Mini E-commerce Java Backend Task

Task 1: REST API Backend

Requirements

Create a Spring Boot REST API that serves the product data:

Core Endpoints:

- GET /api/products - List all products with pagination, sorting, and filtering
- GET /api/products/{id} - Get single product
- POST /api/cart/add - Add item to cart
- GET /api/cart - Get cart contents
- PUT /api/cart/item/{id} - Update cart item quantity
- DELETE /api/cart/item/{id} - Remove item from cart

Technical Requirements:

- Use Spring Boot 3.x
- Implement proper HTTP status codes
- Add input validation with Bean Validation
- Use Spring Data JPA with your preferred database (PostgreSQL, MySQL, MongoDB, etc.)
- Implement CORS for frontend integration
- API documentation with OpenAPI/Swagger
- Unit tests

Task 2: Authentication & Security

Requirements

Extend the basic API with:

- JWT-based authentication
- User registration/login endpoints
- Secure cart operations (user-specific carts) - *See detailed section below*
- Role-based access (USER, ADMIN)

Secure Cart Operations - Detailed Requirements

User-Specific Cart Implementation:

1. **Cart Isolation:**
    a. Each user must have their own isolated cart
    b. Cart items should be tied to authenticated user ID
2. **Authentication Requirements:**
    a. All cart operations require valid JWT token
    b. Extract user ID from JWT claims
    c. Validate user permissions for cart access
3. **Security Implementation Details:**
    a. **JWT Token Validation:** Validate token on every cart operation
    b. **User Authorization:** Check if the requesting user owns the cart/cart item
    c. **Input Sanitization:** Validate all input parameters

Task 3: Integration with Frontend

Frontend Integration Points

- Update the React application to consume the Java API
- Implement proper authentication flow
- Add error handling for API failures
- Show loading states during API calls
- Handle JWT token storage and refresh

Success Criteria

You will be evaluated on:

- Code quality and organization of what you deliver
- Problem-solving approach and technical decisions
- Clear communication about challenges and solutions
- Not on completing 100% of requirements

Important: Focus on What You Can Deliver

This is NOT an all-or-nothing assessment! We understand that some requirements might be challenging or time-consuming. Here's our approach:

Priority Guidelines

- **Core Functionality First:** Focus on getting the API endpoints first
- **Incremental Progress:** Implement features in order of importance
- **Quality Over Quantity:** A well-implemented subset is better than a buggy complete solution
- **Document Your Thinking:** For unfinished features, explain your approach

What to Do If You Can't Complete Everything

If you run out of time or get stuck:

- Submit what you have - partial implementations are valuable, just make sure they don’t break the rest of the application
- Document incomplete features in your README or code comments
- Explain your approach for features you didn't finish
- Show your problem-solving process - we want to see your approach to software development, not just your coding speed

Submission Requirements

- Code: GitHub repository with clear README file
- Documentation:
    - Setup instructions
    - Architecture decisions
    - Self-Assessment: Brief reflection on:
        - Challenges faced and how you solved them
        - What you would improve with more time

Task delivery and support

- You have **3 days** from receiving the task to deliver your solution
- Final solution should be sent to: Elena.Gvoka@bosch.com
- If implementation details are unclear or missing, use your best judgment - implement features as you think they should work by following industry best practices. If still unclear, don't hesitate to contact us via e-mail: Elena.Gvoka@bosch.com