Príloha B: Zadanie - Backend

Order Processing System

POZOR: Časť 1 a Časť 2 treba robiť oddelene (nie cez jeden veľký prompt naraz)

Časť 1: Rest API part

Features:

- Users module:
    - User has following fields: id, name max length 100, email max length 100 and is unique, password string
    - Create CRUD REST API for this module
        - Validate input DTOs. If wrong return 400
- Authetication module:
    - Login REST API
        - Check user credentials (email, password) and if correct return JWT token
- Products module:
    - Product has following fields: id, name string max length 100, description string, price number >= 0, stock number >= 0, created_at timestamp
    - Create CRUD REST API for this module
        - Validate input DTOs. If wrong return 400
- Orders module:
    - Order has following fields: id, user_id , total number >= 0, status enum (pending, processing, completed, expired), items schema id primary key, product_id, quantity number > 0, price number > 0 created_at timestamp, updated_at timestamp
    - Create CRUD REST API for this module
        - Validate input DTOs. The rules are in scheme
- Additional requirements
    - Endpoints has to be protected with JWT Bearer token. Result of Login REST API.
    - Correctly handle error return states
        - 400 Bad Request
        - 401 Unauthorized
        - 404 Not Found
        - 500 Internal Server Error
    - Include OpenAPI/Swagger documentation
    - Integration tests (minimum 5 test cases)

- Use Postgres DBS. Run Postgres in docker and initialize it with docker compose file. Include docker compose file in the GIT repository.
- Include into the final solution DB upgrade mechanism. It has to contain some form of upgrade DB scripts or DB upgrade code.
- Include into DBS also initial seed data. Can be part of the upgrade menchanism too.
- In Readme.md document how to run DB upgrade tool and how to start the service.

Časť 2: Event-Driven Architecture + Background Processing

- Common requirements:
  - Use some messaging service like RabbitMq, Kafka (, may be Redis ?). Update docker compose file so this service can be created inside docker.
  - Add event bus into the project so the messages/events can be sent/published. As the transport system for event bus use the chosen messaging service.
- Order creation handling:
  - When the order is created the `OrderCreated` event has to be published
  - There will be handling of this event which:
    - Update order status: pending → processing
    - Simulate payment processing (5 second delay)
    - Update order status for 50% of cases to completed and publish `OrderCompleted` event
    - In another 50% of cases do not change the status
- Order expiration handling:
  - Add recursive job which will run every 60 seconds
  - The job find orders with status='processing' older than 10 minutes and update the status to 'expired'
  - Publish `OrderExpired` event
- Notifications handling
  - Create new notifications table and add upgrade script/code
  - When the `OrderCompleted` event is published
    - Send email notification (fake/mock - log to console)
    - Save notification to database (audit trail)
  - When the `OrderExpired` event is published
    - Save notification to database (audit trail)

Expected Flow:

1. User creates order via `POST /api/orders`
2. Order saved to DB with status='pending'
3. `OrderCreated` event published

4. OrderProcessor handles event asynchronously:
    - Updates status to 'processing'
    - Simulates payment (5 sec delay)
    - Updates status to 'completed'
5. `OrderCompleted` event published
6. Notifier handles event:
    - Logs fake email to console
    - Saves notification to DB
7. CRON job runs every 60s:
    - Finds pending orders older than 10 minutes
    - Updates them to 'expired'