Trabalho final

UC-13 Desenvolvimento Back-End

Curso Técnico em Desenvolvimento de Sistemas - Profª Aline Timm Pelotas- RS

Desenvolvimento do Backend para uma Plataforma de E-commerce

Data de Entrega: 02/10/2025

1. Objetivo principal:

O objetivo deste projeto final é aplicar os conceitos aprendidos durante a disciplina para construir o backend completo e funcional para uma aplicação de e-commerce. Vocês receberão um projeto front-end pré-construído que será usado como base para o projeto.

Vocês deverão realizar a modelagem de dados com um banco de dados não relacional.

2. Requisitos Funcionais

A API deve prover os seguintes conjuntos de funcionalidades, organizados por entidade:

2.1. Usuários e Autenticação

- **(CREATE)** POST /users/register: Cadastro de um novo usuário (nome, e-mail, senha). A senha deve ser armazenada de forma segura (hashing).
- **(READ)** GET /users/me: Rota protegida que retorna os dados do usuário logado (exceto a senha).
- **(UPDATE)** PUT /users/me: Rota protegida para o usuário atualizar suas próprias informações (nome, e-mail).
- **(DELETE)** DELETE /users/me: Rota protegida para o usuário deletar sua própria conta.
- **Autenticação:**
  - POST /auth/login: Autentica um usuário com e-mail e senha, retornando um token JWT em caso de sucesso.
  - POST /auth/logout: Invalida o token do usuário (opcional, pode ser gerenciado no front-end).

2.2. Produtos

- **(CREATE)** POST /products: Rota de administrador para cadastrar um novo produto (nome, descrição, preço, categoria, estoque, imagem URL).
- **(READ)** GET /products: Listagem de todos os produtos com suporte a filtros básicos (ex: por categoria) e paginação.

- **(READ)** GET /products/{id}: Busca e retorna os detalhes de um produto específico pelo seu ID.
- **(UPDATE)** PUT /products/{id}: Rota de administrador para atualizar as informações de um produto.
- **(DELETE)** DELETE /products/{id}: Rota de administrador para remover um produto.

2.3. Carrinho de Compras

- **(CREATE/UPDATE)** POST /cart/add: Adiciona um produto (e sua quantidade) ao carrinho do usuário logado. Se o produto já estiver no carrinho, atualiza a quantidade.
- **(READ)** GET /cart: Retorna o conteúdo completo do carrinho do usuário logado.
- **(UPDATE)** PUT /cart/update/{productId}: Atualiza a quantidade de um item específico no carrinho.
- **(DELETE)** DELETE /cart/remove/{productId}: Remove um item do carrinho.

2.4. Pedidos

- **(CREATE)** POST /orders: Cria um novo pedido a partir dos itens do carrinho do usuário. Esta ação deve:
  1. Verificar o estoque dos produtos.
  2. "Limpar" o carrinho do usuário.
  3. Salvar o pedido com os produtos, quantidades, preço total e status (ex: "Processando").
  4. Decrementar a quantidade do estoque dos produtos vendidos.
- **(READ)** GET /orders: Retorna o histórico de pedidos do usuário logado.
- **(READ)** GET /orders/{id}: Retorna os detalhes de um pedido específico do usuário.

3. Requisitos Não Funcionais

- **Validação de Dados:** Todos os dados de entrada (body, params, query) devem ser validados. Por exemplo, garantir que o e-mail tenha um formato válido, que senhas tenham um comprimento mínimo, e que preços e estoques sejam números positivos. Bibliotecas como Joi, express-validator ou validação nativa do framework são recomendadas.
- **Segurança:**
  - Implementar autenticação baseada em token JWT para proteger rotas.
  - As senhas dos usuários devem ser criptografadas (hashing com salt) usando bibliotecas como bcrypt.
  - Implementar controle de acesso (RBAC - Role-Based Access Control) simples: rotas de criação/alteração de produtos devem ser acessíveis apenas por usuários com perfil de "Administrador".
  - Proteger contra vulnerabilidades comuns (ex: NoSQL Injection).
- **Tratamento de Erros:** A API deve retornar códigos de status HTTP apropriados (200, 201, 400, 401, 403, 404, 500) e mensagens de erro claras em formato JSON.
- **Banco de Dados Não Relacional:** A modelagem do banco de dados deve ser pensada para NoSQL. Por exemplo, pode-se optar por embutir (embed) informações do produto no pedido em vez de apenas referenciá-las, para manter a consistência histórica do pedido.

4. Documentação

A API deve ser bem documentada. A documentação deve ser feita da seguinte forma:

Arquivo README.md com:

- Descrição do projeto.
- Instruções claras sobre como configurar e rodar o projeto localmente (variáveis de ambiente, instalação de dependências, etc.).
- Descrição de cada endpoint, incluindo:
  - Método HTTP e URL.
  - Descrição da funcionalidade.
  - Parâmetros (se houver).
  - Formato do corpo da requisição (payload).
  - Exemplos de respostas de sucesso e de erro.

O uso de ferramentas como o Swagger/OpenAPI para gerar documentação interativa é um diferencial e será valorizado.

5. Apresentação Final

Os alunos deverão realizar uma apresentação de **15 minutos** na qual devem:

1. **Explicar a Arquitetura:** Descrever a estrutura do projeto, as tecnologias utilizadas e, principalmente, justificar a modelagem de dados escolhida para o banco não relacional.
2. **Demonstração ao Vivo:** Realizar uma demonstração prática da API funcionando em conjunto com o front-end fornecido. Devem ser demonstradas as principais funcionalidades, como cadastro de usuário, login, adição de produtos ao carrinho e finalização de um pedido.
3. **Desafios e Aprendizados:** Comentar sobre os principais desafios encontrados durante o desenvolvimento e os principais aprendizados obtidos.

Critérios de Avaliação

- **Funcionalidade (40%):** Todos os requisitos funcionais foram implementados e estão funcionando corretamente.
- **Qualidade do Código e Boas Práticas (20%):** Código limpo, bem organizado.
- **Segurança e Validação (15%):** Implementação correta de autenticação, hashing de senhas e validação de dados de entrada.
- **Documentação (15%):** Qualidade e clareza da documentação.
- **Apresentação (10%):** Clareza, objetividade e qualidade da apresentação final e da demonstração.