Задание стажировка C# #2

**Цель задания:**

Углубить навык создания микросервисов: продолжить проектирование REST‑API, добавить контейнеризацию (Docker + docker‑compose) и настроить аутентификацию JWT через Keycloak (или аналог), настроить работу с git.

**Крайний срок сдачи задания:**

9:00 - 05 августа 2025 года

**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.
* Я, как менеджер банка Анна, выдал клиенту Ивану выписку по его счетам.
* Я, как клиент банка Иван, запросил баланс своего текущего счёта.

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

* создать счёт
* изменить счёт
* удалить счёт
* получить список счетов
* зарегистрировать входящую/исходящую транзакцию по счёту
* выполнить перевод между счётами (внутри банка)
* получить выписку по счёту за данный период
* 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 FluentValidation для всех валидаторов так, чтобы проверялись все правила, но по каждому полю возвращалась только первая ошибка.
2. Реализовать обобщенный (genec) тип MbResult, который может содержать или результат выполнения операции или ошибку.
   1. Обеспечить возвращение всех результатов ответа REST API операций в этом формате.
   2. Обеспечить возвращение всех ошибочных результатов в этом формате.
3. Ко всем полям свойств объектов доступных через API должны в Swagger показываться описания.
4. Добавить Dockerfile в решение. Сервис должен публиковать API на 80 порт докера.
5. Добавить поддержку CORS - allow all
6. Добавить аутентификацию на все методы API,через JWT токен.
7. Добавить docker-compose с сервером аутентификации и его начальной конфигурацией. Допустимы любые удобные для вас варианты - keycloak, сервер на базе identity-server4, самописный сервер, другие варианты

Мы решили упростить задачу - на данный момент достаточно просто валидности токена, т.е. роль пользователя не проверяем.

**Критерии приемки**:

1. Результат прислан в виде PR из ветки для работы над вторым заданием в основную ветку репозитория.
2. Если репозиторий сервиса клонировать открыть в VS2022, запустить - сервис запуститься, откроется страница со swagger
3. Если репозиторий сервиса клонировать, собрать докер, запустить докер- то на 80-м порту докера будет опубликовано API сервиса, а по пути :swagger будет опубликован swagger сервиса.
4. После запуска resharper -> inspect -> Code Issues In Solution с настройками по умолчанию мы получаем 0 ошибок. Все места, которые как Вы считаете не стоит исправлять, должны быть отключены одностроковыми комментариями с указанием аргументации почему это сделано или с помощью JetBrains.Annotations (например, [UsedImplicitly] если смысл аттрибутов соответствует причине их использования)
5. Все методы, параметры REST API, модели используемые в REST API с из параметрами и возвращаемые значения документированы на странице swagger и содержат описания.  
   Должны быть указаны типы в том числе возвращаемых значений.  
   Должны быть указаны возвращаемые статусы (400, 200, 500, 401 и т.д.)

Также обязательно.  
Для метода POST событий   
- текстовое описание самого метода  
Текстовые описания должны подтягиваться из XML комментариев

1. Можно провести все CRUD операции с событиями используя json для обмена данными.
2. REST API должен удовлетворять Регламенту из Приложения 1 из задания 1
3. Все функциональные зависимости должны передаваться через интерфейсы
4. Функционал реализован с использованием Mediatr и FlientValidation
5. Структура папок оформлена в соответствии с Приложением 3 из задания 1
6. В случае возникновения ошибок валидации возвращается информация в поле MbError тим MbResult
7. Есть возможность (работающий launchsettings) запуска и отладки приложения через запуск Docker
8. Все изменения в проекте залиты через Github flow
9. Добавлена поддержка CORS - allow all, т.е. нам нужно убрать проверку CORS
10. Все свойства моделей и акшены в swagger содержат описания
11. Метод выдают 401 при попытке вызова без аутентификации и проходит аутентификацию через JWT access token полученный с сервера аутентификации
12. При запуске docker-compose вместе с сервисом разворачивается сервер аутентификации, уже настроенный для работы с запущенным в рамках этого docker-compose нашим сервисом

**Приложения:**  
  
1. Модель ветвления:

* git flow - <https://habr.com/ru/articles/106912/>
* github flow - <https://habr.com/ru/articles/346066/> - в задании используем его.

2. Vertical slice architecture - <https://habr.com/ru/articles/916704/>

3. Видео по проверке своего проекта в решарпере <https://t.me/backend_internship_modulebank_1/10>

4.