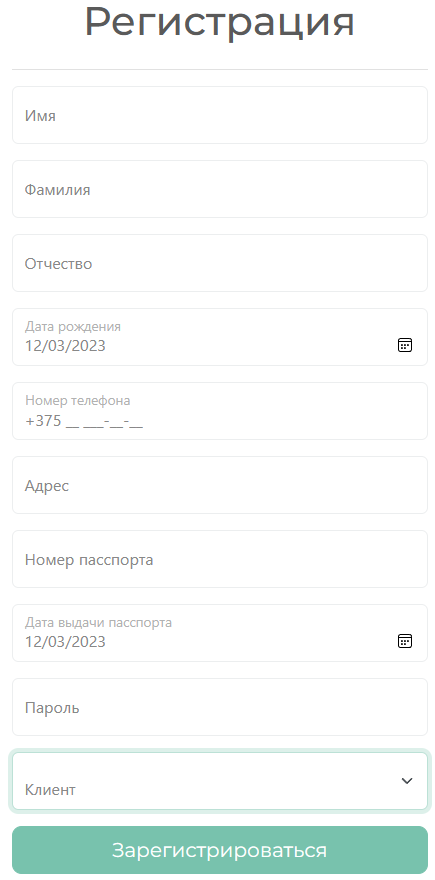


Рисунок 4.4 – Страница для регистрации клиента

Далее при входе в систему пользователь может редактировать и просматривать данные своего аккаунта. Редактировать свои данные и менять пароль. Также получать подробную информацию о себе. Пример данного меню указан на рисунке 4.5.

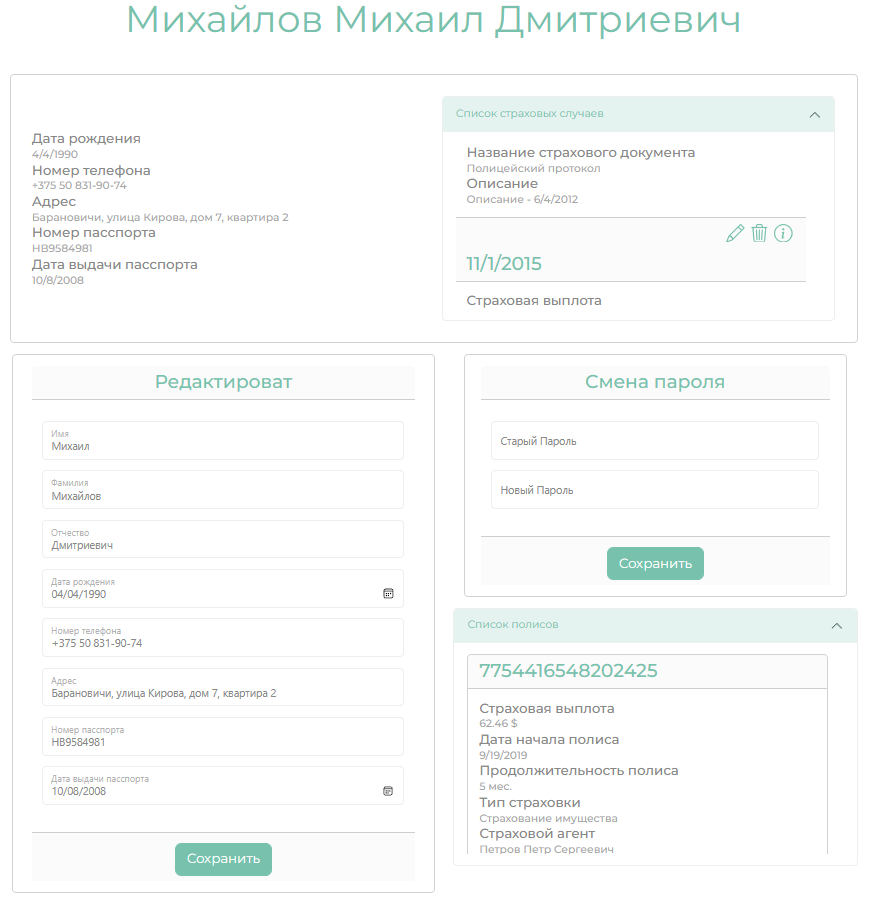


Рисунок 4.5 – Страница для редактирования текущего пользователя

Далее пользователь может просматривать информацию об определенных сущностях. Для этого для каждой сущности были реализованы страницы, на которых выводится список всех сущностей, кнопка добавления новой сущности, кнопки удаления, редактирования и получения деталей для каждой сущности. Также для более удобной и быстрой работы сайта была реализована система разбиения всех сущностей на страницы. Пример страницы для просмотра сущностей указана на рисунке 4.6.

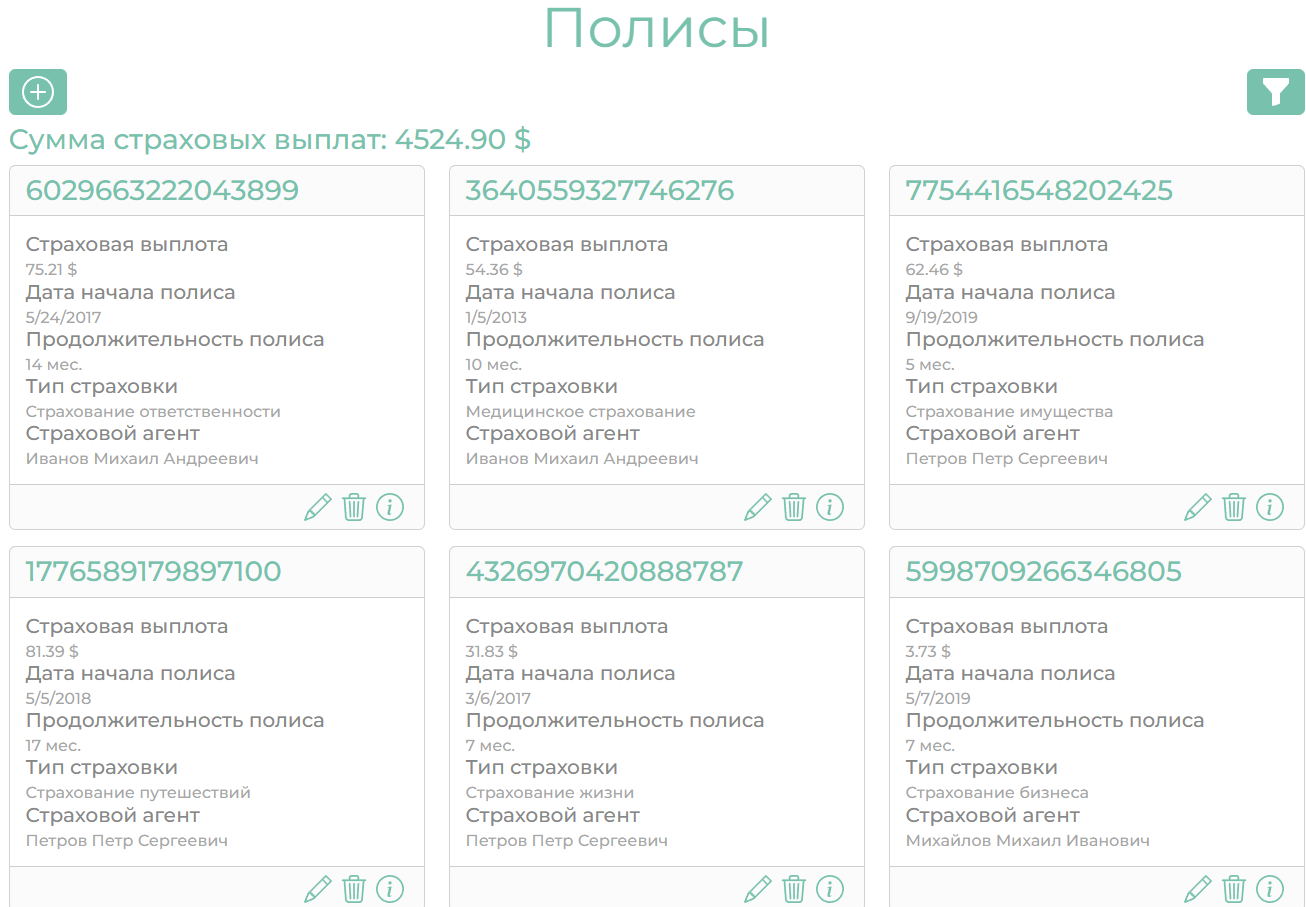


Рисунок 4.6 – Страница просмотра сущностей

Также для нахождения нужного элемента в списке была реализована система фильтров и сортировки. Для этого было реализовано отдельное меню, которое открывается при нажатии на кнопку фильтров. Пример этого меню указан на рисунке 4.7.

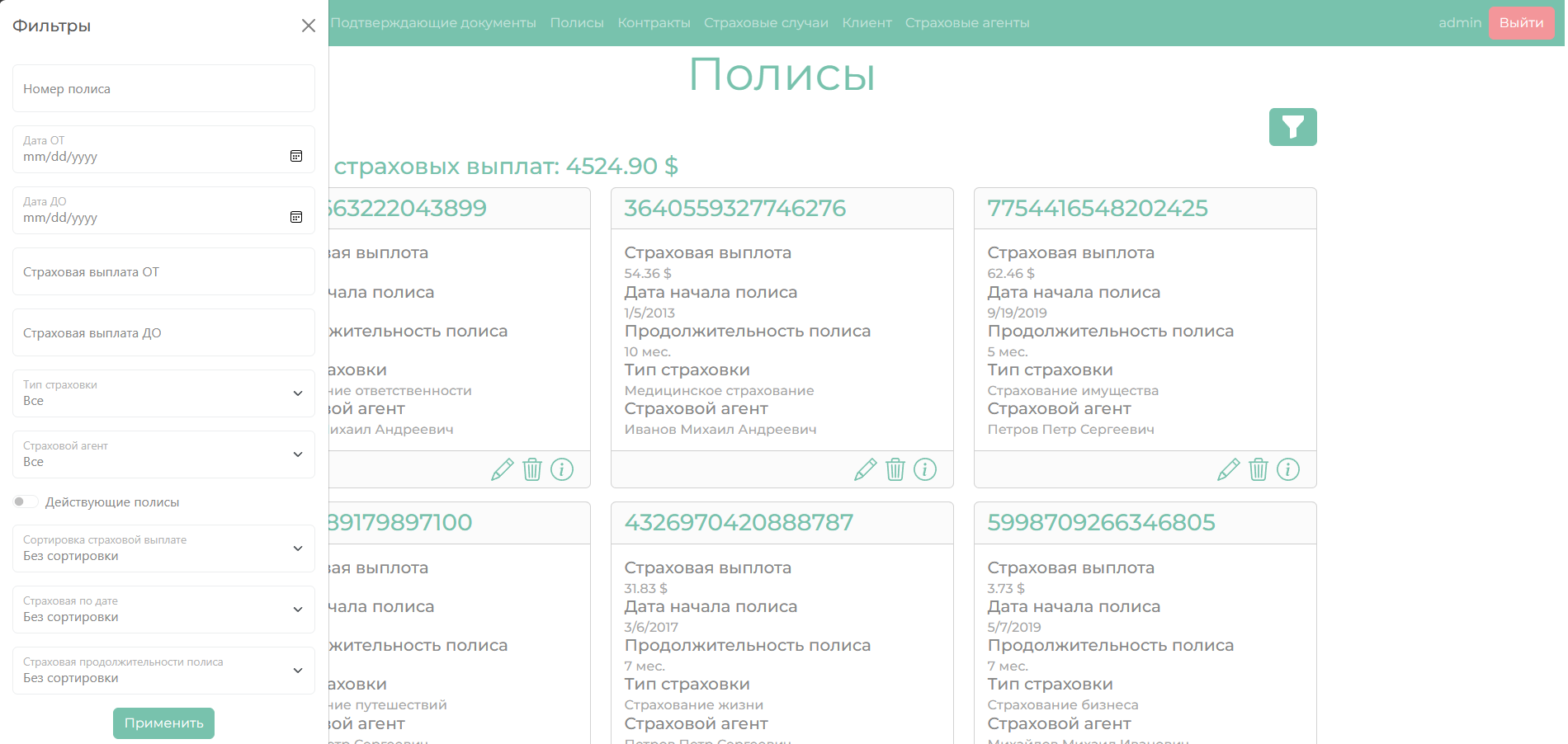


Рисунок 4.7 – Страница просмотра сущностей с панелью фильтрации

При добавлении нового элемента в базу данных открывается станица, в которой пользователю необходимо заполнить все поля. Для того чтобы все данные были правильными и логичными на каждое из полей было наложено ограничения. Соответствующая страница была реализована для редактирования сущности. Пример страницы для добавления новой сущности указан на рисунке 4.8.

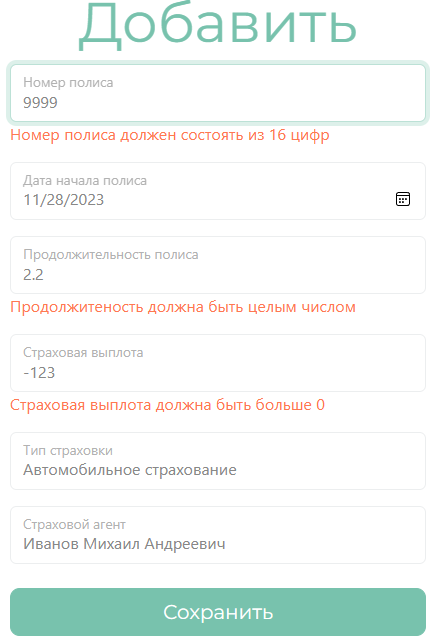


Рисунок 4.8 – Страница для добавления или обновления сущности

Далее пользователь может удалить нужную ему сущность для этого пользователь должен выбрать ее и при нажатии на кнопку удалить пользователю выводится информация об этой сущности и кнопки с подтверждением удаления и отменой. Пример этой страницы указана на рисунке 4.9.

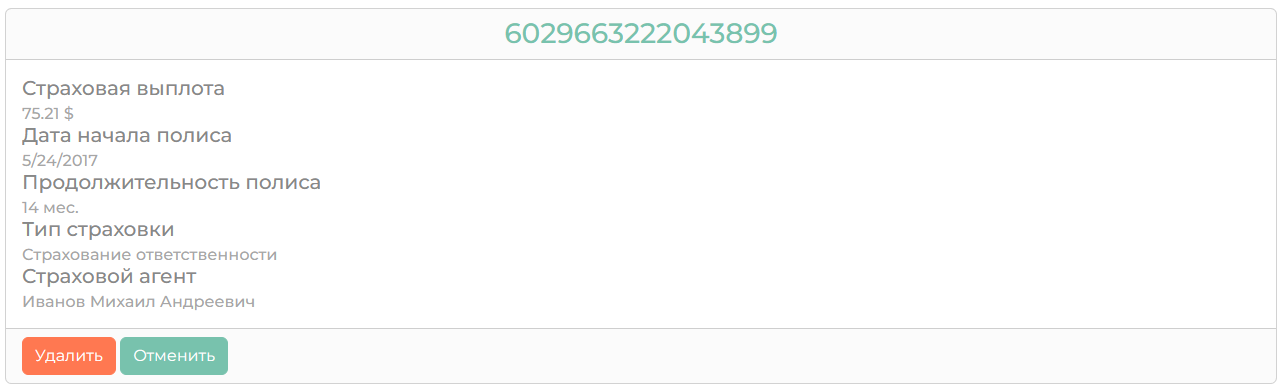


Рисунок 4.9 – Страница для удаления сущности

Также была реализована страница для получения деталей по каждой сущности. На это странице выводится подробная информация о самой сущности, и вся информация о сущностях, которые связаны с ней. Пример страницы, которая выводит детали указана на рисунке 4.10.

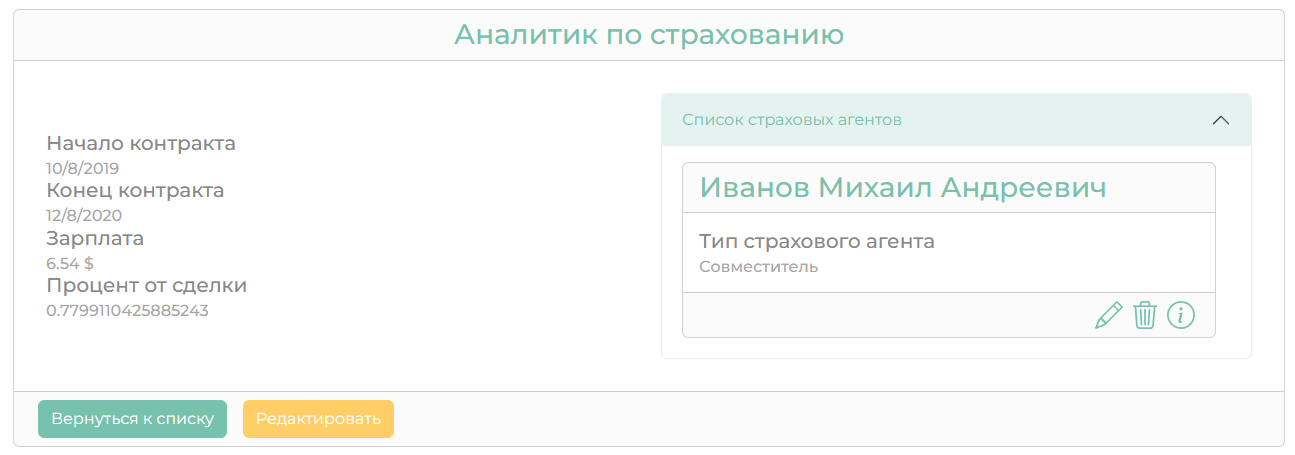


Рисунок 4.10 – Страница для получения подробной информации о сущности

# 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 ходе разработки курсового проекта по созданию информационной системы для страховой компании на базе технологии *ASP*.*NET MVC* с использованием *Microsoft SQL Server* были достигнуты следующие ключевые результаты.

Проект предоставляет комплексное решение для управления страховой деятельностью, включающее в себя учет страховых агентов, видов страхования, клиентов, полисов и страховых случаев. Разработанная система обеспечивает удобный интерфейс для работы с персоналом страховой компании и позволяет эффективно управлять всеми аспектами страхового бизнеса.

Применение архитектурного шаблона *MVC* позволило четко разделить логику представления, бизнес-логику и управление пользовательским вводом. Это обеспечило легкость поддержки, расширения и тестирования приложения. Также использование *ASP*.*NET MVC* и .*NET Core* обеспечило высокую производительность, масштабируемость и кроссплатформенность системы.

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

Проект предлагает всестороннее решение для управления ключевыми аспектами страховой деятельности, начиная от учета страховых агентов и заканчивая обработкой страховых случаев. Он предоставляет не только возможность получение детальной информации о клиентах и полисах, но и обеспечивает систематизацию и обработку данных, необходимых для принятия бизнес-решений.

Таким образом, данный проект является важным шагом в развитии информационных технологий для страхового бизнеса, предоставляя эффективные средства управления и контроля, а также демонстрируя применение современных технологий в веб-разработке и базах данных.

# Список используемых источников

1. Преимущества ASP.NET MVC. – Электрон. данные. – Режим до-ступа: https://professorweb.ru/my/ASP\_NET/mvc/level1/12.php. – Дата доступа: 23.10.2023.

2. Рихтер, Дж. CLR via C#. Программирование на платформе Microsoft .NET Framework 4.5 на языке C# / Дж. Рихтер. – пер. с англ.: – ООО Издательство «Питер», 2013. – 896 с.

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. Metanit [Электронный ресурс] – Постраничная навигация – Режим доступа: https://metanit.com/sharp/aspnetmvc/11.7.php – Дата доступа: 01.11.2023

7. Metanit [Электронный ресурс] – Контроллеры и их действия – Режим доступа: https://metanit.com/sharp/aspnet5/5.1.php – Дата доступа: 14.11.2023

# ПРИЛОЖЕНИЕ А

(обязательное)

# Листинг программы

Листинг класса *AgentType*.*cs*:

using System.ComponentModel.DataAnnotations;

namespace Models.Models;

public partial class AgentType {

public int Id { get; set; }

[Display(Name = "Тип страхового агента")]

[Required(ErrorMessage = "Тип страхового агента это обязательное поле")]

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

public virtual ICollection<InsuranceAgent> InsuranceAgents { get; set; } = new List<InsuranceAgent>();

}

Листинг класса *ApplicationUser*.*cs*:

using Microsoft.AspNetCore.Identity;

using System.ComponentModel.DataAnnotations;

namespace Models.Models {

public class ApplicationUser : IdentityUser {

[MaxLength(50)]

[Display(Name = "Имя")]

[Required(ErrorMessage = "Имя это обязательное поле")]

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

[MaxLength(50)]

[Display(Name = "Фамилия")]

[Required(ErrorMessage = "Фамилия это обязательное поле")]

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

[MaxLength(50)]

[Display(Name = "Отчество")]

[Required(ErrorMessage = "Отчество это обязательное поле")]

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

}

}

Листинг класса *Client*.*cs*:

using System.ComponentModel.DataAnnotations;

namespace Models.Models;

public partial class Client {

public int Id { get; set; }

[Display(Name = "Имя")]

[Required(ErrorMessage = "Имя это обязательное поле")]

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

[Display(Name = "Фамилия")]

[Required(ErrorMessage = "Фамилия это обязательное поле")]

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

[Display(Name = "Отчество")]

[Required(ErrorMessage = "Отчество это обязательное поле")]

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

[Display(Name = "Дата рождения")]

[Required(ErrorMessage = "Дата рождения это обязательное поле")]

public DateTime Birthdate { get; set; }

[Display(Name = "Номер телефона")]

[Required(ErrorMessage = "Номер телефона это обязательное поле")]

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

[Display(Name = "Адрес")]

[Required(ErrorMessage = "Адрес это обязательное поле")]

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

[Display(Name = "Номер пасспорта")]

[Required(ErrorMessage = "Номер пасспорта это обязательное поле")]

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

[Display(Name = "Дата выдачи пасспорта")]

[Required(ErrorMessage = "Дата выдачи пасспорта это обязательное поле")]

public DateTime PassportIssueDate { get; set; }

public virtual ICollection<InsuranceCase> InsuranceCases { get; set; } = new List<InsuranceCase>();

}

Листинг класса *Contract*.*cs*:

using System.ComponentModel.DataAnnotations;

namespace Models.Models;

public partial class Contract {

public int Id { get; set; }

[Display(Name = "Обязанности")]

[Required(ErrorMessage = "Обязанности это обязательное поле")]

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

[DataType(DataType.Date)]

[Display(Name = "Начало контракта")]

[Required(ErrorMessage = "Начало контракта это обязательное поле")]

public DateTime StartDeadline { get; set; }

[DataType(DataType.Date)]

[Display(Name = "Конец контракта")]

[Required(ErrorMessage = "Конец контракта это обязательное поле")]

public DateTime EndDeadline { get; set; }

[Range(0, double.MaxValue, ErrorMessage = "Зарплата должна быть больше 0")]

[Display(Name = "Зарплата")]

[RegularExpression(@"^\d+(\.\d+)?$", ErrorMessage = "Поле Зарплата должно быть числом")]

[Required(ErrorMessage = "Зарплата это обязательное поле")]

public decimal Salary { get; set; }

[Range(0, 1, ErrorMessage = "Процент должен быть в интервале от 0 до 1")]

[Display(Name = "Процент от сделки")]

[RegularExpression(@"^\d+(\.\d+)?$", ErrorMessage = "Поле Процент от сделки должно быть числом")]

[Required(ErrorMessage = "Процент от сделки это обязательное поле")]

public double TransactionPercent { get; set; }

public virtual ICollection<InsuranceAgent> InsuranceAgents { get; set; } = new List<InsuranceAgent>();

}

Листинг класса *InsuranceAgent*.*cs*:

using System.ComponentModel.DataAnnotations;

namespace Models.Models;

public partial class InsuranceAgent {

public int Id { get; set; }

[Display(Name = "Имя")]

[Required(ErrorMessage = "Имя это обязательное поле")]

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

[Display(Name = "Фамилия")]

[Required(ErrorMessage = "Фамилия это обязательное поле")]

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

[Display(Name = "Отчество")]

[Required(ErrorMessage = "Отчество это обязательное поле")]

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

[Display(Name = "Тип страхового агента")]

[Required(ErrorMessage = "Тип страхового агента это обязательное поле")]

public int AgentTypeId { get; set; }

[Display(Name = "Контракт")]

[Required(ErrorMessage = "Контракт это обязательное поле")]

public int ContractId { get; set; }

public virtual AgentType? AgentType { get; set; } = null!;

public virtual Contract? Contract { get; set; } = null!;

public virtual ICollection<Policy> Policies { get; set; } = new List<Policy>();

}

Листинг класса *InsuranceCase*.*cs*:

using System.ComponentModel.DataAnnotations;

namespace Models.Models;

public partial class InsuranceCase {

public int Id { get; set; }

[Display(Name = "Дата")]

[DataType(DataType.Date)]

[Required(ErrorMessage = "Дата это обязательное поле")]

public DateTime Date { get; set; }

[Display(Name = "Описание")]

[Required(ErrorMessage = "Описание это обязательное поле")]

public string? Description { get; set; }

[Display(Name = "Подтверждающие документы")]

[Required(ErrorMessage = "Подтверждающие документы это обязательное поле")]

public int SupportingDocumentId { get; set; }

[Display(Name = "Страховая выплота")]

[Required(ErrorMessage = "Страховая выплота это обязательное поле")]

public decimal InsurancePayment { get; set; }

[Display(Name = "Клиент")]

[Required(ErrorMessage = "Клиент это обязательное поле")]

public int ClientId { get; set; }

public virtual SupportingDocument? SupportingDocument { get; set; } = null!;

public virtual Client? Client { get; set; } = null!;

}

Листинг класса *InsuranceType*.*cs*:

using System.ComponentModel.DataAnnotations;

namespace Models.Models;

public partial class InsuranceType {

public int Id { get; set; }

[Display(Name = "Тип страховки")]

[Required(ErrorMessage = "Тип страховки это обязательное поле")]

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

[Display(Name = "Описание")]

[Required(ErrorMessage = "Описание это обязательное поле")]

public string? Description { get; set; }

public virtual ICollection<Policy> Policies { get; set; } = new List<Policy>();

}

Листинг класса *Policy*.*cs*:

using System.ComponentModel.DataAnnotations;

namespace Models.Models;

public partial class Policy {

public int Id { get; set; }

[Display(Name = "Страховой агент")]

[Required(ErrorMessage = "Страховой агент это обязательное поле")]

public int InsuranceAgentId { get; set; }

[DataType(DataType.Date)]

[Display(Name = "Дата начала полиса")]

[Required(ErrorMessage = "Дата начала полиса это обязательное поле")]

public DateTime ApplicationDate { get; set; }

[RegularExpression(@"^\d{16}$", ErrorMessage = "Номер полиса должен состоять из 16 цифр")]

[MaxLength(16, ErrorMessage = "Длина полиса должна быть равна 16")]

[Display(Name = "Номер полиса")]

[Required(ErrorMessage = "Номер полиса это обязательное поле")]

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

[Display(Name = "Тип страховки")]

[Required(ErrorMessage = "Тип страховки это обязательное поле")]

public int InsuranceTypeId { get; set; }

[Range(0, int.MaxValue, ErrorMessage = "Продолжительность действия полиса должна быть больше 0")]

[Display(Name = "Продолжительность полиса")]

[Required(ErrorMessage = "Продолжительность полиса это обязательное поле")]

[RegularExpression(@"^\d+$", ErrorMessage = "Продолжитеность должна быть целым числом")]

public int PolicyTerm { get; set; }

[Range(0, double.MaxValue, ErrorMessage = "Страховая выплота должна быть больше 0")]

[Display(Name = "Страховая выплота")]

[Required(ErrorMessage = "Страховая выплота это обязательное поле")]

public decimal PolicyPayment { get; set; }

public virtual InsuranceAgent? InsuranceAgent { get; set; } = null!;

public virtual InsuranceType? InsuranceType { get; set; } = null!;

}

Листинг класса *PolicyClient*.*cs*:

using System.ComponentModel.DataAnnotations;

namespace Models.Models {

public class PolicyClient {

public int Id { get; set; }

[Display(Name = "Полис")]

[Required(ErrorMessage = "Полис это обязательное поле")]

public int PolicyId { get; set; }

[Display(Name = "Клиент")]

[Required(ErrorMessage = "Клиент это обязательное поле")]

public int ClientId { get; set; }

public virtual Policy Policy { get; set; } = null!;

public virtual Client Client { get; set; } = null!;

}

}

Листинг класса *SupportingDocument*.*cs*:

using System.ComponentModel.DataAnnotations;

namespace Models.Models;

public partial class SupportingDocument {

public int Id { get; set; }

[Display(Name = "Название")]

[Required(ErrorMessage = "Название это обязательное поле")]

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

[Display(Name = "Описание")]

[Required(ErrorMessage = "Описание это обязательное поле")]

public string? Description { get; set; }

public virtual ICollection<InsuranceCase> InsuranceCases { get; set; } = new List<InsuranceCase>();

}

Листинг класса *SortType*.*cs*:

namespace Models.Utilities {

public enum SortType {

NonSort = 0,

AscendingSort = 1,

DescendingSort = 2

}

}

Листинг класса *ClientFilterModel*.*cs*:

using Models.Utilities;

using System.ComponentModel.DataAnnotations;

namespace Models.ViewModels.FilterViewModels {

public class ClientFilterModel {

[Display(Name = "Имя")]

public string Name { get; set; }

[Display(Name = "Фамилия")]

public string Surname { get; set; }

[Display(Name = "Отчество")]

public string MiddleName { get; set; }

[DataType(DataType.Date)]

[Display(Name = "Дата рождения ОТ")]

public DateTime? BirthdateStart { get; set; }

[DataType(DataType.Date)]

[Display(Name = "Дата рождения ДО")]

public DateTime? BirthdateEnd { get; set; }

[Display(Name = "Адрес")]

public string Address { get; set; }

[Display(Name = "Номер телефона")]

public string MobilePhone { get; set; }

[MaxLength(9)]

[Display(Name = "Номер пасспорта")]

public string PassportNumber { get; set; }

[DataType(DataType.Date)]

[Display(Name = "Дата выдачи пасспорта ОТ")]

public DateTime? PassportIssueDateStart { get; set; }

[DataType(DataType.Date)]

[Display(Name = "Дата выдачи пасспорта ДО")]

public DateTime? PassportIssueDateEnd { get; set; }

[Display(Name = "Полис истекает в следующем месяце")]

public bool PolicyIsFinishNextMounth { get; set; }

[Display(Name = "Сортировка по имени")]

public SortType SortTypeName { get; set; }

[Display(Name = "Сортировка по фамилии")]

public SortType SortTypeSurname { get; set; }

[Display(Name = "Сортировка по отчеству")]

public SortType SortTypeMiddleName { get; set; }

[Display(Name = "Сортировка по дате рождения")]

public SortType SortTypeBirthdate { get; set; }

[Display(Name = "Сортировка по дате выдачи паспорта")]

public SortType SortTypePassportIssueDate { get; set; }

}

}

Листинг класса *ContractFilterModel*.*cs*:

using Models.Utilities;

using System.ComponentModel.DataAnnotations;

namespace Models.ViewModels.FilterViewModels {

public class ContractFilterModel {

[Display(Name = "Обязанности")]

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

[Display(Name = "Срок действия контракта ОТ")]

public DateTime? StartDeadLine { get; set; }

[Display(Name = "Срок действия контракта ДО")]

public DateTime? EndDeadLine { get; set; }

[Display(Name = "Зарплата ОТ")]

public decimal? MinSalary { get; set; }

[Display(Name = "Зарплата ДО")]

public decimal? MaxSalary { get; set; }

[Display(Name = "Процент от сделки ОТ")]

public double? MinTransactionPercent { get; set; }

[Display(Name = "Процент от сделки ДО")]

public double? MaxTransactionPercent { get; set; }

[Display(Name = "Сортировка по обязанностям")]

public SortType SortTypeResponsobilities { get; set; }

[Display(Name = "Сортировка по зарпалате")]

public SortType SortTypeSalary { get; set; }

[Display(Name = "Сортировка по проценту от сделки")]

public SortType SortTypeTransactionPercent { get; set; }

[Display(Name = "Сортировка по продолжительности контракта")]

public SortType SortTypeContractDuration { get; set; }

}

}

Листинг класса *InsuranceAgentFilterModel*.*cs*:

using Models.Utilities;

using System.ComponentModel.DataAnnotations;

namespace Models.ViewModels.FilterViewModels {

public class InsuranceAgentFilterModel {

[Display(Name = "Имя")]

public string Name { get; set; }

[Display(Name = "Фамилия")]

public string Surname { get; set; }

[Display(Name = "Отчество")]

public string MiddleName { get; set; }

[Display(Name = "Тип страхового агента")]

public string AgentType { get; set; }

[Display(Name = "Обязанности")]

public string Responsibilities { get; set; }

[Display(Name = "Срок действия контракта ОТ")]

public DateTime? StartDeadLine { get; set; }

[Display(Name = "Срок действия контракта ДО")]

public DateTime? EndDeadLine { get; set; }

[Display(Name = "Зарплата ОТ")]

public decimal? MinSalary { get; set; }

[Display(Name = "Зарплата ДО")]

public decimal? MaxSalary { get; set; }

[Display(Name = "Процент от сделки ОТ")]

public double? MinTransactionPercent { get; set; }

[Display(Name = "Процент от сделки ДО")]

public double? MaxTransactionPercent { get; set; }

[Display(Name = "Сортировка по имени")]

public SortType SortTypeName { get; set; }

[Display(Name = "Сортировка по фамилии")]

public SortType SortTypeSurname { get; set; }

[Display(Name = "Сортировка по отчеству")]

public SortType SortTypeMiddleName { get; set; }

[Display(Name = "Сортировка по зарплате")]

public SortType SortTypeSalary { get; set; }

[Display(Name = "Сортировка по проценту от сделки")]

public SortType SortTypeTransactionPercent { get; set; }

[Display(Name = "Сортировка по продолжительности контракта")]

public SortType SortTypeContractDuration { get; set; }

}

}

Листинг класса *InsuranceCaseFilterModel*.*cs*:

using Microsoft.AspNetCore.Mvc.Rendering;

using Models.Utilities;

using System.ComponentModel.DataAnnotations;

using System.Text.Json.Serialization;

namespace Models.ViewModels.FilterViewModels {

public class InsuranceCaseFilterModel {

[Display(Name = "Подтверждащий документ")]

public string SupportingDocument { get; set; }

[Display(Name = "Дата ОТ")]

public DateTime? StartDate { get; set; }

[Display(Name = "Дата ДО")]

public DateTime? EndDate { get; set; }

[Display(Name = "Страховая выплата ОТ")]

public decimal? MinInsurancePayment { get; set; }

[Display(Name = "Страховая выплата ДО")]

public decimal? MaxInsurancePayment { get; set; }

[Display(Name = "Тип страховки")]

public int? InsuranceTypeId { get; set; }

[Display(Name = "Сортировка по страховой выплате")]

public SortType SortTypeInsurancePayment { get; set; }

[Display(Name = "Сортировка по дате")]

public SortType SortTypeDate { get; set; }

[Display(Name = "Сортировка по подтверждающему документу")]

public SortType SortTypeSupportingDocument { get; set; }

[JsonIgnore]

public IEnumerable<SelectListItem> InsuranceTypeList { get; set; } = null!;

}

}

Листинг класса *InsuranceTypeFilterModel*.*cs*:

using Models.Utilities;

using System.ComponentModel.DataAnnotations;

namespace Models.ViewModels.FilterViewModels {

public class InsuranceTypeFilterModel {

[Display(Name = "Название")]

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

[Display(Name = "Описание")]

public string? Description { get; set; }

[Display(Name = "Сортировка по имени")]

public SortType SortTypeName { get; set; }

[Display(Name = "Сортировка по описанию")]

public SortType SortTypeDescription { get; set; }

}

}

Листинг класса *PolicyFilterModel*.*cs*:

using Microsoft.AspNetCore.Mvc.Rendering;

using Models.Utilities;

using System.ComponentModel.DataAnnotations;

using System.Text.Json.Serialization;

namespace Models.ViewModels.FilterViewModels {

public class PolicyFilterModel {

[Display(Name = "Страховой агент")]

public int? InsuranceAgentId { get; set; }

[Display(Name = "Тип страховки")]

public int? InsuranceTypeId { get; set; }

[MaxLength(16)]

[Display(Name = "Номер полиса")]

public string PolicyNumber { get; set; }

[Display(Name = "Дата ОТ")]

public DateTime? ApplicationDateStart { get; set; }

[Display(Name = "Дата ДО")]

public DateTime? ApplicationDateEnd { get; set; }

[Display(Name = "Страховая выплата ОТ")]

public decimal? MinPolicyPayment { get; set; }

[Display(Name = "Страховая выплата ДО")]

public decimal? MaxPolicyPayment { get; set; }

[Display(Name = "Действующие полисы")]

public bool PolicyIsActing { get; set; }

[Display(Name = "Сортировка страховой выплате")]

public SortType SortTypePolicyPaymen { get; set; }

[Display(Name = "Страховая продолжительности полиса")]

public SortType SortTypePolicyTerm { get; set; }

[Display(Name = "Страховая по дате")]

public SortType SortTypeApplicationDate { get; set; }

[JsonIgnore]

public IEnumerable<SelectListItem> InsuranceAgentList { get; set; } = null!;

[JsonIgnore]

public IEnumerable<SelectListItem> InsuranceTypeList { get; set; } = null!;

}

}

Листинг класса *SupportingDocumentFilterModel*.*cs*:

using Models.Utilities;

using System.ComponentModel.DataAnnotations;

namespace Models.ViewModels.FilterViewModels {

public class SupportingDocumentFilterModel {

[Display(Name = "Название")]

public string Name { get; set; }

[Display(Name = "Сортировка по названию")]

public SortType SortTypeName { get; set; }

}

}

Листинг класса *ChangePasswordViewModel*.*cs*:

using System.ComponentModel.DataAnnotations;

namespace Models.ViewModels.IdentityViewModels {

public class ChangePasswordViewModel {

[Required(ErrorMessage = "Необходимо ввести старый пароль в поле.")]

[Display(Name = "Старый Пароль")]

[DataType(DataType.Password)]

public string OldPassword { get; set; }

[Required(ErrorMessage = "Необходимо ввести новый пароль в поле.")]

[Display(Name = "Новый Пароль")]

[DataType(DataType.Password)]

public string NewPassword { get; set; }

}

}

Листинг класса *ClientInputModel*.*cs*:

using System.ComponentModel.DataAnnotations;

namespace Models.ViewModels.IdentityViewModels {

public class ClientInputModel {

[Required(ErrorMessage = "Необходимо ввести дату рождения в поле.")]

[DataType(DataType.Date)]

[Display(Name = "Дата")]

public DateTime BrithDate { get; set; }

[Required(ErrorMessage = "Необходимо ввести номер телефона в поле.")]

[Display(Name = "Номер телефона")]

public string MobilePhone { get; set; }

[Required(ErrorMessage = "Необходимо ввести адрес в поле.")]

[Display(Name = "Адрес")]

public string Address { get; set; }

[Required(ErrorMessage = "Необходимо ввести номер пасспорта в поле.")]

[Display(Name = "Номер пасспорта")]

public string PassportNumber { get; set; }

[Required(ErrorMessage = "Необходимо ввести дату выдачи пасспорта в поле.")]

[DataType(DataType.Date)]

[Display(Name = "Дата выдачи пасспорта")]

public DateTime PassportIssueDate { get; set; }

}

}

Листинг класса *InsuranceAgentInputModel*.*cs*:

using Microsoft.AspNetCore.Mvc.ModelBinding.Validation;

using Microsoft.AspNetCore.Mvc.Rendering;

using System.ComponentModel.DataAnnotations;

namespace Models.ViewModels.IdentityViewModels {

public class InsuranceAgentInputModel {

[Required(ErrorMessage = "Необходимо выбрать тип агента из списка")]

[Display(Name = "Тип страхового агента")]

public string? AgentType { get; set; }

[ValidateNever]

public IEnumerable<SelectListItem> AgentTypeList { get; set; }

[Required(ErrorMessage = "Необходимо выбрать контракт из списка")]

[Display(Name = "Контракты")]

public string? Contract { get; set; }

[ValidateNever]

public IEnumerable<SelectListItem> ContractList { get; set; }

}

}

Листинг класса *LoginInputModel*.*cs*:

using System.ComponentModel.DataAnnotations;

namespace Models.ViewModels.IdentityViewModels {

public class LoginInputModel {

[Required(ErrorMessage = "Необходимо ввести имя в поле.")]

[Display(Name = "Имя")]

public string Name { get; set; }

[Required(ErrorMessage = "Необходимо ввести фамилию в поле.")]

[Display(Name = "Фамилия")]

public string Surname { get; set; }

[Required(ErrorMessage = "Необходимо ввести отчество в поле.")]

[Display(Name = "Отчество")]

public string MiddleName { get; set; }

[Required(ErrorMessage = "Необходимо ввести пароль в поле.")]

[Display(Name = "Пароль")]

[DataType(DataType.Password)]

public string Password { get; set; }

}

}

Листинг класса *RegisterInputModel*.*cs*:

using Microsoft.AspNetCore.Mvc.ModelBinding.Validation;

using Microsoft.AspNetCore.Mvc.Rendering;

using System.ComponentModel.DataAnnotations;

namespace Models.ViewModels.IdentityViewModels {

public class RegisterInputModel {

[Required(ErrorMessage = "Необходимо ввести имя в поле.")]

[Display(Name = "Имя")]

public string Name { get; set; }

[Required(ErrorMessage = "Необходимо ввести фамилию в поле.")]

[Display(Name = "Фамилия")]

public string Surname { get; set; }

[Required(ErrorMessage = "Необходимо ввести отчество в поле.")]

[Display(Name = "Отчество")]

public string MiddleName { get; set; }

[Required(ErrorMessage = "Необходимо ввести пароль в поле.")]

[Display(Name = "Пароль")]

[DataType(DataType.Password)]

public string Password { get; set; }

[Required(ErrorMessage = "Необходимо выбрать роль из списка.")]

[Display(Name = "Роль")]

public string? Role { get; set; }

[ValidateNever]

public IEnumerable<SelectListItem> RoleList { get; set; }

}

}

Листинг класса *PageModel*.*cs*:

namespace Models.ViewModels.PageViewModels {

public class PageModel<T, K> {

public IEnumerable<T> Entities { get; set; }

public int PageNumber { get; set; }

public int TotalPages { get; set; }

public int PageSize { get; set; }

public K FilterModel { get; set; }

public PageModel(int page, int pageSize, IEnumerable<T> entities, K filterModel) {

int totalPages = (int)Math.Ceiling((double)entities.Count() / pageSize);

page = Math.Max(1, Math.Min(totalPages, page));

entities = entities

.Skip((page - 1) \* pageSize)

.Take(pageSize)

.ToList();

PageNumber = page;

TotalPages = totalPages;

PageSize = pageSize;

Entities = entities;

FilterModel = filterModel;

}

}

}

Листинг класса *PageModelWithAggregateValue*.*cs*:

namespace Models.ViewModels.PageViewModels {

public class PageModelWithAggregateValue<T, K, E> : PageModel<T, K> {

public E AggregateValue { get; set; }

public PageModelWithAggregateValue(

int page,

int pageSize,

IEnumerable<T> entities,

K filterModel,

E aggregateValue) : base(page, pageSize, entities, filterModel) {

AggregateValue = aggregateValue;

}

}

}

Листинг класса *InsuranceCompanyContext*.*cs*:

using Microsoft.EntityFrameworkCore;

using Models.Models;

namespace Repository;

public class InsuranceCompanyContext : DbContext {

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

public virtual DbSet<AgentType> AgentTypes { get; set; }

public virtual DbSet<Client> Clients { get; set; }

public virtual DbSet<Contract> Contracts { get; set; }

public virtual DbSet<InsuranceAgent> InsuranceAgents { get; set; }

public virtual DbSet<InsuranceCase> InsuranceCases { get; set; }

public virtual DbSet<InsuranceType> InsuranceTypes { get; set; }

public virtual DbSet<Policy> Policies { get; set; }

public virtual DbSet<SupportingDocument> SupportingDocuments { get; set; }

public virtual DbSet<PolicyClient> PolicyClients { get; set; }

}

Листинг класса *InsuranceCompanyIdentityContext*.*cs*:

using Microsoft.AspNetCore.Identity;

using Microsoft.AspNetCore.Identity.EntityFrameworkCore;

using Microsoft.EntityFrameworkCore;

using Models.Models;

namespace Repository {

public partial class InsuranceCompanyIdentityContext : IdentityDbContext<IdentityUser> {

public InsuranceCompanyIdentityContext(DbContextOptions options) : base(options) {}

public virtual DbSet<ApplicationUser> ApplicationUsers { get; set; }

}

}

Листинг класса *InitializeData*.*cs*:

namespace Repository.DbInitializer {

public static class InitializeData {

public static List<string> IdentityRoleNames = new() {

"Клиент",

"Страховой агент"

};

public static List<string> AgentTypes = new() {

"Штатный работник",

"Совместитель"

};

public static List<string> Responsibilities = new() {

"Страховой агент",

"Актюарий",

"Агент по обслуживанию клиентов",

"Менеджер по продажам",

"Администратор страховых полисов",

"Эксперт по риску",

"Управляющий отделом страхования",

"Аналитик по страхованию",

"Администратор базы данных страхования"

};

public static List<string> Names = new() {

"Иван",

"Анна",

"Петр",

"Екатерина",

"Александр",

"София",

"Михаил",

"Елена",

"Дмитрий",

"Мария"

};

public static List<string> Surnames = new() {

"Иванов",

"Петров",

"Сидоров",

"Смирнов",

"Козлов",

"Михайлов",

"Александров",

"Егоров",

"Васильев",

"Кузнецов"

};

public static List<string> MiddleNames = new() {

"Иванович",

"Петрович",

"Сидорович",

"Михайлович",

"Александрович",

"Дмитриевич",

"Сергеевич",

"Андреевич",

"Николаевич",

"Геннадьевич"

};

public static List<string> SupportingDocumentNames = new() {

"Свидетельство о происшествии",

"Полицейский протокол",

"Медицинская справка",

"Диагностический отчет",

"Договор страхования",

"Фотографии ущерба",

"Подтверждающее письмо клиента",

"Справка с места работодателя",

"Свидетельство о смерти",

"Свидетельство о рождении"

};

public static List<string> Addresses = new()

{

"Гродно, улица Советская, дом 14, квартира 3",

"Минск, проспект Независимости, дом 45, квартира 7",

"Брест, улица Ленина, дом 28, квартира 12",

"Гомель, проспект Ленина, дом 55, квартира 1",

"Витебск, улица Пушкина, дом 19, квартира 4",

"Могилев, улица Гагарина, дом 36, квартира 9",

"Барановичи, улица Кирова, дом 7, квартира 2",

"Пинск, улица Советская, дом 10, квартира 6",

"Бобруйск, улица Лермонтова, дом 3, квартира 8",

"Мозырь, проспект Мира, дом 30, квартира 5"

};

public static List<string> InsuranceTypeNames = new() {

"Автомобильное страхование",

"Страхование жизни",

"Медицинское страхование",

"Страхование недвижимости",

"Страхование от несчастных случаев",

"Страхование ответственности",

"Страхование имущества",

"Страхование путешествий",

"Страхование бизнеса",

"Страхование грузов"

};

}

}

Листинг класса *RandomExtensions*.*cs*:

namespace Repository.DbInitializer {

public static class RandomExtensions {

public static DateTime NextDate(this Random rand, int startYear, int endYear) {

var day = rand.Next(1, 29);

var month = rand.Next(1, 13);

var year = rand.Next(startYear, endYear);

return new DateTime(year, month, day);

}

public static T NextItem<T>(this Random rand, IList<T> items) {

return items[rand.Next(0, items.Count)];

}

public static string NextPhoneNumber(this Random rand) {

return $"+375 {rand.Next(10, 100)} {rand.Next(100, 1000)}-{rand.Next(10, 100)}-{rand.Next(10, 100)}";

}

public static string NextPassportNumber(this Random rand) {

return $"HB{rand.Next(1000000, 10000000)}";

}

public static string NextPolicyNumber(this Random rand) {

return $"{rand.NextInt64(1000000000000000, 10000000000000000)}";

}

}

}

Листинг класса *InsuranceCompanyInitializer*.*cs*:

using Microsoft.AspNetCore.Identity;

using Microsoft.EntityFrameworkCore;

using Models.Models;

namespace Repository.DbInitializer {

public static class InsuranceCompanyInitializer {

private static Random rand = new();

public static void Initialize(InsuranceCompanyContext db, InsuranceCompanyIdentityContext identityDb, UserManager<ApplicationUser> userManager) {

if (!db.Database.CanConnect()) {

db.Database.Migrate();

}

if (!identityDb.Roles.Any()) {

InitializeIdentityRoles(identityDb);

}

if (!db.AgentTypes.Any()) {

InitializeAgentTypes(db);

}

if (!db.Contracts.Any()) {

InitializeContracts(db);

}

if (!db.InsuranceAgents.Any()) {

InitializeInsuranceAgents(db, userManager);

}

if (!db.SupportingDocuments.Any()) {

InitializeSupportingDocuments(db);

}

if (!db.Clients.Any()) {

InitializeClients(db, userManager);

}

if (!db.InsuranceCases.Any()) {

InitializeInsuranceCases(db);

}

if (!db.InsuranceTypes.Any()) {

InitializeInsuranceTypes(db);

}

if (!db.Policies.Any()) {

InitializePolicies(db);

}

if (!db.PolicyClients.Any()) {

InitializePolicyClients(db);

}

db.SaveChanges();

}

private static void InitializeIdentityRoles(InsuranceCompanyIdentityContext identityDb) {

foreach (var name in InitializeData.IdentityRoleNames) {

identityDb.Add(new IdentityRole() { Name = name, NormalizedName = name.ToUpper() });

}

identityDb.SaveChanges();

}

private static void InitializePolicyClients(InsuranceCompanyContext db) {

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

int policyId = rand.Next(1, 101);

int clientId = rand.Next(1, 51);

db.Add(new PolicyClient() {

PolicyId = policyId,

ClientId = clientId

});

}

db.SaveChanges();

}

private static void InitializePolicies(InsuranceCompanyContext db) {

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

int insuranceAgentId = rand.Next(1, 51);

var applicationDate = rand.NextDate(2010, 2023);

string policyNumber = rand.NextPolicyNumber();

int insuranceTypeId = rand.Next(1, InitializeData.InsuranceTypeNames.Count + 1);

int policyTerm = rand.Next(1, 20);

decimal policyPayment = (decimal)(100 \* rand.NextDouble());

db.Add(new Policy() {

InsuranceAgentId = insuranceAgentId,

ApplicationDate = applicationDate,

PolicyNumber = policyNumber,

InsuranceTypeId = insuranceTypeId,

PolicyTerm = policyTerm,

PolicyPayment = policyPayment

});

}

db.SaveChanges();

}

private static void InitializeInsuranceTypes(InsuranceCompanyContext db) {

foreach (var name in InitializeData.InsuranceTypeNames) {

db.Add(new InsuranceType() { Name = name, Description = $"Описание - {name}" });

}

}

private static void InitializeClients(InsuranceCompanyContext db, UserManager<ApplicationUser> userManager) {

for (int i = 0; i < 50; i++) {

string name = rand.NextItem(InitializeData.Names);

string surname = rand.NextItem(InitializeData.Surnames);

string middleName = rand.NextItem(InitializeData.MiddleNames);

var birthdate = rand.NextDate(1950, 2005);

string mobilePhone = rand.NextPhoneNumber();

string address = rand.NextItem(InitializeData.Addresses);

string passportNumber = rand.NextPassportNumber();

var passportIssueDate = rand.NextDate(2000, 2023);

string password = $"Client{i}!";

var userName = $"{name} {surname} {middleName}";

var applicationUser = new ApplicationUser() {

UserName = userName,

NormalizedUserName = userName,

Name = name,

Surname = surname,

MiddleName = middleName

};

var result = userManager.CreateAsync(applicationUser, password).GetAwaiter().GetResult();

if (result.Succeeded) {

userManager.AddToRoleAsync(applicationUser, "Клиент").GetAwaiter().GetResult();

db.Add(new Client() {

Name = name,

Surname = surname,

MiddleName = middleName,

Birthdate = birthdate,

MobilePhone = mobilePhone,

Address = address,

PassportNumber = passportNumber,

PassportIssueDate = passportIssueDate

});

}

else i--;

}

db.SaveChanges();

}

private static void InitializeInsuranceCases(InsuranceCompanyContext db) {

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

var date = rand.NextDate(2010, 2023);

int supportingDocumentId = rand.Next(1, InitializeData.SupportingDocumentNames.Count + 1);

decimal insurancePayment = (decimal)(100 \* rand.NextDouble());

int clientId = rand.Next(1, 51);

string description = $"Описание - {date.ToShortDateString()}";

db.Add(new InsuranceCase() {

Date = date,

SupportingDocumentId = supportingDocumentId,

InsurancePayment = insurancePayment,

ClientId = clientId,

Description = description

});

}

db.SaveChanges();

}

private static void InitializeSupportingDocuments(InsuranceCompanyContext db) {

foreach (var name in InitializeData.SupportingDocumentNames) {

db.Add(new SupportingDocument() { Name = name, Description = $"Описание - {name}" });

}

db.SaveChanges();

}

private static void InitializeAgentTypes(InsuranceCompanyContext db) {

foreach (var type in InitializeData.AgentTypes) {

db.Add(new AgentType() { Type = type });

}

db.SaveChanges();

}

private static void InitializeContracts(InsuranceCompanyContext db) {

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

var startDeadline = rand.NextDate(2010, 2023);

var endDeadline = startDeadline.AddMonths(rand.Next(1, 20));

var responsibilities = rand.NextItem(InitializeData.Responsibilities);

decimal salary = (decimal)(100 \* rand.NextDouble());

double transactionPercent = rand.NextDouble();

db.Add(new Contract() {

Responsibilities = responsibilities,

StartDeadline = startDeadline,

EndDeadline = endDeadline,

Salary = salary,

TransactionPercent = transactionPercent

});

}

db.SaveChanges();

}

private static void InitializeInsuranceAgents(InsuranceCompanyContext db, UserManager<ApplicationUser> userManager) {

for (int i = 0; i < 50; i++) {

string name = rand.NextItem(InitializeData.Names);

string surname = rand.NextItem(InitializeData.Surnames);

string middleName = rand.NextItem(InitializeData.MiddleNames);

int agentTypeId = rand.Next(1, InitializeData.AgentTypes.Count + 1);

int contractId = rand.Next(1, 101);

string password = $"InsuranceAgent{i}!";

var userName = $"{name} {surname} {middleName}";

var applicationUser = new ApplicationUser() {

UserName = userName,

NormalizedUserName = userName,

Name = name,

Surname = surname,

MiddleName = middleName

};

var result = userManager.CreateAsync(applicationUser, password).GetAwaiter().GetResult();

if (result.Succeeded) {

userManager.AddToRoleAsync(applicationUser, "Страховой агент").GetAwaiter().GetResult();

db.Add(new InsuranceAgent() {

Name = name,

Surname = surname,

MiddleName = middleName,

AgentTypeId = agentTypeId,

ContractId = contractId,

});

}

else i--;

}

db.SaveChanges();

}

}

}

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 *InsuranceCompanyFactory*.*cs*:

using Microsoft.EntityFrameworkCore;

using Repository;

namespace Service {

public static class InsuranceCompanyFactory {

public static IEnumerable<T> GetEnites<T>(string entityName, InsuranceCompanyContext db) {

return entityName switch {

"AgentType" => GetAgentTypes<T>(db),

"Client" => GetClients<T>(db),

"Contract" => GetContracts<T>(db),

"InsuranceType" => GetInsuranceTypes<T>(db),

"SupportingDocument" => GetSupportingDocuments<T>(db),

"InsuranceAgent" => GetInsuranceAgents<T>(db),

"InsuranceCase" => GetInsuranceCases<T>(db),

"Policy" => GetPolicies<T>(db),

"PolicyClient" => GetPolicyClients<T>(db),

\_ => throw new NullReferenceException(),

};

}

private static IEnumerable<T> GetAgentTypes<T>(InsuranceCompanyContext db) => (IEnumerable<T>)db.AgentTypes.ToList();

private static IEnumerable<T> GetClients<T>(InsuranceCompanyContext db) => (IEnumerable<T>)db.Clients.ToList();

private static IEnumerable<T> GetContracts<T>(InsuranceCompanyContext db) => (IEnumerable<T>)db.Contracts.ToList();

private static IEnumerable<T> GetInsuranceTypes<T>(InsuranceCompanyContext db) => (IEnumerable<T>)db.InsuranceTypes.ToList();

private static IEnumerable<T> GetSupportingDocuments<T>(InsuranceCompanyContext db) => (IEnumerable<T>)db.SupportingDocuments.ToList();

private static IEnumerable<T> GetInsuranceAgents<T>(InsuranceCompanyContext db) {

return (IEnumerable<T>)db.InsuranceAgents

.Include(e => e.AgentType)

.Include(e => e.Contract)

.ToList();

}

private static IEnumerable<T> GetInsuranceCases<T>(InsuranceCompanyContext db) {

return (IEnumerable<T>)db.InsuranceCases

.Include(e => e.Client)

.Include(e => e.SupportingDocument)

.ToList();

}

private static IEnumerable<T> GetPolicies<T>(InsuranceCompanyContext db) {

return (IEnumerable<T>)db.Policies

.Include(e => e.InsuranceType)

.Include(e => e.InsuranceAgent)

.ThenInclude(e => e.Contract)

.Include(e => e.InsuranceAgent)

.ThenInclude(e => e.AgentType)

.ToList();

}

private static IEnumerable<T> GetPolicyClients<T>(InsuranceCompanyContext db) {

return (IEnumerable<T>)db.PolicyClients

.Include(e => e.Client)

.Include(e => e.Policy)

.ThenInclude(e => e.InsuranceType)

.Include(e => e.Policy)

.ThenInclude(e => e.InsuranceAgent)

.ThenInclude(e => e.AgentType)

.Include(e => e.Policy)

.ThenInclude(e => e.InsuranceAgent)

.ThenInclude(e => e.Contract)

.ToList();

}

}

}

Листинг класса *InsuranceCompanyFilter*.*cs*:

using Models.Models;

using Models.ViewModels.FilterViewModels;

namespace Service {

public class InsuranceCompanyFilter {

private readonly InsuranceCompanyCache \_cache;

public InsuranceCompanyFilter(InsuranceCompanyCache cache) {

\_cache = cache;

}

public IEnumerable<InsuranceType> Filter(InsuranceTypeFilterModel filter) {

var result = \_cache.GetEntity<InsuranceType>();

if (filter != null) {

return result

.Where(e => filter.Name == null || e.Name.StartsWith(filter.Name))

.Where(e => filter.Description == null || e.Description.StartsWith(filter.Description))

.Sort(filter);

}

return result;

}

public IEnumerable<Contract> Filter(ContractFilterModel filter) {

var result = \_cache.GetEntity<Contract>();

if (filter != null) {

return result

.Where(e => filter.Responsibilities == null || e.Responsibilities.StartsWith(filter.Responsibilities))

.Where(e => filter.StartDeadLine == null || e.StartDeadline >= filter.StartDeadLine)

.Where(e => filter.EndDeadLine == null || e.EndDeadline <= filter.EndDeadLine)

.Where(e => filter.MinSalary == null || e.Salary >= filter.MinSalary)

.Where(e => filter.MaxSalary == null || e.Salary <= filter.MaxSalary)

.Where(e => filter.MinTransactionPercent == null || e.TransactionPercent >= filter.MinTransactionPercent)

.Where(e => filter.MaxTransactionPercent == null || e.TransactionPercent <= filter.MaxTransactionPercent)

.Sort(filter);

}

return result;

}

public IEnumerable<Policy> Filter(PolicyFilterModel filter) {

var nowTime = DateTime.Now;

var result = \_cache.GetEntity<Policy>();

if (filter != null) {

return result

.Where(e => filter.InsuranceTypeId == null || filter.InsuranceTypeId == -1 || filter.InsuranceTypeId == e.InsuranceTypeId)

.Where(e => filter.InsuranceAgentId == null || filter.InsuranceAgentId == -1 || filter.InsuranceAgentId == e.InsuranceAgentId)

.Where(e => filter.PolicyNumber == null || filter.PolicyNumber == e.PolicyNumber)

.Where(e => filter.ApplicationDateStart == null || filter.ApplicationDateStart <= e.ApplicationDate)

.Where(e => filter.ApplicationDateEnd == null || filter.ApplicationDateEnd >= e.ApplicationDate.AddMonths(e.PolicyTerm))

.Where(e => filter.MinPolicyPayment == null || e.PolicyPayment >= filter.MinPolicyPayment)

.Where(e => filter.MaxPolicyPayment == null || e.PolicyPayment <= filter.MaxPolicyPayment)

.Where(e => !filter.PolicyIsActing || (filter.PolicyIsActing && nowTime >= e.ApplicationDate && nowTime <= e.ApplicationDate.AddMonths(e.PolicyTerm)))

.Sort(filter);

}

return result;

}

public IEnumerable<Client> Filter(ClientFilterModel filter) {

var nowTime = DateTime.Now;

var result = \_cache.GetEntity<Client>();

var policyClients = \_cache.GetEntity<PolicyClient>();

if (filter != null) {

return result

.Where(e => filter.Name == null || filter.Name == e.Name)

.Where(e => filter.Surname == null || filter.Surname == e.Surname)

.Where(e => filter.MiddleName == null || filter.MiddleName == e.MiddleName)

.Where(e => filter.PassportNumber == null || e.PassportNumber.Contains(filter.PassportNumber))

.Where(e => filter.Address == null || e.Address.Contains(filter.Address))

.Where(e => filter.MobilePhone == null || e.MobilePhone.Contains(filter.MobilePhone))

.Where(e => filter.BirthdateStart == null || filter.BirthdateStart <= e.Birthdate)

.Where(e => filter.BirthdateEnd == null || filter.BirthdateEnd >= e.Birthdate)

.Where(e => filter.PassportIssueDateStart == null || filter.PassportIssueDateStart <= e.PassportIssueDate)

.Where(e => filter.PassportIssueDateEnd == null || filter.PassportIssueDateEnd >= e.PassportIssueDate)

.Where(e => {

if (!filter.PolicyIsFinishNextMounth) return true;

var policyClient = policyClients.FirstOrDefault(m => m.ClientId == e.Id);

if (policyClient == null) return false;

var dayDiff = (policyClient.Policy.ApplicationDate.AddMonths(policyClient.Policy.PolicyTerm) - nowTime).TotalDays;

return dayDiff < 30 && dayDiff > 0;

})

.Sort(filter);

}

return result;

}

public IEnumerable<InsuranceAgent> Filter(InsuranceAgentFilterModel filter) {

var result = \_cache.GetEntity<InsuranceAgent>();

if (filter != null) {

return result

.Where(e => filter.Name == null || filter.Name == e.Name)

.Where(e => filter.Surname == null || filter.Surname == e.Surname)

.Where(e => filter.MiddleName == null || filter.MiddleName == e.MiddleName)

.Where(e => filter.Responsibilities == null || filter.Responsibilities == e.Contract.Responsibilities)

.Where(e => filter.AgentType == null || filter.AgentType == e.AgentType.Type)

.Where(e => filter.StartDeadLine == null || e.Contract.StartDeadline >= filter.StartDeadLine)

.Where(e => filter.EndDeadLine == null || e.Contract.EndDeadline <= filter.EndDeadLine)

.Where(e => filter.MinSalary == null || e.Contract.Salary >= filter.MinSalary)

.Where(e => filter.MaxSalary == null || e.Contract.Salary <= filter.MaxSalary)

.Where(e => filter.MinTransactionPercent == null || e.Contract.TransactionPercent >= filter.MinTransactionPercent)

.Where(e => filter.MaxTransactionPercent == null || e.Contract.TransactionPercent <= filter.MaxTransactionPercent)

.Sort(filter);

}

return result;

}

public IEnumerable<InsuranceCase> Filter(InsuranceCaseFilterModel filter) {

var result = \_cache.GetEntity<InsuranceCase>();

var policyClients = \_cache.GetEntity<PolicyClient>();

if (filter != null) {

return result

.Where(e => filter.SupportingDocument == null || e.SupportingDocument.Name.Contains(filter.SupportingDocument))

.Where(e => filter.StartDate == null || filter.StartDate <= e.Date)

.Where(e => filter.EndDate == null || filter.EndDate >= e.Date)

.Where(e => filter.MinInsurancePayment == null || e.InsurancePayment >= filter.MinInsurancePayment)

.Where(e => filter.MaxInsurancePayment == null || e.InsurancePayment <= filter.MaxInsurancePayment)

.Where(e => {

if (filter.InsuranceTypeId == null || filter.InsuranceTypeId == -1) return true;

return policyClients

.FirstOrDefault(m => e.ClientId == m.ClientId && m.Policy.InsuranceTypeId == filter.InsuranceTypeId) != null;

})

.Sort(filter);

}

return result;

}

public IEnumerable<SupportingDocument> Filter(SupportingDocumentFilterModel filter) {

var result = \_cache.GetEntity<SupportingDocument>();

if (filter != null) {

return result

.Where(e => filter.Name == null || e.Name.Contains(filter.Name))

.Sort(filter);

}

return result;

}

public IEnumerable<Policy> Filter(PolicyFilterModel filter, int clientId) {

var policiesId = Filter(filter).Select(e => e.Id);

var result = \_cache.GetEntity<PolicyClient>();

if (filter != null) {

return result

.Where(e => e.ClientId == clientId)

.Where(e => policiesId.Contains(e.PolicyId))

.Select(e => e.Policy)

.Sort(filter);

}

return result.Select(e => e.Policy);

}

}

}

Листинг класса *IUpdateCache*.*cs*:

namespace Service {

public interface IUpdateCache {

public void UpdateCache();

}

}

Листинг класса *SortExtensions*.*cs*:

using Models.Models;

using Models.Utilities;

using Models.ViewModels.FilterViewModels;

using System.Linq.Expressions;

namespace Service {

public static class SortExtensions {

public static IEnumerable<InsuranceType> Sort(this IEnumerable<InsuranceType> insuranceTypes, InsuranceTypeFilterModel filter) {

return insuranceTypes

.Sort(filter.SortTypeName, e => e.Name)

.Sort(filter.SortTypeDescription, e => e.Description);

}

public static IEnumerable<Contract> Sort(this IEnumerable<Contract> contracts, ContractFilterModel filter) {

return contracts

.Sort(filter.SortTypeResponsobilities, e => e.Responsibilities)

.Sort(filter.SortTypeTransactionPercent, e => e.TransactionPercent)

.Sort(filter.SortTypeSalary, e => e.Salary)

.Sort(filter.SortTypeContractDuration, e => e.EndDeadline - e.StartDeadline);

}

public static IEnumerable<Policy> Sort(this IEnumerable<Policy> policies, PolicyFilterModel filter) {

return policies

.Sort(filter.SortTypeApplicationDate, e => e.ApplicationDate)

.Sort(filter.SortTypePolicyTerm, e => e.PolicyTerm)

.Sort(filter.SortTypePolicyPaymen, e => e.PolicyPayment);

}

public static IEnumerable<Client> Sort(this IEnumerable<Client> clients, ClientFilterModel filter) {

return clients

.Sort(filter.SortTypeSurname, e => e.Surname)

.Sort(filter.SortTypeName, e => e.Name)

.Sort(filter.SortTypeMiddleName, e => e.MiddleName)

.Sort(filter.SortTypeBirthdate, e => e.Birthdate)

.Sort(filter.SortTypePassportIssueDate, e => e.PassportIssueDate);

}

public static IEnumerable<InsuranceAgent> Sort(this IEnumerable<InsuranceAgent> insuranceAgents, InsuranceAgentFilterModel filter) {

return insuranceAgents

.Sort(filter.SortTypeSurname, e => e.Surname)

.Sort(filter.SortTypeName, e => e.Name)

.Sort(filter.SortTypeMiddleName, e => e.MiddleName)

.Sort(filter.SortTypeSalary, e => e.Contract.Salary)

.Sort(filter.SortTypeTransactionPercent, e => e.Contract.TransactionPercent)

.Sort(filter.SortTypeContractDuration, e => e.Contract.EndDeadline - e.Contract.StartDeadline);

}

public static IEnumerable<InsuranceCase> Sort(this IEnumerable<InsuranceCase> insuranceCase, InsuranceCaseFilterModel filter) {

return insuranceCase

.Sort(filter.SortTypeDate, e => e.Date)

.Sort(filter.SortTypeInsurancePayment, e => e.InsurancePayment)

.Sort(filter.SortTypeSupportingDocument, e => e.SupportingDocument.Name);

}

public static IEnumerable<SupportingDocument> Sort(this IEnumerable<SupportingDocument> supportingDocuments, SupportingDocumentFilterModel filter) {

return supportingDocuments

.Sort(filter.SortTypeName, e => e.Name);

}

private static IEnumerable<T> Sort<T, K>(this IEnumerable<T> entities, SortType sortType, Expression<Func<T, K>> sortExpression) {

entities = sortType switch {

SortType.AscendingSort => entities.AsQueryable().OrderBy(sortExpression),

SortType.DescendingSort => entities.AsQueryable().OrderByDescending(sortExpression),

\_ => entities

};

return entities;

}

}

}

Листинг класса *Program*.*cs*:

using InsuranceCompany.Middleware;

using InsuranceCompany.Services;

using Microsoft.AspNetCore.Identity;

using Microsoft.EntityFrameworkCore;

using Models.Models;

using Repository;

using Service;

var builder = WebApplication.CreateBuilder(args);

string connectionString = builder.Configuration.GetConnectionString("MSSQL");

builder.Services.AddDbContext<InsuranceCompanyContext>(option => option.UseSqlServer(connectionString));

builder.Services.AddDbContext<InsuranceCompanyIdentityContext>(option => option.UseSqlServer(connectionString));

builder.Services.AddDistributedMemoryCache();

builder.Services.AddSession();

builder.Services

.AddDefaultIdentity<ApplicationUser>(options =>

options.User.AllowedUserNameCharacters = "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789-.\_@+=\\/?! абвгдеёжзийклмнопрстуфхцчшщъыьэюяАБВГДЕЁЖЗИЙКЛМНОПРСТУФХЦЧШЩЪЫЬЭЮЯ"

)

.AddDefaultTokenProviders()

.AddRoles<IdentityRole>()

.AddEntityFrameworkStores<InsuranceCompanyIdentityContext>();

builder.Services.AddTransient<InsuranceCompanyCookieManager>();

builder.Services.AddTransient<InsuranceCompanyCache>();

builder.Services.AddTransient<InsuranceCompanyFilter>();

builder.Services.AddMemoryCache();

var app = builder.Build();

if (!app.Environment.IsDevelopment()) {

app.UseExceptionHandler("/Home/Error");

}

app.UseStaticFiles();

app.UseSession();

app.UseDbInitializerMiddleware();

app.UseRouting();

app.UseAuthorization();

app.MapRazorPages();

app.MapControllerRoute(

name: "default",

pattern: "{controller=Home}/{action=Index}/{id?}");

app.Run();

Листинг класса *InsuranceCompanyCookieManager*.*cs*:

using System.Text.Json;

namespace InsuranceCompany.Services {

public class InsuranceCompanyCookieManager {

public void SetCookie<T>(T Entity, IResponseCookies cookies, string key = null) {

var json = JsonSerializer.Serialize(Entity);

key = key == null ? typeof(T).Name : typeof(T).Name+key;

cookies.Append(key, json);

}

public T GetCookie<T>(IRequestCookieCollection cookies, string key = null) where T : new() {

key = key == null ? typeof(T).Name : typeof(T).Name + key;

if (cookies.TryGetValue(key, out var json)) {

return JsonSerializer.Deserialize<T>(json);

}

return new T();

}

}

}

Листинг класса *HtmlHelperExtensions*.*cs*:

using Microsoft.AspNetCore.Html;

using Microsoft.AspNetCore.Mvc.Rendering;

namespace InsuranceCompany.Utilities {

public static class HtmlHelperExtensions {

public static IHtmlContent AddIcon(this IHtmlHelper htmlHelper, int width, int height) {

var svg = $@"

<svg xmlns='http://www.w3.org/2000/svg' width='{width}' height='{height}' fill='currentColor' class='bi bi-plus-circle' viewBox='0 0 16 16'>

<path d='M8 15A7 7 0 1 1 8 1a7 7 0 0 1 0 14m0 1A8 8 0 1 0 8 0a8 8 0 0 0 0 16' />

<path d='M8 4a.5.5 0 0 1 .5.5v3h3a.5.5 0 0 1 0 1h-3v3a.5.5 0 0 1-1 0v-3h-3a.5.5 0 0 1 0-1h3v-3A.5.5 0 0 1 8 4' />

</svg>";

return new HtmlString(svg);

}

public static IHtmlContent FilterIcon(this IHtmlHelper htmlHelper, int width, int height) {

var svg = $@"

<svg xmlns='http://www.w3.org/2000/svg' width='{width}' height='{height}' fill='currentColor' class='bi bi-funnel-fill' viewBox='0 0 16 16'>

<path d=""M1.5 1.5A.5.5 0 0 1 2 1h12a.5.5 0 0 1 .5.5v2a.5.5 0 0 1-.128.334L10 8.692V13.5a.5.5 0 0 1-.342.474l-3 1A.5.5 0 0 1 6 14.5V8.692L1.628 3.834A.5.5 0 0 1 1.5 3.5z"" />

</svg>";

return new HtmlString(svg);

}

public static IHtmlContent DeleteIcon(this IHtmlHelper htmlHelper, int width, int height) {

var svg = $@"

<svg xmlns='http://www.w3.org/2000/svg' width='{width}' height='{height}' fill='currentColor' class='bi bi-trash3' viewBox='0 0 16 16'>

<path d='M6.5 1h3a.5.5 0 0 1 .5.5v1H6v-1a.5.5 0 0 1 .5-.5M11 2.5v-1A1.5 1.5 0 0 0 9.5 0h-3A1.5 1.5 0 0 0 5 1.5v1H2.506a.58.58 0 0 0-.01 0H1.5a.5.5 0 0 0 0 1h.538l.853 10.66A2 2 0 0 0 4.885 16h6.23a2 2 0 0 0 1.994-1.84l.853-10.66h.538a.5.5 0 0 0 0-1h-.995a.59.59 0 0 0-.01 0zm1.958 1-.846 10.58a1 1 0 0 1-.997.92h-6.23a1 1 0 0 1-.997-.92L3.042 3.5zm-7.487 1a.5.5 0 0 1 .528.47l.5 8.5a.5.5 0 0 1-.998.06L5 5.03a.5.5 0 0 1 .47-.53Zm5.058 0a.5.5 0 0 1 .47.53l-.5 8.5a.5.5 0 1 1-.998-.06l.5-8.5a.5.5 0 0 1 .528-.47ZM8 4.5a.5.5 0 0 1 .5.5v8.5a.5.5 0 0 1-1 0V5a.5.5 0 0 1 .5-.5' />

</svg>";

return new HtmlString(svg);

}

public static IHtmlContent InfoIcon(this IHtmlHelper htmlHelper, int width, int height) {

var svg = $@"

<svg xmlns='http://www.w3.org/2000/svg' width='{width}' height='{height}' fill='currentColor' class='bi bi-info-circle' viewBox='0 0 16 16'>

<path d='M8 15A7 7 0 1 1 8 1a7 7 0 0 1 0 14m0 1A8 8 0 1 0 8 0a8 8 0 0 0 0 16' />

<path d='m8.93 6.588-2.29.287-.082.38.45.083c.294.07.352.176.288.469l-.738 3.468c-.194.897.105 1.319.808 1.319.545 0 1.178-.252 1.465-.598l.088-.416c-.2.176-.492.246-.686.246-.275 0-.375-.193-.304-.533zM9 4.5a1 1 0 1 1-2 0 1 1 0 0 1 2 0' />

</svg>";

return new HtmlString(svg);

}

public static IHtmlContent EditIcon(this IHtmlHelper htmlHelper, int width, int height) {

var svg = $@"

<svg xmlns='http://www.w3.org/2000/svg' width='{width}' height='{height}' fill='currentColor' class='bi bi-pencil' viewBox='0 0 16 16'>

<path d='M12.146.146a.5.5 0 0 1 .708 0l3 3a.5.5 0 0 1 0 .708l-10 10a.5.5 0 0 1-.168.11l-5 2a.5.5 0 0 1-.65-.65l2-5a.5.5 0 0 1 .11-.168l10-10zM11.207 2.5 13.5 4.793 14.793 3.5 12.5 1.207zm1.586 3L10.5 3.207 4 9.707V10h.5a.5.5 0 0 1 .5.5v.5h.5a.5.5 0 0 1 .5.5v.5h.293zm-9.761 5.175-.106.106-1.528 3.821 3.821-1.528.106-.106A.5.5 0 0 1 5 12.5V12h-.5a.5.5 0 0 1-.5-.5V11h-.5a.5.5 0 0 1-.468-.325z' />

</svg>";

return new HtmlString(svg);

}

}

}

Листинг класса *DbInitializerMiddleware*.*cs*:

using Microsoft.AspNetCore.Identity;

using Models.Models;

using Repository;

using Repository.DbInitializer;

namespace InsuranceCompany.Middleware {

public class DbInitializerMiddleware {

private readonly RequestDelegate \_next;

public DbInitializerMiddleware(RequestDelegate next) {

\_next = next;

}

public Task Invoke(

HttpContext httpContext,

InsuranceCompanyContext db,

InsuranceCompanyIdentityContext identityDb,

UserManager<ApplicationUser> userManager) {

if (!(httpContext.Session.Keys.Contains("database"))) {

InsuranceCompanyInitializer.Initialize(db, identityDb, userManager);

httpContext.Session.SetString("database", "initial");

}

return \_next.Invoke(httpContext);

}

}

public static class DbInitializerMiddlewareExtensions {

public static IApplicationBuilder UseDbInitializerMiddleware(this IApplicationBuilder builder) {

return builder.UseMiddleware<DbInitializerMiddleware>();

}

}

}

Листинг класса *LoginModel*.*cs*:

using Microsoft.AspNetCore.Authentication;

using Microsoft.AspNetCore.Identity;

using Microsoft.AspNetCore.Mvc;

using Microsoft.AspNetCore.Mvc.RazorPages;

using Models.Models;

using Models.ViewModels.IdentityViewModels;

namespace InsuranceCompany.Areas.Identity.Pages.Account {

public class LoginModel : PageModel {

private readonly SignInManager<ApplicationUser> \_signInManager;

public LoginModel(SignInManager<ApplicationUser> signInManager, RoleManager<IdentityRole> roleManager) {

\_signInManager = signInManager;

}

[BindProperty]

public LoginInputModel Input { get; set; }

public IList<AuthenticationScheme> ExternalLogins { get; set; }

public string ReturnUrl { get; set; }

[TempData]

public string ErrorMessage { get; set; }

public async Task OnGetAsync(string returnUrl = null) {

if (!string.IsNullOrEmpty(ErrorMessage)) {

ModelState.AddModelError(string.Empty, ErrorMessage);

}

returnUrl ??= Url.Content("~/");

await HttpContext.SignOutAsync(IdentityConstants.ExternalScheme);

ExternalLogins = (await \_signInManager.GetExternalAuthenticationSchemesAsync()).ToList();

ReturnUrl = returnUrl;

}

public async Task<IActionResult> OnPostAsync(string returnUrl = null) {

returnUrl ??= Url.Content("~/");

ExternalLogins = (await \_signInManager.GetExternalAuthenticationSchemesAsync()).ToList();

var userName = $"{Input.Name.Trim()} {Input.Surname.Trim()} {Input.MiddleName.Trim()}";

if (ModelState.IsValid) {

var result = await \_signInManager.PasswordSignInAsync(userName, Input.Password, false, lockoutOnFailure: false);

if (result.Succeeded) {

return LocalRedirect(returnUrl);

}

else {

ModelState.AddModelError(string.Empty, "Неверная попытка входа в систему.");

return Page();

}

}

return Page();

}

}

}

Листинг класса *LogoutModel*.*cs*:

using Microsoft.AspNetCore.Identity;

using Microsoft.AspNetCore.Mvc;

using Microsoft.AspNetCore.Mvc.RazorPages;

using Models.Models;

namespace InsuranceCompany.Areas.Identity.Pages.Account {

public class LogoutModel : PageModel {

private readonly SignInManager<ApplicationUser> \_signInManager;

public LogoutModel(SignInManager<ApplicationUser> signInManager) {

\_signInManager = signInManager;

}

public async Task<IActionResult> OnPost(string returnUrl = null) {

await \_signInManager.SignOutAsync();

if (returnUrl != null) {

return LocalRedirect(returnUrl);

}

else {

return RedirectToPage();

}

}

}

}

Листинг класса *RegisterModel*.*cs*:

using Microsoft.AspNetCore.Authentication;

using Microsoft.AspNetCore.Identity;

using Microsoft.AspNetCore.Mvc;

using Microsoft.AspNetCore.Mvc.RazorPages;

using Microsoft.AspNetCore.Mvc.Rendering;

using Models.Models;

using Models.ViewModels.IdentityViewModels;

using Repository;

using Service;

namespace InsuranceCompany.Areas.Identity.Pages.Account {

public class RegisterModel : PageModel {

private readonly SignInManager<ApplicationUser> \_signInManager;

private readonly UserManager<ApplicationUser> \_userManager;

private readonly IUserStore<ApplicationUser> \_userStore;

private readonly RoleManager<IdentityRole> \_roleManager;

private readonly InsuranceCompanyContext \_db;

private readonly InsuranceCompanyCache \_cache;

public RegisterModel(

UserManager<ApplicationUser> userManager,

IUserStore<ApplicationUser> userStore,

SignInManager<ApplicationUser> signInManager,

RoleManager<IdentityRole> roleManager,

InsuranceCompanyContext db,

InsuranceCompanyCache cache) {

\_userManager = userManager;

\_userStore = userStore;

\_signInManager = signInManager;

\_roleManager = roleManager;

\_db = db;

\_cache = cache;

}

[BindProperty]

public RegisterInputModel Input { get; set; }

[BindProperty]

public ClientInputModel ClientInput { get; set; }

[BindProperty]

public InsuranceAgentInputModel InsuranceAgentInput { get; set; }

public string ReturnUrl { get; set; }

public IList<AuthenticationScheme> ExternalLogins { get; set; }

public async Task OnGetAsync(string returnUrl = null) {

ReturnUrl = returnUrl;

ExternalLogins = (await \_signInManager.GetExternalAuthenticationSchemesAsync()).ToList();

Input = new RegisterInputModel() { RoleList = GetRoleList() };

InsuranceAgentInput = new InsuranceAgentInputModel() {

AgentTypeList = GetAgentTypeList(),

ContractList = GetContractList()

};

ClientInput = new ClientInputModel() { BrithDate = DateTime.Now, PassportIssueDate = DateTime.Now };

}

public async Task<IActionResult> OnPostAsync(string returnUrl = null) {

returnUrl ??= Url.Content("~/");

ExternalLogins = (await \_signInManager.GetExternalAuthenticationSchemesAsync()).ToList();

if ((Input.Role == "Клиент" && ModelState.ErrorCount <= 2) || (Input.Role == "Страховой агент" && ModelState.ErrorCount <= 3)) {

var user = CreateUser();

var userName = $"{Input.Name.Trim()} {Input.Surname.Trim()} {Input.MiddleName.Trim()}";

await \_userStore.SetUserNameAsync(user, userName, CancellationToken.None);

user.Name = Input.Name;

user.Surname = Input.Surname;

user.MiddleName = Input.MiddleName;

var result = await \_userManager.CreateAsync(user, Input.Password);

if (result.Succeeded) {

await \_userManager.AddToRoleAsync(user, Input.Role);

if (Input.Role == "Клиент")

AddClient();

else if (Input.Role == "Страховой агент")

AddInsuranceAgent();

\_db.SaveChanges();

if (Input.Role == "Клиент") \_cache.SetEntity<Client>();

else \_cache.SetEntity<InsuranceAgent>();

await \_signInManager.SignInAsync(user, isPersistent: false);

return LocalRedirect(returnUrl);

}

foreach (var error in result.Errors) {

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

}

}

Input.RoleList = GetRoleList();

InsuranceAgentInput.ContractList = GetContractList();

InsuranceAgentInput.AgentTypeList = GetAgentTypeList();

return Page();

}

private void AddClient() {

\_db.Add(new Client() {

Name = Input.Name,

Surname = Input.Surname,

MiddleName = Input.MiddleName,

Birthdate = ClientInput.BrithDate,

Address = ClientInput.Address,

MobilePhone = ClientInput.MobilePhone,

PassportNumber = ClientInput.PassportNumber,

PassportIssueDate = ClientInput.PassportIssueDate

});

}

private void AddInsuranceAgent() {

\_db.Add(new InsuranceAgent() {

Name = Input.Name,

Surname = Input.Surname,

MiddleName = Input.MiddleName,

AgentTypeId = int.Parse(InsuranceAgentInput.AgentType),

ContractId = int.Parse(InsuranceAgentInput.Contract)

});

}

private IEnumerable<SelectListItem> GetRoleList() {

return \_roleManager.Roles.Select(e => new SelectListItem() { Text = e.Name, Value = e.Name });

}

private IEnumerable<SelectListItem> GetAgentTypeList() {

return \_db.AgentTypes.Select(e => new SelectListItem() { Text = e.Type, Value = e.Id.ToString() }).ToList();

}

private IEnumerable<SelectListItem> GetContractList() {

return \_db.Contracts.Select(e => new SelectListItem() { Text = $"{e.Responsibilities}\n({e.StartDeadline.ToShortDateString()} - {e.EndDeadline.ToShortDateString()})", Value = e.Id.ToString() }).ToList();

}

private ApplicationUser CreateUser() {

try {

return Activator.CreateInstance<ApplicationUser>();

}

catch {

throw new InvalidOperationException($"Can't create an instance of '{nameof(ApplicationUser)}'. " +

$"Ensure that '{nameof(ApplicationUser)}' is not an abstract class and has a parameterless constructor, or alternatively " +

$"override the register page in /Areas/Identity/Pages/Account/Register.cshtml");

}

}

}

}

Листинг класса *BaseController*.*cs*:

using InsuranceCompany.Services;

using Microsoft.AspNetCore.Mvc;

using Repository;

using Service;

namespace InsuranceCompany.Controllers {

public class BaseController : Controller {

protected readonly InsuranceCompanyContext \_context;

protected readonly InsuranceCompanyCache \_cache;

protected readonly InsuranceCompanyCookieManager \_cookieManager;

protected readonly InsuranceCompanyFilter \_filter;

public BaseController(

InsuranceCompanyContext context,

InsuranceCompanyCache cache,

InsuranceCompanyCookieManager cookieManager,

InsuranceCompanyFilter filter) {

\_context = context;

\_cache = cache;

\_cookieManager = cookieManager;

\_filter = filter;

}

}

}

Листинг класса *ClientInsuranceCasesController*.*cs*:

using InsuranceCompany.Services;

using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.Identity;

using Microsoft.AspNetCore.Mvc;

using Microsoft.AspNetCore.Mvc.Rendering;

using Microsoft.EntityFrameworkCore;

using Models.Models;

using Models.ViewModels.FilterViewModels;

using Models.ViewModels.PageViewModels;

using Repository;

using Service;

namespace InsuranceCompany.Controllers {

[Authorize(Roles = "Клиент")]

public class ClientInsuranceCasesController : BaseController, IUpdateCache {

private UserManager<ApplicationUser> \_userManager;

private const int PAGE\_SIZE = 9;

public ClientInsuranceCasesController(

InsuranceCompanyContext context,

InsuranceCompanyCache cache,

InsuranceCompanyCookieManager cookieManager,

InsuranceCompanyFilter filter,

UserManager<ApplicationUser> userManager) : base(context, cache, cookieManager, filter) {

\_userManager = userManager;

}

public async Task<IActionResult> Index(int page, int pageSize = PAGE\_SIZE) {

var insuranceCaseFilter = \_cookieManager.GetCookie<InsuranceCaseFilterModel>(HttpContext.Request.Cookies, "client");

insuranceCaseFilter.InsuranceTypeList = GetInsuranceTypeList();

var client = GetClient();

var insuranceCases = \_filter.Filter(insuranceCaseFilter).Where(e => e.ClientId == client.Id);

var viewModel = new PageModel<InsuranceCase, InsuranceCaseFilterModel>(page, pageSize, insuranceCases, insuranceCaseFilter);

return View(viewModel);

}

[HttpPost]

public async Task<IActionResult> Index(InsuranceCaseFilterModel insuranceCaseFilter, int page, int pageSize = PAGE\_SIZE) {

\_cookieManager.SetCookie(insuranceCaseFilter, HttpContext.Response.Cookies, "client");

insuranceCaseFilter.InsuranceTypeList = GetInsuranceTypeList();

var client = GetClient();

var insuranceCases = \_filter.Filter(insuranceCaseFilter).Where(e => e.ClientId == client.Id);

var viewModel = new PageModel<InsuranceCase, InsuranceCaseFilterModel>(page, pageSize, insuranceCases, insuranceCaseFilter);

return View(viewModel);

}

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

if (id == null) {

return NotFound();

}

var insuranceCase = \_cache.GetEntity<InsuranceCase>().FirstOrDefault(m => m.Id == id);

if (insuranceCase == null) {

return NotFound();

}

return View(insuranceCase);

}

public IActionResult Create() {

ViewData["SupportingDocumentId"] = GetSupportingDocumentsList();

return View();

}

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<IActionResult> Create(InsuranceCase insuranceCase) {

var client = GetClient();

insuranceCase.ClientId = client.Id;

if (ModelState.IsValid) {

\_context.Add(insuranceCase);

await \_context.SaveChangesAsync();

UpdateCache();

return RedirectToAction(nameof(Index));

}

ViewData["SupportingDocumentId"] = GetSupportingDocumentsList();

return View(insuranceCase);

}

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

if (id == null) {

return NotFound();

}

var insuranceCase = \_cache.GetEntity<InsuranceCase>().FirstOrDefault(e => e.Id == id);

if (insuranceCase == null) {

return NotFound();

}

ViewData["SupportingDocumentId"] = GetSupportingDocumentsList();

return View(insuranceCase);

}

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<IActionResult> Edit(int id, InsuranceCase insuranceCase) {

if (id != insuranceCase.Id) {

return NotFound();

}

var client = GetClient();

insuranceCase.ClientId = client.Id;

if (ModelState.IsValid) {

try {

\_context.Update(insuranceCase);

UpdateCache();

await \_context.SaveChangesAsync();

}

catch (DbUpdateConcurrencyException) {

return NotFound();

}

return RedirectToAction(nameof(Index));

}

ViewData["SupportingDocumentId"] = GetSupportingDocumentsList();

return View(insuranceCase);

}

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

if (id == null) {

return NotFound();

}

var insuranceCase = \_cache.GetEntity<InsuranceCase>().FirstOrDefault(m => m.Id == id);

if (insuranceCase == null) {

return NotFound();

}

return View(insuranceCase);

}

[HttpPost, ActionName("Delete")]

[ValidateAntiForgeryToken]

public async Task<IActionResult> DeleteConfirmed(int id) {

var insuranceCase = \_cache.GetEntity<InsuranceCase>().FirstOrDefault(e => e.Id == id);

if (insuranceCase != null) {

\_context.InsuranceCases.Remove(insuranceCase);

}

await \_context.SaveChangesAsync();

UpdateCache();

return RedirectToAction(nameof(Index));

}

private IEnumerable<SelectListItem> GetSupportingDocumentsList() {

return \_cache.GetEntity<SupportingDocument>()

.Select(e => new SelectListItem() {

Text = e.Name,

Value = e.Id.ToString()

});

}

private IEnumerable<SelectListItem> GetInsuranceTypeList() {

return \_cache.GetEntity<InsuranceType>()

.Select(e => new SelectListItem() {

Text = e.Name,

Value = e.Id.ToString()

});

}

private Client GetClient() {

var applicationUser = \_userManager.GetUserAsync(HttpContext.User).Result;

return \_cache.GetEntity<Client>()

.FirstOrDefault(e =>

applicationUser.Name == e.Name &&

applicationUser.Surname == e.Surname &&

applicationUser.MiddleName == e.MiddleName);

}

public void UpdateCache() {

\_cache.SetEntity<InsuranceCase>();

}

}

}

Листинг класса *ClientManagerController*.*cs*:

using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.Identity;

using Microsoft.AspNetCore.Mvc;

using Microsoft.EntityFrameworkCore;

using Models.Models;

using Models.ViewModels.IdentityViewModels;

using Repository;

using Service;

using System.Data;

namespace InsuranceCompany.Controllers {

[Authorize(Roles = "Клиент")]

public class ClientManagerController : Controller, IUpdateCache {

private readonly InsuranceCompanyContext \_context;

private readonly InsuranceCompanyCache \_cache;

private readonly UserManager<ApplicationUser> \_userManager;

public ClientManagerController(

InsuranceCompanyContext context,

InsuranceCompanyCache cache,

UserManager<ApplicationUser> userManager) {

\_context = context;

\_cache = cache;

\_userManager = userManager;

}

public IActionResult Index() {

var client = GetClient();

client.InsuranceCases = \_cache

.GetEntity<InsuranceCase>()

.Where(e => e.ClientId == client.Id).ToList();

var policies = \_cache

.GetEntity<PolicyClient>()

.Where(e => e.ClientId == client.Id)

.Select(e => e.Policy);

return View((client, policies));

}

[HttpPost]

public async Task<IActionResult> Edit(Client client) {

if (ModelState.IsValid) {

try {

var oldClient = GetClient();

var oldUserName = $"{oldClient.Name} {oldClient.Surname} {oldClient.MiddleName}";

var newUserName = $"{client.Name} {client.Surname} {client.MiddleName}";

var applicationUser = \_userManager.Users.FirstOrDefault(e => e.UserName == oldUserName);

applicationUser.Name = client.Name;

applicationUser.Surname = client.Surname;

applicationUser.MiddleName = client.MiddleName;

applicationUser.UserName = newUserName;

client.Id = oldClient.Id;

await \_userManager.UpdateAsync(applicationUser);

\_context.Update(client);

await \_context.SaveChangesAsync();

UpdateCache();

}

catch (DbUpdateConcurrencyException) {

return NotFound();

}

return RedirectToAction(nameof(Index));

}

return View(client);

}

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<ActionResult> ChangePassword(ChangePasswordViewModel model) {

var user = await \_userManager.GetUserAsync(HttpContext.User);

var result = await \_userManager.ChangePasswordAsync(user, model.OldPassword, model.NewPassword);

return RedirectToAction(nameof(Index));

}

private Client GetClient() {

var applicationUser = \_userManager.GetUserAsync(HttpContext.User).Result;

return \_cache.GetEntity<Client>()

.FirstOrDefault(e =>

applicationUser.Name == e.Name &&

applicationUser.Surname == e.Surname &&

applicationUser.MiddleName == e.MiddleName);

}

public void UpdateCache() {

\_cache.SetEntity<Client>();

\_cache.SetEntity<InsuranceCase>();

\_cache.SetEntity<SupportingDocument>();

\_cache.SetEntity<PolicyClient>();

}

}

}

Листинг класса *ClientsController*.*cs*:

using InsuranceCompany.Services;

using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.Identity;

using Microsoft.AspNetCore.Mvc;

using Microsoft.EntityFrameworkCore;

using Models.Models;

using Models.ViewModels.FilterViewModels;

using Models.ViewModels.PageViewModels;

using Repository;

using Service;

namespace InsuranceCompany.Controllers {

[Authorize(Roles = "Страховой агент")]

public class ClientsController : BaseController, IUpdateCache {

private readonly UserManager<ApplicationUser> \_userManager;

private const int PAGE\_SIZE = 9;

public ClientsController(

InsuranceCompanyContext context,

InsuranceCompanyCache cache,

InsuranceCompanyCookieManager cookieManager,

InsuranceCompanyFilter filter,

UserManager<ApplicationUser> userManager) : base(context, cache, cookieManager, filter) {

\_userManager = userManager;

}

public async Task<IActionResult> Index(int page, int pageSize = PAGE\_SIZE) {

var clientFilter = \_cookieManager.GetCookie<ClientFilterModel>(HttpContext.Request.Cookies);

var clients = \_filter.Filter(clientFilter);

var viewModel = new PageModel<Client, ClientFilterModel>(page, pageSize, clients, clientFilter);

return View(viewModel);

}

[HttpPost]

public async Task<IActionResult> Index(ClientFilterModel clientFilter, int page, int pageSize = PAGE\_SIZE) {

\_cookieManager.SetCookie(clientFilter, HttpContext.Response.Cookies);

var clients = \_filter.Filter(clientFilter);

var viewModel = new PageModel<Client, ClientFilterModel>(page, pageSize, clients, clientFilter);

return View(viewModel);

}

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

if (id == null) {

return NotFound();

}

var client = \_cache.GetEntity<Client>().FirstOrDefault(m => m.Id == id);

if (client == null) {

return NotFound();

}

client.InsuranceCases = \_cache.GetEntity<InsuranceCase>().Where(m => m.ClientId == id).ToList();

var policies = \_cache.GetEntity<PolicyClient>().Where(e => e.ClientId == id).Select(e => e.Policy);

var viewModel = (client, policies);

return View(viewModel);

}

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

if (id == null) {

return NotFound();

}

var client = \_cache.GetEntity<Client>().FirstOrDefault(m => m.Id == id);

if (client == null) {

return NotFound();

}

return View(client);

}

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<IActionResult> Edit(int id, Client client) {

if (id != client.Id) {

return NotFound();

}

if (ModelState.IsValid) {

try {

var oldClient = \_cache.GetEntity<Client>().FirstOrDefault(e => e.Id == id);

var oldUserName = $"{oldClient.Name} {oldClient.Surname} {oldClient.MiddleName}";

var newUserName = $"{client.Name} {client.Surname} {client.MiddleName}";

var applicationUser = \_userManager.Users.FirstOrDefault(e => e.UserName == oldUserName);

applicationUser.Name = client.Name;

applicationUser.Surname = client.Surname;

applicationUser.MiddleName = client.MiddleName;

applicationUser.UserName = newUserName;

await \_userManager.UpdateAsync(applicationUser);

\_context.Update(client);

await \_context.SaveChangesAsync();

UpdateCache();

}

catch (DbUpdateConcurrencyException) {

return NotFound();

}

return RedirectToAction(nameof(Index));

}

return View(client);

}

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

if (id == null) {

return NotFound();

}

var client = \_cache.GetEntity<Client>().FirstOrDefault(m => m.Id == id);

if (client == null) {

return NotFound();

}

client.InsuranceCases = \_cache.GetEntity<InsuranceCase>().Where(e => e.ClientId == id).ToList();

return View(client);

}

[HttpPost, ActionName("Delete")]

[ValidateAntiForgeryToken]

public async Task<IActionResult> DeleteConfirmed(int id) {

var client = \_cache.GetEntity<Client>().FirstOrDefault(m => m.Id == id);

if (client != null) {

\_context.Clients.Remove(client);

var userName = $"{client.Name} {client.Surname} {client.MiddleName}";

var applicationUser = \_userManager.Users.FirstOrDefault(e => e.UserName == userName);

await \_userManager.DeleteAsync(applicationUser);

}

await \_context.SaveChangesAsync();

UpdateCache();

return RedirectToAction(nameof(Index));

}

public void UpdateCache() {

\_cache.SetEntity<Client>();

\_cache.SetEntity<InsuranceCase>();

\_cache.SetEntity<SupportingDocument>();

\_cache.SetEntity<PolicyClient>();

}

}

}

Листинг класса *ContractsController*.*cs*:

using InsuranceCompany.Services;

using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.Mvc;

using Microsoft.EntityFrameworkCore;

using Models.Models;

using Models.ViewModels.FilterViewModels;

using Models.ViewModels.PageViewModels;

using Repository;

using Service;

namespace InsuranceCompany.Controllers {

[Authorize(Roles = "Страховой агент")]

public class ContractsController : BaseController, IUpdateCache {

private const int PAGE\_SIZE = 9;

public ContractsController(

InsuranceCompanyContext context,

InsuranceCompanyCache cache,

InsuranceCompanyCookieManager cookieManager,

InsuranceCompanyFilter filter) : base(context, cache, cookieManager, filter) {

}

public async Task<IActionResult> Index(int page, int pageSize = PAGE\_SIZE) {

var contractFilter = \_cookieManager.GetCookie<ContractFilterModel>(HttpContext.Request.Cookies);

var contracts = \_filter.Filter(contractFilter);

var viewModel = new PageModel<Contract, ContractFilterModel>(page, pageSize, contracts, contractFilter);

return View(viewModel);

}

[HttpPost]

public async Task<IActionResult> Index(ContractFilterModel contractFilter, int page, int pageSize = PAGE\_SIZE) {

\_cookieManager.SetCookie(contractFilter, HttpContext.Response.Cookies);

var contracts = \_filter.Filter(contractFilter);

var viewModel = new PageModel<Contract, ContractFilterModel>(page, pageSize, contracts, contractFilter);

return View(viewModel);

}

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

if (id == null) {

return NotFound();

}

var contract = \_cache.GetEntity<Contract>().FirstOrDefault(m => m.Id == id);

if (contract == null) {

return NotFound();

}

contract.InsuranceAgents = \_cache.GetEntity<InsuranceAgent>().Where(e => e.ContractId == id).ToList();

return View(contract);

}

public IActionResult Create() {

return View();

}

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<IActionResult> Create(Contract contract) {

if (ModelState.IsValid) {

\_context.Add(contract);

await \_context.SaveChangesAsync();

UpdateCache();

return RedirectToAction(nameof(Index));

}

return View(contract);

}

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

if (id == null) {

return NotFound();

}

var contract = \_cache.GetEntity<Contract>().FirstOrDefault(e => e.Id == id);

if (contract == null) {

return NotFound();

}

return View(contract);

}

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<IActionResult> Edit(int id, Contract contract) {

if (id != contract.Id) {

return NotFound();

}

if (ModelState.IsValid) {

try {

\_context.Update(contract);

await \_context.SaveChangesAsync();

UpdateCache();

}

catch (DbUpdateConcurrencyException) {

return NotFound();

}

return RedirectToAction(nameof(Index));

}

return View(contract);

}

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

if (id == null) {

return NotFound();

}

var contract = \_cache.GetEntity<Contract>().FirstOrDefault(e => e.Id == id);

if (contract == null) {

return NotFound();

}

return View(contract);

}

[HttpPost, ActionName("Delete")]

[ValidateAntiForgeryToken]

public async Task<IActionResult> DeleteConfirmed(int id) {

var contract = \_cache.GetEntity<Contract>().FirstOrDefault(e => e.Id == id);

if (contract != null) {

\_context.Contracts.Remove(contract);

}

await \_context.SaveChangesAsync();

UpdateCache();

return RedirectToAction(nameof(Index));

}

public void UpdateCache() {

\_cache.SetEntity<Contract>();

\_cache.SetEntity<InsuranceAgent>();

\_cache.SetEntity<Policy>();

\_cache.SetEntity<PolicyClient>();

}

}

}

Листинг класса *HomeController*.*cs*:

using Microsoft.AspNetCore.Mvc;

using Models.Models;

using Service;

namespace InsuranceCompany.Controllers {

public class HomeController : Controller {

private readonly InsuranceCompanyCache \_cache;

public HomeController(InsuranceCompanyCache cache) {

\_cache = cache;

}

public IActionResult Index() {

return View(\_cache.GetEntity<InsuranceType>());

}

public IActionResult Privacy() {

return View();

}

}

}

Листинг класса *InsuranceAgentManagerController*.*cs*:

using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.Identity;

using Microsoft.AspNetCore.Mvc;

using Microsoft.AspNetCore.Mvc.Rendering;

using Microsoft.EntityFrameworkCore;

using Models.Models;

using Models.ViewModels.IdentityViewModels;

using Repository;

using Service;

namespace InsuranceCompany.Controllers {

[Authorize(Roles = "Страховой агент")]

public class InsuranceAgentManagerController : Controller, IUpdateCache {

private readonly InsuranceCompanyContext \_context;

private readonly InsuranceCompanyCache \_cache;

private readonly UserManager<ApplicationUser> \_userManager;

public InsuranceAgentManagerController(

InsuranceCompanyContext context,

InsuranceCompanyCache cache,

UserManager<ApplicationUser> userManager) {

\_context = context;

\_cache = cache;

\_userManager = userManager;

}

public IActionResult Index() {

ViewData["AgentTypeId"] = GetAgentTypesList();

ViewData["ContractId"] = GetContractsList();

var insuranceAgent = GetInsuranceAgent();

insuranceAgent.Policies = \_cache

.GetEntity<Policy>()

.Where(e => e.InsuranceAgentId == insuranceAgent.Id).ToList();

var clients = \_cache

.GetEntity<PolicyClient>()

.Where(e => e.Policy.InsuranceAgentId == insuranceAgent.Id)

.Select(e => e.Client);

return View((insuranceAgent, clients));

}

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<IActionResult> Edit(InsuranceAgent insuranceAgent) {

if (ModelState.IsValid) {

try {

var oldInsuranceAgent = GetInsuranceAgent();

var oldUserName = $"{oldInsuranceAgent.Name} {oldInsuranceAgent.Surname} {oldInsuranceAgent.MiddleName}";

var newUserName = $"{insuranceAgent.Name} {insuranceAgent.Surname} {insuranceAgent.MiddleName}";

var applicationUser = \_userManager.Users.FirstOrDefault(e => e.UserName == oldUserName);

applicationUser.Name = insuranceAgent.Name;

applicationUser.Surname = insuranceAgent.Surname;

applicationUser.MiddleName = insuranceAgent.MiddleName;

applicationUser.UserName = newUserName;

insuranceAgent.Id = oldInsuranceAgent.Id;

await \_userManager.UpdateAsync(applicationUser);

\_context.Update(insuranceAgent);

await \_context.SaveChangesAsync();

UpdateCache();

}

catch (DbUpdateConcurrencyException) {

return NotFound();

}

return RedirectToAction(nameof(Index));

}

ViewData["AgentTypeId"] = GetAgentTypesList();

ViewData["ContractId"] = GetContractsList();

return View(insuranceAgent);

}

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<ActionResult> ChangePassword(ChangePasswordViewModel model) {

var user = await \_userManager.GetUserAsync(HttpContext.User);

var result = await \_userManager.ChangePasswordAsync(user, model.OldPassword, model.NewPassword);

return RedirectToAction(nameof(Index));

}

private InsuranceAgent GetInsuranceAgent() {

var applicationUser = \_userManager.GetUserAsync(HttpContext.User).Result;

return \_cache.GetEntity<InsuranceAgent>()

.FirstOrDefault(e =>

applicationUser.Name == e.Name &&

applicationUser.Surname == e.Surname &&

applicationUser.MiddleName == e.MiddleName);

}

private IEnumerable<SelectListItem> GetContractsList() {

return \_cache.GetEntity<Contract>().Select(e => new SelectListItem() {

Text = $"{e.Responsibilities} ({e.StartDeadline.ToShortDateString()} - {e.EndDeadline.ToShortDateString()})",

Value = e.Id.ToString()

});

}

private IEnumerable<SelectListItem> GetAgentTypesList() {

return \_cache.GetEntity<AgentType>().Select(e => new SelectListItem() {

Text = e.Type,

Value = e.Id.ToString()

});

}

public void UpdateCache() {

\_cache.SetEntity<InsuranceAgent>();

\_cache.SetEntity<Policy>();

\_cache.SetEntity<PolicyClient>();

}

}

}

Листинг класса *InsuranceAgentsController*.*cs*:

using InsuranceCompany.Services;

using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.Identity;

using Microsoft.AspNetCore.Mvc;

using Microsoft.AspNetCore.Mvc.Rendering;

using Microsoft.EntityFrameworkCore;

using Models.Models;

using Models.ViewModels.FilterViewModels;

using Models.ViewModels.PageViewModels;

using Repository;

using Service;

namespace InsuranceCompany.Controllers {

[Authorize(Roles = "Страховой агент")]

public class InsuranceAgentsController : BaseController, IUpdateCache {

private readonly UserManager<ApplicationUser> \_userManager;

private const int PAGE\_SIZE = 9;

public InsuranceAgentsController(

InsuranceCompanyContext context,

InsuranceCompanyCache cache,

InsuranceCompanyCookieManager cookieManager,

InsuranceCompanyFilter filter,

UserManager<ApplicationUser> userManager) : base(context, cache, cookieManager, filter) {

\_userManager = userManager;

}

public async Task<IActionResult> Index(int page, int pageSize = PAGE\_SIZE) {

var insuranceAgentFilter = \_cookieManager.GetCookie<InsuranceAgentFilterModel>(HttpContext.Request.Cookies);

var insuranceAgents = \_filter.Filter(insuranceAgentFilter);

var viewModel = new PageModel<InsuranceAgent, InsuranceAgentFilterModel>(page, pageSize, insuranceAgents, insuranceAgentFilter);

return View(viewModel);

}

[HttpPost]

public async Task<IActionResult> Index(InsuranceAgentFilterModel insuranceAgentFilter, int page, int pageSize = PAGE\_SIZE) {

\_cookieManager.SetCookie(insuranceAgentFilter, HttpContext.Response.Cookies);

var insuranceAgents = \_filter.Filter(insuranceAgentFilter);

var viewModel = new PageModel<InsuranceAgent, InsuranceAgentFilterModel>(page, pageSize, insuranceAgents, insuranceAgentFilter);

return View(viewModel);

}

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

if (id == null) {

return NotFound();

}

var insuranceAgent = \_cache.GetEntity<InsuranceAgent>().FirstOrDefault(m => m.Id == id);

if (insuranceAgent == null) {

return NotFound();

}

insuranceAgent.Policies = \_cache.GetEntity<Policy>().Where(e => e.InsuranceAgentId == id).ToList();

return View(insuranceAgent);

}

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

if (id == null) {

return NotFound();

}

var insuranceAgent = \_cache.GetEntity<InsuranceAgent>().FirstOrDefault(m => m.Id == id);

if (insuranceAgent == null) {

return NotFound();

}

ViewData["AgentTypeId"] = GetAgentTypesList();

ViewData["ContractId"] = GetContractsList();

return View(insuranceAgent);

}

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<IActionResult> Edit(int id, InsuranceAgent insuranceAgent) {

if (id != insuranceAgent.Id) {

return NotFound();

}

if (ModelState.IsValid) {

try {

var oldInsuranceAgent = \_cache.GetEntity<InsuranceAgent>().FirstOrDefault(e => e.Id == id);

var oldUserName = $"{oldInsuranceAgent.Name} {oldInsuranceAgent.Surname} {oldInsuranceAgent.MiddleName}";

var newUserName = $"{insuranceAgent.Name} {insuranceAgent.Surname} {insuranceAgent.MiddleName}";

var applicationUser = \_userManager.Users.FirstOrDefault(e => e.UserName == oldUserName);

applicationUser.Name = insuranceAgent.Name;

applicationUser.Surname = insuranceAgent.Surname;

applicationUser.MiddleName = insuranceAgent.MiddleName;

applicationUser.UserName = newUserName;

await \_userManager.UpdateAsync(applicationUser);

\_context.Update(insuranceAgent);

await \_context.SaveChangesAsync();

UpdateCache();

}

catch (DbUpdateConcurrencyException) {

return NotFound();

}

return RedirectToAction(nameof(Index));

}

ViewData["AgentTypeId"] = GetAgentTypesList();

ViewData["ContractId"] = GetContractsList();

return View(insuranceAgent);

}

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

if (id == null) {

return NotFound();

}

var insuranceAgent = \_cache.GetEntity<InsuranceAgent>().FirstOrDefault(m => m.Id == id);

if (insuranceAgent == null) {

return NotFound();

}

return View(insuranceAgent);

}

[HttpPost, ActionName("Delete")]

[ValidateAntiForgeryToken]

public async Task<IActionResult> DeleteConfirmed(int id) {

var insuranceAgent = \_cache.GetEntity<InsuranceAgent>().FirstOrDefault(m => m.Id == id);

if (insuranceAgent != null) {

\_context.InsuranceAgents.Remove(insuranceAgent);

var userName = $"{insuranceAgent.Name} {insuranceAgent.Surname} {insuranceAgent.MiddleName}";

var applicationUser = \_userManager.Users.FirstOrDefault(e => e.UserName == userName);

await \_userManager.DeleteAsync(applicationUser);

}

await \_context.SaveChangesAsync();

UpdateCache();

return RedirectToAction(nameof(Index));

}

private IEnumerable<SelectListItem> GetContractsList() {

return \_cache.GetEntity<Contract>().Select(e => new SelectListItem() {

Text = $"{e.Responsibilities} ({e.StartDeadline.ToShortDateString()} - {e.EndDeadline.ToShortDateString()})",

Value = e.Id.ToString()

});

}

private IEnumerable<SelectListItem> GetAgentTypesList() {

return \_cache.GetEntity<AgentType>().Select(e => new SelectListItem() {

Text = e.Type,

Value = e.Id.ToString()

});

}

public void UpdateCache() {

\_cache.SetEntity<InsuranceAgent>();

\_cache.SetEntity<Policy>();

\_cache.SetEntity<PolicyClient>();

}

}

}

Листинг класса *InsuranceCasesController*.*cs*:

using InsuranceCompany.Services;

using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.Mvc;

using Microsoft.AspNetCore.Mvc.Rendering;

using Models.Models;

using Models.ViewModels.FilterViewModels;

using Models.ViewModels.PageViewModels;

using Service;

using System.Data;

namespace InsuranceCompany.Controllers {

[Authorize(Roles = "Страховой агент")]

public class InsuranceCasesController : Controller {

private readonly InsuranceCompanyCache \_cache;

private readonly InsuranceCompanyCookieManager \_cookieManager;

private readonly InsuranceCompanyFilter \_filter;

private const int PAGE\_SIZE = 9;

public InsuranceCasesController(

InsuranceCompanyCache cache,

InsuranceCompanyCookieManager cookieManager,

InsuranceCompanyFilter filter) {

\_cache = cache;

\_cookieManager = cookieManager;

\_filter = filter;

}

public async Task<IActionResult> Index(int page, int pageSize = PAGE\_SIZE) {

var insuranceCaseFilter = \_cookieManager.GetCookie<InsuranceCaseFilterModel>(HttpContext.Request.Cookies);

insuranceCaseFilter.InsuranceTypeList = GetInsuranceTypeList();

var insuranceCases = \_filter.Filter(insuranceCaseFilter);

var viewModel = new PageModel<InsuranceCase, InsuranceCaseFilterModel>(page, pageSize, insuranceCases, insuranceCaseFilter);

return View(viewModel);

}

[HttpPost]

public async Task<IActionResult> Index(InsuranceCaseFilterModel insuranceCaseFilter, int page, int pageSize = PAGE\_SIZE) {

\_cookieManager.SetCookie(insuranceCaseFilter, HttpContext.Response.Cookies);

insuranceCaseFilter.InsuranceTypeList = GetInsuranceTypeList();

var insuranceCases = \_filter.Filter(insuranceCaseFilter);

var viewModel = new PageModel<InsuranceCase, InsuranceCaseFilterModel>(page, pageSize, insuranceCases, insuranceCaseFilter);

return View(viewModel);

}

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

if (id == null) {

return NotFound();

}

var insuranceCase = \_cache.GetEntity<InsuranceCase>().FirstOrDefault(m => m.Id == id);

if (insuranceCase == null) {

return NotFound();

}

return View(insuranceCase);

}

private IEnumerable<SelectListItem> GetInsuranceTypeList() {

return \_cache.GetEntity<InsuranceType>()

.Select(e => new SelectListItem() {

Text = e.Name,

Value = e.Id.ToString()

});

}

}

}

Листинг класса *InsuranceTypesController*.*cs*:

using InsuranceCompany.Services;

using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.Mvc;

using Microsoft.EntityFrameworkCore;

using Models.Models;

using Models.ViewModels.FilterViewModels;

using Models.ViewModels.PageViewModels;

using Repository;

using Service;

namespace InsuranceCompany.Controllers {

[Authorize(Roles = "Страховой агент")]

public class InsuranceTypesController : BaseController, IUpdateCache {

private const int PAGE\_SIZE = 9;

public InsuranceTypesController(

InsuranceCompanyContext context,

InsuranceCompanyCache cache,

InsuranceCompanyCookieManager cookieManager,

InsuranceCompanyFilter filter) : base(context, cache, cookieManager, filter) {

}

public async Task<IActionResult> Index(int page, int pageSize = PAGE\_SIZE) {

var insuranceTypeFilter = \_cookieManager.GetCookie<InsuranceTypeFilterModel>(HttpContext.Request.Cookies);

var insuranceTypes = \_filter.Filter(insuranceTypeFilter);

var viewModel = new PageModel<InsuranceType, InsuranceTypeFilterModel>(page, pageSize, insuranceTypes, insuranceTypeFilter);

return View(viewModel);

}

[HttpPost]

public async Task<IActionResult> Index(InsuranceTypeFilterModel insuranceTypeFilter, int page, int pageSize = PAGE\_SIZE) {

\_cookieManager.SetCookie(insuranceTypeFilter, HttpContext.Response.Cookies);

var insuranceTypes = \_filter.Filter(insuranceTypeFilter);

var viewModel = new PageModel<InsuranceType, InsuranceTypeFilterModel>(page, pageSize, insuranceTypes, insuranceTypeFilter);

return View(viewModel);

}

public IActionResult Create() {

return View();

}

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<IActionResult> Create(InsuranceType insuranceType) {

if (ModelState.IsValid) {

\_context.Add(insuranceType);

await \_context.SaveChangesAsync();

UpdateCache();

return RedirectToAction(nameof(Index));

}

return View(insuranceType);

}

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

if (id == null) {

return NotFound();

}

var insuranceTypes = \_cache.GetEntity<InsuranceType>().FirstOrDefault(m => m.Id == id);

if (insuranceTypes == null) {

return NotFound();

}

insuranceTypes.Policies = \_cache.GetEntity<Policy>().Where(e => e.InsuranceTypeId == id).ToList();

return View(insuranceTypes);

}

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

if (id == null) {

return NotFound();

}

var insuranceType = \_cache.GetEntity<InsuranceType>().FirstOrDefault(e => e.Id == id);

if (insuranceType == null) {

return NotFound();

}

return View(insuranceType);

}

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<IActionResult> Edit(int id, InsuranceType insuranceType) {

if (id != insuranceType.Id) {

return NotFound();

}

if (ModelState.IsValid) {

try {

\_context.Update(insuranceType);

await \_context.SaveChangesAsync();

UpdateCache();

}

catch (DbUpdateConcurrencyException) {

return NotFound();

}

return RedirectToAction(nameof(Index));

}

return View(insuranceType);

}

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

if (id == null) {

return NotFound();

}

var insuranceType = \_cache.GetEntity<InsuranceType>().FirstOrDefault(e => e.Id == id);

if (insuranceType == null) {

return NotFound();

}

return View(insuranceType);

}

[HttpPost, ActionName("Delete")]

[ValidateAntiForgeryToken]

public async Task<IActionResult> DeleteConfirmed(int id) {

var insuranceType = \_cache.GetEntity<InsuranceType>().FirstOrDefault(e => e.Id == id);

if (insuranceType != null) {

\_context.InsuranceTypes.Remove(insuranceType);

}

await \_context.SaveChangesAsync();

UpdateCache();

return RedirectToAction(nameof(Index));

}

public void UpdateCache() {

\_cache.SetEntity<InsuranceType>();

\_cache.SetEntity<Policy>();

\_cache.SetEntity<PolicyClient>();

}

}

}

Листинг класса *PoliciesController*.*cs*:

using InsuranceCompany.Services;

using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.Mvc;

using Microsoft.AspNetCore.Mvc.Rendering;

using Microsoft.EntityFrameworkCore;

using Models.Models;

using Models.ViewModels.FilterViewModels;

using Models.ViewModels.PageViewModels;

using Repository;

using Service;

using System.Data;

namespace InsuranceCompany.Controllers {

[Authorize(Roles = "Страховой агент")]

public class PoliciesController : BaseController, IUpdateCache {

private const int PAGE\_SIZE = 9;

public PoliciesController(

InsuranceCompanyContext context,

InsuranceCompanyCache cache,

InsuranceCompanyCookieManager cookieManager,

InsuranceCompanyFilter filter) : base(context, cache, cookieManager, filter) {

}

public async Task<IActionResult> Index(int page, int pageSize = PAGE\_SIZE) {

var policyFilter = \_cookieManager.GetCookie<PolicyFilterModel>(HttpContext.Request.Cookies);

policyFilter.InsuranceAgentList = GetInsuranceAgentsList();

policyFilter.InsuranceTypeList = GetInsuranceTypesList();

var contracts = \_filter.Filter(policyFilter);

var totalInsurancePayment = contracts.Sum(e => e.PolicyPayment);

var viewModel = new PageModelWithAggregateValue<Policy, PolicyFilterModel, decimal>(page, pageSize, contracts, policyFilter, totalInsurancePayment);

return View(viewModel);

}

[HttpPost]

public async Task<IActionResult> Index(PolicyFilterModel policyFilter, int page, int pageSize = PAGE\_SIZE) {

\_cookieManager.SetCookie(policyFilter, HttpContext.Response.Cookies);

policyFilter.InsuranceAgentList = GetInsuranceAgentsList();

policyFilter.InsuranceTypeList = GetInsuranceTypesList();

var contracts = \_filter.Filter(policyFilter);

var totalInsurancePayment = contracts.Sum(e => e.PolicyPayment);

var viewModel = new PageModelWithAggregateValue<Policy, PolicyFilterModel, decimal>(page, pageSize, contracts, policyFilter, totalInsurancePayment);

return View(viewModel);

}

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

if (id == null) {

return NotFound();

}

var policy = \_cache.GetEntity<Policy>().FirstOrDefault(m => m.Id == id);

if (policy == null) {

return NotFound();

}

var clients = \_cache.GetEntity<PolicyClient>().Where(e => e.PolicyId == id).Select(e => e.Client);

var viewModel = (policy, clients);

return View(viewModel);

}

public IActionResult Create() {

ViewData["InsuranceAgentId"] = GetInsuranceAgentsList();

ViewData["InsuranceTypeId"] = GetInsuranceTypesList();

return View();

}

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<IActionResult> Create(Policy policy) {

if (ModelState.IsValid) {

\_context.Add(policy);

await \_context.SaveChangesAsync();

UpdateCache();

return RedirectToAction(nameof(Index));

}

ViewData["InsuranceAgentId"] = GetInsuranceAgentsList();

ViewData["InsuranceTypeId"] = GetInsuranceTypesList();

return View(policy);

}

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

if (id == null) {

return NotFound();

}

var policy = \_cache.GetEntity<Policy>().FirstOrDefault(m => m.Id == id);

if (policy == null) {

return NotFound();

}

ViewData["InsuranceAgentId"] = GetInsuranceAgentsList();

ViewData["InsuranceTypeId"] = GetInsuranceTypesList();

return View(policy);

}

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<IActionResult> Edit(int id, Policy policy) {

if (id != policy.Id) {

return NotFound();

}

if (ModelState.IsValid) {

try {

\_context.Update(policy);

await \_context.SaveChangesAsync();

UpdateCache();

}

catch (DbUpdateConcurrencyException) {

return NotFound();

}

return RedirectToAction(nameof(Index));

}

ViewData["InsuranceAgentId"] = GetInsuranceAgentsList();

ViewData["InsuranceTypeId"] = GetInsuranceTypesList();

return View(policy);

}

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

if (id == null) {

return NotFound();

}

var policy = \_cache.GetEntity<Policy>().FirstOrDefault(m => m.Id == id);

if (policy == null) {

return NotFound();

}

return View(policy);

}

[HttpPost, ActionName("Delete")]

[ValidateAntiForgeryToken]

public async Task<IActionResult> DeleteConfirmed(int id) {

var policy = \_cache.GetEntity<Policy>().FirstOrDefault(m => m.Id == id);

if (policy != null) {

\_context.Policies.Remove(policy);

}

await \_context.SaveChangesAsync();

UpdateCache();

return RedirectToAction(nameof(Index));

}

private IEnumerable<SelectListItem> GetInsuranceAgentsList() {

return \_cache.GetEntity<InsuranceAgent>().Select(e => new SelectListItem() {

Text = $"{e.Surname} {e.Name} {e.MiddleName}",

Value = e.Id.ToString()

});

}

private IEnumerable<SelectListItem> GetInsuranceTypesList() {

return \_cache.GetEntity<InsuranceType>().Select(e => new SelectListItem() {

Text = e.Name,

Value = e.Id.ToString()

});

}

public void UpdateCache() {

\_cache.SetEntity<Policy>();

\_cache.SetEntity<PolicyClient>();

}

}

}

Листинг класса *PolicyClientsController*.*cs*:

using InsuranceCompany.Services;

using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.Identity;

using Microsoft.AspNetCore.Mvc;

using Microsoft.AspNetCore.Mvc.Rendering;

using Models.Models;

using Models.ViewModels.FilterViewModels;

using Models.ViewModels.PageViewModels;

using Repository;

using Service;

namespace InsuranceCompany.Controllers {

[Authorize(Roles = "Клиент")]

public class PolicyClientsController : BaseController, IUpdateCache {

private readonly UserManager<ApplicationUser> \_userManager;

private const int PAGE\_SIZE = 9;

public PolicyClientsController(

InsuranceCompanyContext context,

InsuranceCompanyCache cache,

InsuranceCompanyCookieManager cookieManager,

InsuranceCompanyFilter filter,

UserManager<ApplicationUser> userManager) : base(context, cache, cookieManager, filter) {

\_userManager = userManager;

}

public async Task<IActionResult> Index(int page, int pageSize = PAGE\_SIZE) {

var policyFilter = \_cookieManager.GetCookie<PolicyFilterModel>(HttpContext.Request.Cookies, "client");

policyFilter.InsuranceAgentList = GetInsuranceAgentsList();

policyFilter.InsuranceTypeList = GetInsuranceTypesList();

var client = GetClient();

var policies = \_filter.Filter(policyFilter, client.Id);

var viewModel = new PageModel<Policy, PolicyFilterModel>(page, pageSize, policies, policyFilter);

return View(viewModel);

}

[HttpPost]

public async Task<IActionResult> Index(PolicyFilterModel policyFilter, int page, int pageSize = PAGE\_SIZE) {

\_cookieManager.SetCookie(policyFilter, HttpContext.Response.Cookies, "client");

policyFilter.InsuranceAgentList = GetInsuranceAgentsList();

policyFilter.InsuranceTypeList = GetInsuranceTypesList();

var client = GetClient();

var policies = \_filter.Filter(policyFilter, client.Id);

var viewModel = new PageModel<Policy, PolicyFilterModel>(page, pageSize, policies, policyFilter);

return View(viewModel);

}

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

if (id == null) {

return NotFound();

}

var policy = \_cache.GetEntity<Policy>().FirstOrDefault(m => m.Id == id);

if (policy == null) {

return NotFound();

}

return View(policy);

}

public async Task<IActionResult> Create(int page, int pageSize = PAGE\_SIZE) {

var policyFilter = \_cookieManager.GetCookie<PolicyFilterModel>(HttpContext.Request.Cookies, "client");

policyFilter.InsuranceAgentList = GetInsuranceAgentsList();

policyFilter.InsuranceTypeList = GetInsuranceTypesList();

var policies = \_filter.Filter(policyFilter);

var viewModel = new PageModel<Policy, PolicyFilterModel>(page, pageSize, policies, policyFilter);

return View(viewModel);

}

[HttpPost]

public async Task<IActionResult> Create(PolicyFilterModel policyFilter, int page, int pageSize = PAGE\_SIZE) {

\_cookieManager.SetCookie(policyFilter, HttpContext.Response.Cookies, "client");

policyFilter.InsuranceAgentList = GetInsuranceAgentsList();

policyFilter.InsuranceTypeList = GetInsuranceTypesList();

var policies = \_filter.Filter(policyFilter);

var viewModel = new PageModel<Policy, PolicyFilterModel>(page, pageSize, policies, policyFilter);

return View(viewModel);

}

public async Task<IActionResult> Add(int? id) {

if (id == null) {

return NotFound();

}

var policy = \_cache.GetEntity<Policy>().FirstOrDefault(e => e.Id == id);

if (policy == null) {

return NotFound();

}

return View(policy);

}

[HttpPost]

public async Task<IActionResult> Add(int id) {

var policy = \_cache.GetEntity<Policy>().FirstOrDefault(e => e.Id == id);

var client = GetClient();

if (policy == null) {

return NotFound();

}

\_context.Add(new PolicyClient() { ClientId = client.Id, PolicyId = policy.Id });

await \_context.SaveChangesAsync();

UpdateCache();

return RedirectToAction(nameof(Index));

}

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

if (id == null) {

return NotFound();

}

var policy = \_cache.GetEntity<Policy>().FirstOrDefault(m => m.Id == id);

if (policy == null) {

return NotFound();

}

return View(policy);

}

[HttpPost]

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

var clientId = GetClient().Id;

var policyClient = \_cache.GetEntity<PolicyClient>().FirstOrDefault(e => e.PolicyId == id && e.ClientId == clientId);

if (policyClient != null) {

\_context.PolicyClients.Remove(policyClient);

}

await \_context.SaveChangesAsync();

UpdateCache();

return RedirectToAction(nameof(Index));

}

private IEnumerable<SelectListItem> GetInsuranceAgentsList() {

return \_cache.GetEntity<InsuranceAgent>().Select(e => new SelectListItem() {

Text = $"{e.Surname} {e.Name} {e.MiddleName}",

Value = e.Id.ToString()

});

}

private IEnumerable<SelectListItem> GetInsuranceTypesList() {

return \_cache.GetEntity<InsuranceType>().Select(e => new SelectListItem() {

Text = e.Name,

Value = e.Id.ToString()

});

}

private Client GetClient() {

var applicationUser = \_userManager.GetUserAsync(HttpContext.User).Result;

return \_cache.GetEntity<Client>()

.FirstOrDefault(e =>

applicationUser.Name == e.Name &&

applicationUser.Surname == e.Surname &&

applicationUser.MiddleName == e.MiddleName);

}

public void UpdateCache() {

\_cache.SetEntity<PolicyClient>();

}

}

}

Листинг класса *SupportingDocumentsController*.*cs*:

using InsuranceCompany.Services;

using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.Mvc;

using Microsoft.EntityFrameworkCore;

using Models.Models;

using Models.ViewModels.FilterViewModels;

using Models.ViewModels.PageViewModels;

using Repository;

using Service;

using System.Data;

namespace InsuranceCompany.Controllers {

[Authorize(Roles = "Страховой агент")]

public class SupportingDocumentsController : BaseController, IUpdateCache {

private const int PAGE\_SIZE = 9;

public SupportingDocumentsController(

InsuranceCompanyContext context,

InsuranceCompanyCache cache,

InsuranceCompanyCookieManager cookieManager,

InsuranceCompanyFilter filter) : base(context, cache, cookieManager, filter) {

}

public async Task<IActionResult> Index(int page, int pageSize = PAGE\_SIZE) {

var supportingDocumentFilter = \_cookieManager.GetCookie<SupportingDocumentFilterModel>(HttpContext.Request.Cookies);

var supportingDocuments = \_filter.Filter(supportingDocumentFilter);

var viewModel = new PageModel<SupportingDocument, SupportingDocumentFilterModel>(page, pageSize, supportingDocuments, supportingDocumentFilter);

return View(viewModel);

}

[HttpPost]

public async Task<IActionResult> Index(SupportingDocumentFilterModel supportingDocumentFilter, int page, int pageSize = PAGE\_SIZE) {

\_cookieManager.SetCookie(supportingDocumentFilter, HttpContext.Response.Cookies);

var supportingDocuments = \_filter.Filter(supportingDocumentFilter);

var viewModel = new PageModel<SupportingDocument, SupportingDocumentFilterModel>(page, pageSize, supportingDocuments, supportingDocumentFilter);

return View(viewModel);

}

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

if (id == null) {

return NotFound();

}

var supportingDocument = \_cache.GetEntity<SupportingDocument>().FirstOrDefault(e => e.Id == id);

if (supportingDocument == null) {

return NotFound();

}

supportingDocument.InsuranceCases = \_cache.GetEntity<InsuranceCase>().Where(e => e.SupportingDocumentId == id).ToList();

return View(supportingDocument);

}

public IActionResult Create() {

return View();

}

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<IActionResult> Create(SupportingDocument supportingDocument) {

if (ModelState.IsValid) {

\_context.Add(supportingDocument);

await \_context.SaveChangesAsync();

UpdateCache();

return RedirectToAction(nameof(Index));

}

return View(supportingDocument);

}

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

if (id == null) {

return NotFound();

}

var supportingDocument = \_cache.GetEntity<SupportingDocument>().FirstOrDefault(e => e.Id == id);

if (supportingDocument == null) {

return NotFound();

}

return View(supportingDocument);

}

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<IActionResult> Edit(int id, SupportingDocument supportingDocument) {

if (id != supportingDocument.Id) {

return NotFound();

}

if (ModelState.IsValid) {

try {

\_context.Update(supportingDocument);

UpdateCache();

await \_context.SaveChangesAsync();

}

catch (DbUpdateConcurrencyException) {

return NotFound();

}

return RedirectToAction(nameof(Index));

}

return View(supportingDocument);

}

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

if (id == null) {

return NotFound();

}

var supportingDocument = \_cache.GetEntity<SupportingDocument>().FirstOrDefault(e => e.Id == id);

if (supportingDocument == null) {

return NotFound();

}

supportingDocument.InsuranceCases = \_cache.GetEntity<InsuranceCase>().Where(e => e.SupportingDocumentId == id).ToList();

return View(supportingDocument);

}

[HttpPost, ActionName("Delete")]

[ValidateAntiForgeryToken]

public async Task<IActionResult> DeleteConfirmed(int id) {

var supportingDocument = \_cache.GetEntity<SupportingDocument>().FirstOrDefault(e => e.Id == id);

if (supportingDocument != null) {

\_context.SupportingDocuments.Remove(supportingDocument);

}

await \_context.SaveChangesAsync();

UpdateCache();

return RedirectToAction(nameof(Index));

}

public void UpdateCache() {

\_cache.SetEntity<SupportingDocument>();

\_cache.SetEntity<InsuranceCase>();

}

}

}

# ПРИЛОЖЕНИЕ Б

(обязательное)

**Чертёж структуры *web*-приложения**