**Инструкция по запуску проекта**

**1. Предварительные требования**

Убедитесь, что на вашем компьютере установлены:

* **.NET SDK** версии 6.0 или выше
* **Microsoft SQL Server** (если используете локальную базу данных SQL Server)
* **Postman** или любой другой инструмент для тестирования API
* **Visual Studio** или **Visual Studio Code** для редактирования и запуска кода
* Пакет **Entity Framework Core** и соответствующие провайдеры базы данных (например, Microsoft.EntityFrameworkCore.SqlServer)

**2. Структура проекта**

В проекте организованы следующие папки и классы:

* **Entity**
  + Member: Класс для представления участников события.
  + MyEvent: Класс для представления событий.
  + EventImage: Класс для работы с изображениями событий.
* **Data**
  + ModsenPracticeContext: Класс контекста базы данных, определяющий связи между сущностями и работу с Entity Framework.
* **Controllers**
  + MyEventsController: Контроллер для управления событиями (CRUD операции, работа с изображениями).
  + MyMembersController: Контроллер для управления участниками события.
* **Helpers**
  + MyHelpers: Статический класс с дополнительной логикой, включая методы для работы с изображениями и валидацией данных.
* **wwwroot/Images**: Папка, в которой хранятся изображения событий.

**3. Настройка базы данных**

1. **Конфигурация строки подключения**:
   * Откройте appsettings.json и настройте строку подключения к базе данных SQL Server (или другой СУБД, которую вы используете).
   * Пример настройки строки подключения:

json

Копировать код

"ConnectionStrings": {

"DefaultConnection": "Server=YOUR\_SERVER\_NAME;Database=YOUR\_DATABASE\_NAME;Trusted\_Connection=True;"

}

1. **Применение миграций**:
   * Откройте консоль диспетчера пакетов в Visual Studio и выполните команду для создания базы данных и применения миграций:

bash

Копировать код

dotnet ef database update

1. Если миграции еще не были созданы, используйте команду dotnet ef migrations add InitialCreate для их создания.

**4. Запуск приложения**

1. Откройте проект в **Visual Studio** или **Visual Studio Code**.
2. Убедитесь, что выбран профиль запуска IIS Express или Kestrel.
3. Нажмите **F5** или выберите **Запуск** для старта приложения. API должен запуститься на https://localhost:порт.

**5. Использование API**

Для тестирования API используйте **Postman** или любой другой HTTP-клиент.

**Работа с событиями (MyEventsController)**

1. **Получение списка всех событий**:
   * **GET** запрос к https://localhost:порт/api/MyEvents
2. **Получение определенного события по ID**:
   * **GET** запрос к https://localhost:порт/api/MyEvents/{id}
3. **Получение события по названию**:
   * **GET** запрос с параметром name к https://localhost:порт/api/MyEvents?name=название события
4. **Добавление нового события**:
   * **POST** запрос к https://localhost:порт/api/MyEvents
   * Передавайте данные о событии в теле запроса в формате JSON.
5. **Изменение информации о существующем событии**:
   * **PUT** запрос к https://localhost:порт/api/MyEvents/id
   * Передавайте измененные данные о событии и изображения в теле запроса.
6. **Удаление события**:
   * **DELETE** запрос к https://localhost:порт/api/MyEvents/id
7. **Получение списка событий по критериям (дата, место проведения, категория)**:
   * **GET** запрос с параметрами date, location или category к https://localhost:порт/api/MyEvents?date=дата&location=место&category=категория

**Работа с участниками (MyMembersController)**

1. **Регистрация участия пользователя в событии**:
   * **POST** запрос к https://localhost:порт/api/MyMembers
   * Передайте данные об участнике в теле запроса.
2. **Получение списка участников события**:
   * **GET** запрос к https://localhost:порт/api/MyMembers/event/eventId
3. **Получение определенного участника по ID**:
   * **GET** запрос к https://localhost:порт/api/MyMembers/id
4. **Отмена участия пользователя в событии**:
   * **DELETE** запрос к https://localhost:порт/api/MyMembers/id

**Отправка уведомлений**

Для отправки уведомлений участникам после изменений в событии используйте встроенную функцию SendEmail в контроллере MyEventsController.

**6. Важные моменты**

* **Работа с изображениями**:
  + При создании и обновлении событий можно загружать изображения. Изображения будут сохраняться в папке wwwroot/Images.
  + Убедитесь, что у приложения есть доступ к файлам и папкам, чтобы корректно сохранять и удалять изображения.
* **Отправка Email**:
  + При изменении события система отправляет email-уведомление участникам о произведенных изменениях.
  + Для отправки сообщений используется учетная запись Gmail. Проверьте настройки безопасности учетной записи для разрешения отправки сообщений через сторонние приложения.

**Часть 2. Работа с дополнительными требованиями проекта**

1. **Ветвление и функционал Git**: Основной функционал проекта находится в ветке main. Для реализации требований был создана отдельная ветка ForFeatures, где проводилась основная работа по внедрению дополнительных возможностей. Ветка ForFeatures содержит все изменения, связанные с расширением функционала и применением необходимых технологий.
2. **Реализация авторизации с использованием JWT и Refresh токенов**:
   * В папке Entity добавлены классы RoleName и User, необходимые для управления ролями и пользователями.
   * Для передачи данных и авторизации была создана папка DTO (Data Transfer Object), содержащая классы TokenRequestDto, UserLoginDto, и UserRegistrationDto. Эти классы используются для передачи данных между слоями и работы с токенами.
   * В файле Program.cs настроены параметры для использования токенов, а в appsettings.json указано время их жизни и назначение.
   * Токены добавлены в базу данных, и функционал авторизации протестирован для корректности.
3. **Реализация паттерна Repository и Unit of Work**:
   * В проекте создана папка Patterns, содержащая директории Repository и UnitOfWork для структурирования кода.
   * В Repository находятся интерфейсы IRepository, IRoleRepository, IUserRepository, а также их реализации RoleRepository и UserRepository.
   * Для Unit of Work добавлены интерфейс IUnitOfWork и реализация UnitOfWork, обеспечивающие централизованное управление репозиториями и транзакциями.
4. **Middleware для глобальной обработки исключений**:
   * В папке Helpers создан класс ExceptionHandlingMiddleware, который выполняет обработку исключений.
   * Middleware зарегистрирован в Program.cs, и обработка исключений проверена для предотвращения некорректного поведения приложения.
5. **Реализация пагинации**:
   * В классе UserRepository добавлен метод, возвращающий всех пользователей с логикой пагинации, что позволяет гибко получать данные в заданных пределах.
6. **Проект для Unit-тестирования**:
   * Создан отдельный проект PracticeTest для написания тестов. В проект добавлена ссылка на основной проект, что позволяет использовать его классы и методы в тестах.
   * Написаны юнит-тесты для основного функционала, проверяющие корректность методов и обработки данных.
7. **Используемые технологии**:
   * Необходимые технологии, такие как .NET 5.0+, Entity Framework Core, JWT-токены, и xUnit для тестирования, интегрированы и протестированы в проекте.