Задание стажировка C# #1

**Цель задания:**

Сформировать навык создания простого сервиса.  
Проектирование REST API, работа с библиотеками Mediatr и Fluent validation

**Крайний срок сдачи задания:**

9:00 - 29 июля 2025

**Описание бизнес задачи:**

На протяжении курса мы будем разрабатывать микросервис «Банковские счета», обслуживающий процессы розничного банка.

На данном этапе реализации сервиса мы хотим реализовать следующие пользовательские истории:

* Я, как менеджер банка Анна, открыла клиенту Ивану бесплатный текущий счёт, чтобы он мог хранить средства.
* Я, как менеджер банка Анна, открыла клиенту Ивану срочный вклад «Надёжный‑6» под 3 % годовых, чтобы он смог накопить средства.
* Я, как кассир банка Алексей, пополнил текущий счёт клиента Ивана на 1 000 рублей наличными.
* Я, как клиент банка Иван, перевёл 200 рублей со своего текущего счёта на вклад «Надёжный‑6», чтобы пополнить вклад.

После груминга было решено создать сервис «Счета» (Account Service) с REST‑API, который позволяет:

* создать счёт
* изменить счёт
* удалить счёт
* получить список счетов
* зарегистрировать транзакцию по счёту
* выполнить перевод между счётами
* выдать выписку клиенту
* проверить наличие счёта у клиента

Свойства счёта

* id (GUID)
* ownerId (GUID)
* тип (Checking | Deposit | Credit)
* валюта (ISO 4217)
* баланс (decimal)
* процентная ставка (decimal, опционально — только для Deposit и Credit)
* дата открытия
* дата закрытия (опционально)
* коллекция транзакций

Свойства транзакции

* id (GUID)
* accountId (GUID)
* counterpartyAccountId (GUID, опционально)
* сумма (decimal)
* валюта (ISO 4217)
* тип (Credit | Debit)
* описание
* дата/время

**Техническая задача:**

1. Настроить окружение:
   1. Установить VS 2022 Community Edition
   2. Установить Resharper
2. Спроектировать REST API согласно бизнес заданию, только для тех методов и свойств, используя регламент из Приложения 1
3. Создать сервис WebApi dotnet 9
4. Реализовать спроектированный REST API используя заглушки. Формат данных - JSON.
5. Подключить библиотеку Mediatr <https://github.com/jbogard/MediatR>   
   <https://code-maze.com/cqrs-mediatr-fluentvalidation/>
6. Заменить логику в контроллерах на вызов соответствующих операций Mediatr
7. Подключить библиотеку FluentValidation <https://github.com/FluentValidation/FluentValidation>
8. Включить библиотеку FluentValidation в pipeline Mediatr https://www.youtube.com/watch?v=2JzQuIvxIqk
9. Создать и подключить фильтр для обработки ошибок валидации которые вызваны работой FuentValidation
10. Реализовать валидацию запросов
11. Подключить swagger для документации API <https://github.com/domaindrivendev/Swashbuckle.AspNetCore>   
    все методы API и их параметры должны быть описаны, также указаны возвращаемые значения
12. Создать заглушку сервиса хранения счётов (CRUD модель).
13. Создать заглушку сервиса верификации клиентов (метод проверки существования ownerId).
14. Создать заглушку сервиса валют (метод проверки поддерживаемой валюты).
15. С помощью этих заглушек реализовать функционал сервиса.
16. Опубликовать сервис на github и прислать ссылку куратору. @cyber\_universe

\* В данном контексте заглушка - класс с простейшей реализацией требуемых операций (например, in‑memory лист, внедряемый как Singleton).

**Критерии приемки**:

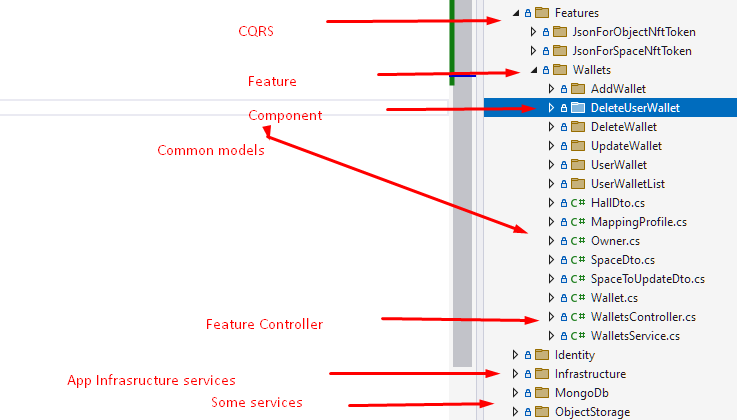
1. Сервис опубликован на github как public
2. Сервис запускается будучи открыт в VS2022
3. При запуске открывается страница swagger
4. Все методы, параметры REST API и возвращаемые значения документированы на странице swagger
5. Можно провести все CRUD операции с событиями используя JSON для обмена данными.
6. REST API должен удовлетворять Регламенту из Приложения 1
7. Все зависимости должны передаваться через интерфейсы
8. При задании некорректных дат события (начало позже окончания, например), или не существующих изображения или пространства, или если пространство null - выдается 400 ошибка с указанием причины.
9. Функционал реализован с использованием Mediatr и FlientValidation
10. Код должен быть оформлен в соответствии с правилами оформления Resharper по умолчанию.
11. Должно быть 0 замечаний при анализе Resharper code issues in solution

**Приложение 1:**  
  
**Регламент REST API**

Требования к REST API микросервисов

1. Формат передачи данных
   1. Данные передаются в формате JSON.
   2. Исключение составляет передача файлов, для которой используются соответствующие форматы (например, multipart/form-data).
2. Использование HTTP-методов
   1. Для получения данных используется метод GET.
   2. Для идемпотентного изменения (замена целиком) ресурса используется метод PUT.
   3. Для частичного изменения ресурса используется метод PATCH.
   4. Для неидемпотентных операций и создания новых ресурсов используется метод POST.
   5. Для удаления ресурсов используется метод DELETE.
   6. Исключение: если параметры GET-запроса превышают допустимый размер, допускается реализация запроса на получение данных через метод POST.
3. Документирование REST API
   1. Каждый метод REST API должен быть документирован в Swagger (OpenAPI).
   2. Документация метода должна содержать модель входящих данных, если метод принимает данные.
   3. Документация метода должна содержать модель исходящих данных, если метод возвращает данные.
   4. Документация метода должна содержать текстовое описание, если действие метода не очевидно из его названия и/или HTTP-метода.
4. Формирование URL  
   На примере сущности house используются следующие правила:
   1. Получение всех сущностей: GET /houses
   2. Получение сущностей с фильтрацией и сортировкой: GET /houses?type=barnhouse&orderBy=name
   3. Получение конкретной сущности по идентификатору: GET /houses/{houseId}
   4. Добавление новой сущности: POST /houses
   5. Изменение сущности целиком: PUT /houses/{houseId}
   6. Удаление сущности: DELETE /houses/{houseId}
   7. Получение отдельного свойства сущности (например, type): GET /houses/{houseId}/type
   8. Изменение отдельного свойства сущности: PUT /houses/{houseId}/type
   9. Получение вложенных коллекций и свойств (например, комнат дома): GET /houses/{houseId}/rooms
   10. При большой иерархии вложенных ресурсов и необходимости упрощения запросов допустимо переносить вложенные коллекции на верхний уровень. Например: GET /rooms?houseId=1
5. Сериализация перечислимых типов
   1. Перечислимые свойства сериализуются в виде строковых значений.

**Приложение 2**  
  
Рекомендуемая структура папок проекта



***Приложение 3***  
  
1. Код должен быть оформлен в соответствии с правилами оформления Resharper по умолчанию.  
2. Должно быть 0 замечаний при анализе Resharper code issues in solution