ТЗ: В результате задания должно получиться

**1. приложение на ASP.Net MVC, которое умеет:**

1.1. показывать сохраненные в базе данных записи по посещениям врачей/пользовательским записям/календарям/прививкам/списку мед препаратов и далее по списку в зависимости от имеющихся сущностей.

1.1.1. Календарь запланированных прошедших посещений специалистов

1.1.2. Что-то вроде личного кабинета пользователя в рамках, которого у него есть записи, оставленные после посещения специалистов с назначенными препаратами/диагнозом и иной информацией на ваше усмотрение

1.1.3. Такой же календарь со сделанными прививками и запланированные новые (опциональный функционал)

**Что хотелось бы видеть:**

1.2. Показывает это все сугубо авторизованному пользователю и сугубо его данные/либо те, к которым у него есть доступ.

1.3. содержит роли (админ и пользователь)

1.4. для админа должна быть доступна отдельная страница, которая позволит решать базовые функциональные задачи вроде получения препаратов, сброса пользовательских паролей, возможно, рассылки каких-то писем всем пользователям и т.п..

1.5. В идеальном случае хотелось бы видеть алгоритм парсинга сторонних источников:

1.5.1. К примеру работа с tabletka.by для получения базовой информации о мед препаратах. Базово имплементация подразумевает использование HTML AGILITY PACK.

1.6. Использовать обработку ошибок и их логгирование посредством Serilog.

* (Обработка ошибок: Обработка ошибок является важной частью разработки программного обеспечения. Ошибки могут возникать в различных частях приложения, их обработка позволяет предотвратить сбои и падения приложения и обеспечить гибкую обработку ошибок. Это может включать в себя проверку условий, обработку исключений и возврат информации об ошибках пользователям или администраторам.
* Логгирование: Логгирование - это процесс записи информации о работе приложения и его состоянии в журналы или файлы. Логи могут быть особенно полезными при обработке ошибок, поскольку они позволяют отслеживать последовательность событий и идентифицировать проблемные места в коде. Логи также могут быть полезными для отладки и мониторинга приложения.
* Serilog: Serilog - это библиотека для логгирования в приложениях на платформе .NET. Она предоставляет гибкую и выразительную конфигурацию логгирования с поддержкой различных назначений вывода (например, консоль, файл, база данных и другие). Serilog также обладает богатыми возможностями форматирования сообщений, фильтрации, структурирования и расширения.

Таким образом, фраза "Использовать обработку ошибок и их логгирование посредством Serilog" означает, что в приложении следует использовать Serilog для обработки ошибок и записи информации о них в журналы или файлы. Serilog предоставляет удобный и гибкий способ настройки и управления логгированием, что позволяет эффективно отслеживать и реагировать на ошибки, а также облегчает отладку и мониторинг приложения.)

1.7. Работа с базой должна быть реализована с использованием паттернов Repository и UoW. В идеальном случае с использованием Generic Repository и UoW. Либо с использованием паттерна CQS/CQRS.

* (Паттерны Repository и UoW: Паттерн Repository (Репозиторий) используется для абстрагирования доступа к данным и предоставления единообразного интерфейса для работы с ними. Репозиторий предоставляет методы для создания, чтения, обновления и удаления (CRUD) данных в базе данных. Паттерн Unit of Work (Единица работы) отвечает за управление транзакцией и сохранением изменений в базе данных. Он объединяет несколько операций в одну транзакцию и обеспечивает целостность данных.
* Generic Repository и UoW: В идеальном случае, реализация паттернов Repository и UoW может быть улучшена с использованием Generic Repository и UoW. Generic Repository позволяет создать универсальную реализацию репозитория, которая может работать с различными типами сущностей без необходимости создания отдельного репозитория для каждой сущности. Generic UoW обеспечивает общий механизм управления транзакциями и сохранения изменений для всех репозиториев.
* Паттерн CQS/CQRS: Паттерн CQS (Command-Query Separation) разделяет операции, которые изменяют состояние данных (команды), от операций, которые только получают данные (запросы). Паттерн CQRS (Command-Query Responsibility Separation) расширяет паттерн CQS, разделяя модели данных для команд и запросов. В контексте работы с базой данных, это означает, что команды для изменения данных и запросы для получения данных должны быть разделены на отдельные модели или классы. Это позволяет оптимизировать доступ к данным и упростить логику приложения.

Таким образом, фраза "работа с базой должна быть реализована с использованием паттернов Repository и UoW. В идеальном случае с использованием Generic Repository и UoW. Либо с использованием паттерна CQS/CQRS" означает, что при работе с базой данных рекомендуется использовать паттерны Repository и UoW для абстрагирования доступа к данным и управления транзакциями. В идеале, можно использовать Generic Repository и UoW для более универсальной реализации. Альтернативно, можно применить паттерн CQS/CQRS для разделения операций изменения данных (команд) от операций получения данных (запросов), что может упростить и оптимизировать работу с базой данных и логику приложения.)

1.8. Приложение должно использовать встроенный в ASP.Net Core IoC контейнер для внедрения зависимостей во всех местах, где это необходимо.

* (Приложение: Здесь речь идет о разрабатываемом приложении, которое использует фреймворк ASP.Net Core для создания веб-приложений. ASP.Net Core предоставляет множество функций и инструментов для разработки современных и надежных веб-приложений.
* Встроенный в ASP.Net Core IoC контейнер: Встроенный в ASP.Net Core IoC (Inversion of Control) контейнер предоставляет механизм для управления зависимостями в приложении. Он позволяет определить, какие объекты должны быть созданы и как они должны быть связаны друг с другом. IoC контейнер в ASP.Net Core реализует паттерн внедрения зависимостей (Dependency Injection), который помогает управлять сложностью приложения, повышает его гибкость и облегчает тестирование.
* Внедрение зависимостей: Внедрение зависимостей - это процесс предоставления объектам необходимых зависимостей для их работы. Вместо того, чтобы объекты создавали свои зависимости напрямую, они получают их из внешнего источника. В ASP.Net Core это осуществляется с использованием встроенного IoC контейнера.
* Во всех местах, где это необходимо: Внедрение зависимостей должно быть применено во всех частях приложения, где требуется использование внешних компонентов или сервисов. Это может быть в контроллерах, сервисах, репозиториях и других компонентах приложения. Цель состоит в том, чтобы разделить ответственность между компонентами и обеспечить легкую замену зависимостей при необходимости.

Таким образом, фраза "Приложение должно использовать встроенный в ASP.Net Core IoC контейнер для внедрения зависимостей во всех местах, где это необходимо" означает, что в приложении следует использовать встроенный IoC контейнер для управления зависимостями между компонентами приложения. Это позволяет легко добавлять, изменять или заменять зависимости в различных частях приложения, обеспечивая гибкость, расширяемость и легкость тестирования.)

\* 1.9.(опционально) наличие функционала который использует запросы на сервер посредством Ajax (JavaScript или jQuery).

**2. приложение на ASP.Net Web Api, которое**

2.1. Предоставляет собой Rest API для работы с логикой приложения аналогичной MVC и

потенциального построения клиентских приложений

* (Rest API: Это означает, что данное приложение предоставляет интерфейс программирования приложений (API), основанный на архитектурном стиле REST (Representational State Transfer). REST является популярным подходом к созданию веб-сервисов, где ресурсы (например, данные) представлены в виде уникальных URL-адресов и доступны через стандартные HTTP-методы, такие как GET, POST, PUT и DELETE. Rest API позволяет клиентским приложениям взаимодействовать с сервером и выполнять операции над ресурсами.
* Работа с логикой приложения: Это означает, что Rest API обеспечивает доступ и выполнение операций, связанных с логикой вашего приложения. Это может включать создание, чтение, обновление и удаление (CRUD) данных, выполнение бизнес-логики, обработку запросов и многое другое, в зависимости от требований вашего приложения.
* Аналогичной MVC: Здесь упоминается сходство с популярным шаблоном проектирования MVC (Model-View-Controller). MVC разделяет приложение на три основных компонента: модель (Model), представление (View) и контроллер (Controller). Модель представляет данные и бизнес-логику, представление отображает данные пользователю, а контроллер обрабатывает входящие запросы и управляет взаимодействием между моделью и представлением. Это означает, что Rest API может использовать подобный подход, где логика приложения разделена на модели, которые представляют данные, контроллеры, которые обрабатывают запросы, и представления, которые отображают данные.
* Потенциального построения клиентских приложений: Это означает, что Rest API предоставляет интерфейс для разработки клиентских приложений. Клиентские приложения могут быть веб-приложениями, мобильными приложениями или другими системами, которые могут использовать Rest API для взаимодействия с сервером и получения данных или выполнения операций. Rest API предоставляет стандартизированный способ общения между клиентскими и серверными приложениями, что позволяет разрабатывать разнообразные клиентские приложения, опираясь на логику приложения, предоставляемую через Rest API.)

2.2. Использует Scheduled Job (Hangfire) для выполнения фоновых задач.

* (Scheduled Job: Это означает, что в приложении используется планировщик задач, который позволяет выполнять задачи по расписанию. Задачи могут быть запланированы для выполнения в определенное время или с определенной периодичностью. Это позволяет автоматизировать выполнение операций, которые должны выполняться регулярно или в определенные моменты времени.
* Hangfire: Hangfire - это библиотека для выполнения фоновых задач в приложении на платформе .NET. Она предоставляет простой и надежный способ запуска задач в фоновом режиме. Hangfire позволяет создавать и планировать фоновые задачи, которые выполняются вне основного потока запросов вашего приложения. Она обеспечивает надежность выполнения задач даже в случае перезапуска приложения или сбоев.

Таким образом, фраза "Использует Scheduled Job (Hangfire) для выполнения фоновых задач" означает, что в приложении используется библиотека Hangfire для планирования и выполнения фоновых задач. Благодаря этому, задачи могут быть запланированы для выполнения в заданное время или с определенной периодичностью, и будут выполняться в фоновом режиме вне основного потока запросов вашего приложения. Это может быть полезно для выполнения длительных или ресурсоемких операций, которые не должны замедлять ответы на запросы пользователей и могут быть выполнены асинхронно.)

2.3. Использует Swagger UI в рамках WebAPI проекта

2.4. Имеет аутентификацию/авторизацию с использованием JWT

2.5. Логика работы с базой в CQS/CQRS + MediatR

\* 2.6. (Дополнительно) Для пытливых умов посмотреть и реализовать паттерн Specification для запросов в рамках работы с CQS и MediatR

Кодовая база приложений должна быть максимально общей.

**3. Для желающих (опционально)**

Начать разбираться с React\Angular\VueJS\Blazor\etc на ваш выбор.

Версии берите САМЫЕ ПОСЛЕДНИЕ. Как развернуть стандартный проект, инфраструктуру

для него и т.д. NodeJS - последний LTS.

ИНСТРУМЕНТЫ И РЕАЛИЗАЦИЯ ПРИ УСЛОВИИ ОТСУТСТВИИ КОНКРЕТИКИ В ТЗ

ОСТАЮТСЯ НА ВАШЕ УСМОТРЕНИЕ. РЕШЕНИЕ МОЖЕТ НЕСКОЛЬКО ОТХОДИТЬ ОТ ИЗНАЧАЛЬНОГО УСЛОВИЯ ПРИ СОХРАНЕНИИ БАЗОВОЙ КОНЦЕПЦИИ ТЕМАТИКИ.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\* Visual Studio 2022 Developer PowerShell v17.9.5

\*\* Copyright (c) 2022 Microsoft Corporation

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

PS E:\C\_Sharp\_ASP\_NET\_Core\FamilyCalendar> dotnet tool install --global dotnet-ef

Инструмент "dotnet-ef" был переустановлен. Установлена стабильная версия (версия "8.0.4").

PS E:\C\_Sharp\_ASP\_NET\_Core\FamilyCalendar> cd..

PS E:\C\_Sharp\_ASP\_NET\_Core> dotnet ef dbcontext list -p FamilyCalendar -s FamilyCalendar

Build started...

Build succeeded.

Calendar.DataAccess.CalendarDbContext

PS E:\C\_Sharp\_ASP\_NET\_Core> dotnet ef migrations add -p FamilyCalendar -s FamilyCalendar

Build started...

Build succeeded.

Missing required argument '<NAME>'.

PS E:\C\_Sharp\_ASP\_NET\_Core> dotnet ef migrations add -p FamilyCalendar -s FamilyCalendar Firstmigration

Build started...

Build succeeded.

Unable to create a 'DbContext' of type ''. The exception 'The entity type 'CalendarEntity' requires a primary key to be defined. If you intended to use a keyless entity type, call 'HasNoKey' in 'OnModelCreating'. For more information on keyless entity types, see https://go.microsoft.com/fwlink/?linkid=2141943.' was thrown while attempting to create an instance. For the different patterns supported at design time, see https://go.microsoft.com/fwlink/?linkid=851728

PS E:\C\_Sharp\_ASP\_NET\_Core>