Задание Модульбанк стажировка C# #3

**Цель задания:**

Укрепить навыки разработки микросервисов: перенести хранение данных на PostgreSQL, покрыть бизнес‑логику модульными и интеграционными тестами, освоить оптимизацию производительности и гарантировать потокобезопасное исполнение операций, работа с HangFire.

**Крайний срок сдачи задания:**

9:00 – 12 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.
* Я, как менеджер банка Анна, закрыла срочный вклад клиента Ивана «Надёжный‑6» и начислила проценты, чтобы завершить договор.
* Я, как система, каждый день начисляю проценты по вкладу, чтобы поддерживать актуальное состояние счета по вкладу.

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 счетами (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. Добавить файл README.md в корень решения с кратким описанием назначения сервиса и пошаговой инструкцией запуска (локально и в Docker).

2. Заменить in‑memory хранилище на слой репозитория, сохраняющий счета и транзакции в PostgreSQL (Entity Framework Core 9, миграции code‑first). База данных должна подниматься в контейнере Postgres внутри docker‑compose.

3. Создать индексы для ускорения частых запросов:  
 • hash‑индекс по колонке ownerId в таблице Accounts;  
 • составной индекс (accountId, "date") в таблице Transactions;  
 • GiST‑индекс по колонке "date" в таблице Transactions для выборок по диапазону дат.

4. Создать хранимую процедуру PL/pgSQL accrue\_interest(account\_id UUID) для расчёта процентов по вкладам. Процедура вызывается из обработчика AccrueInterest и участвует в общей транзакции.

5. Операцию перевода средств реализовать внутри явной транзакции EF Core с уровнем изоляции Serializable. При несоответствии итоговых балансов транзакция должна откатываться.

6. Использовать оптимистичную блокировку через concurrency‑token (xmin) либо rowversion‑колонку. При конфликте возвращать HTTP 409 Conflict.

7. Добавить модульные и интеграционные тесты (xUnit/NUnit + Testcontainers‑DotNet):  
 • минимум три модульных теста бизнес‑логики;  
 • интеграционный тест на параллельную обработку запросов - ParallelTransferTests — 50 параллельных переводов при вызове метода http API и проверка сохранности суммарного баланса по итогу выполнения операций.

9. Обновить Docker-Compose (добавляем Postgres).

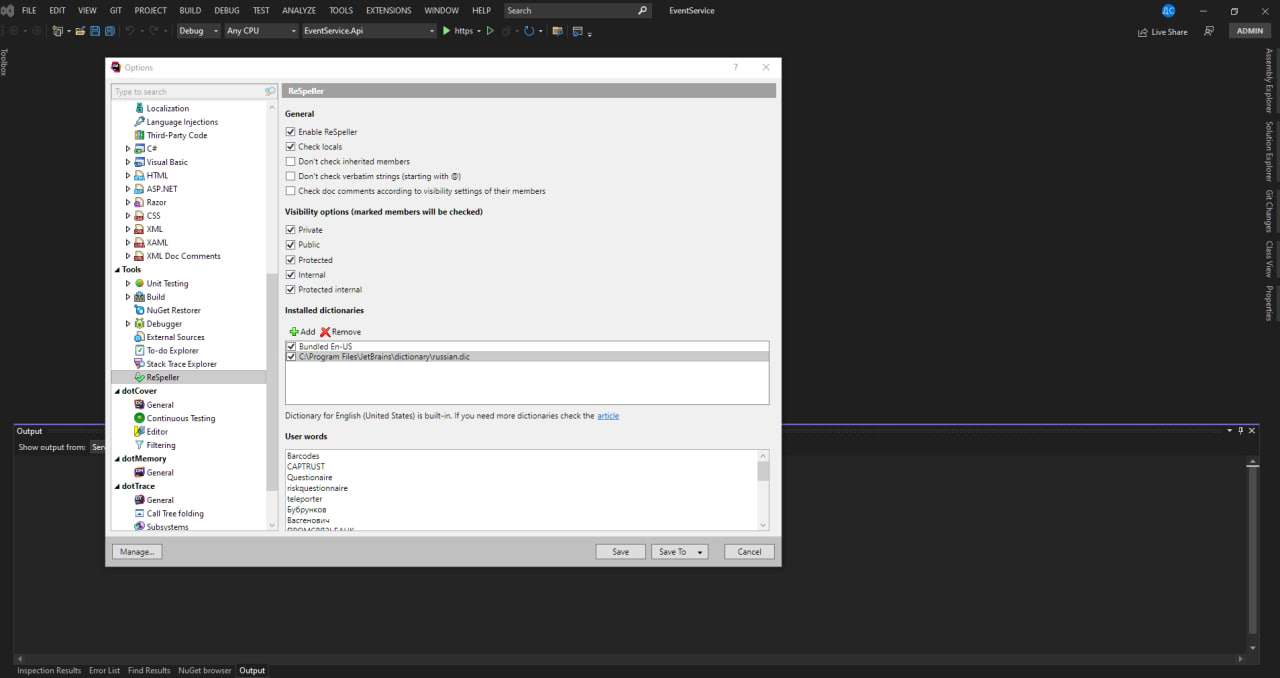
10. Настроить строгую проверку кода: Resharper → Inspect → Code Issues In Solution возвращает 0 ошибок.

11. Обеспечить соответствие структуры проекта рекомендациям из Приложения 3 задания #1 и #2 (vertical slice, папки Features\<FeatureName>\<Operation>).

12. Реализовать ежедневный Cron-Job HangFire по начислению процентов.

**Критерии приемки**:

1. Если выполнить все шаги из README.md, сервис поднимается через docker‑compose, на http://localhost:80 открывается Swagger, а данные сохраняются в контейнере Postgres.
2. Все операции CRUD и бизнес‑действия выполняются и отражаются в базе данных.
3. При повторном запуске контейнеров данные не теряются (volume или named‑volume).
4. Модульные и интеграционные тесты успешно проходят (dotnet test) в чистом окружении, включая ParallelTransferTests (50 параллельных переводов).
5. EXPLAIN ANALYZE для запроса выписки по счёту и периоду демонстрирует использование составного индекса.
6. При одновременной попытке изменить один счёт из двух потоков второй запрос получает 409 Conflict.
7. Баланс счёта не уходит в минус без кредитного лимита; итоговая сумма балансов счётов в тестовом сценарии ParallelTransferTests совпадает с исходной.
8. Resharper не показывает ошибок уровня Error/Warning (исключая подавленные с пояснением).
9. Все методы требуют валидного JWT‑токена; при его отсутствии сервис возвращает 401.
10. Swagger содержит описания методов, моделей, параметров, возможных кодов ответа; XML‑комментарии подгружаются автоматически.
11. REST‑API соблюдает регламент из Приложения 1 (HTTP‑глаголы, формирование URL и т. д.).
12. Сформирован ежедневный Cron-Job Hangfire по начислению процентов

**Приложение 1**  
  
Русификация проверок Resharper  
  


<https://www.jetbrains.com/help/resharper/Spell_Checking.html#respeller-dictionaries>

Будем считать, что этот словарь установлен у всех, в том числе у проверяющих