Amazon-like E-commerce Application

A full-featured Spring Boot e-commerce application with LocalStack S3 image storage, Redis caching, ELK stack logging, and comprehensive checkout flow.

🚀 Features

Core Functionality

- **Product Catalog**: 35+ products across 12 categories with images
- **Shopping Cart**: Add/remove items, update quantities
- **Checkout Flow**: Multi-step checkout (shipping, payment, review)
- **Order Management**: Complete order history and tracking
- **Search**: Full-text product search
- **User Management**: Pre-populated test users

Advanced Features

- **LocalStack S3**: Local AWS S3 for image storage (35 unique colored SVG images)
- **Redis Caching**: 10-minute TTL for fast page loads
- **ELK Stack Integration**: Elasticsearch, Logstash, Kibana for log management
- **Responsive UI**: Amazon-inspired design with Thymeleaf

🛠️ Tech Stack

Backend

- Spring Boot 4.0.0
- Spring Data JPA
- Spring Security
- Spring Data Redis
- Spring Elasticsearch
- AWS SDK for S3

Database & Storage

- PostgreSQL (primary database)
- Redis (caching layer)
- LocalStack S3 (image storage)

Monitoring & Logging

- Elasticsearch 8.11.0
- Logstash 8.11.0
- Kibana 8.11.0
- Logstash Logback Encoder

Frontend

- Thymeleaf
- HTML5/CSS3
- Vanilla JavaScript

📋 Prerequisites

1. **Java 17** or higher
2. **PostgreSQL** installed and running locally
3. **Docker & Docker Compose** (for LocalStack, Redis, ELK)
4. **Gradle** (included via wrapper)

🔧 Setup & Installation

1. Database Setup

Create PostgreSQL database:

```
psql -U postgres
CREATE DATABASE ecommerce;
CREATE USER aerloki WITH PASSWORD '';
GRANT ALL PRIVILEGES ON DATABASE ecommerce TO aerloki;
\q
```

2. Start Docker Services

Start all services (LocalStack S3, Redis, ELK stack):

```
docker-compose up -d
```

Verify services are running:

```
docker ps
```

You should see:

- `localstack` (port 4566) - S3 storage
- `redis` (port 6379) - Caching
- `elasticsearch` (port 9200) - Search & logs
- `logstash` (port 5000) - Log aggregation
- `kibana` (port 5601) - Log visualization

3. Run the Application

```
```

```
./gradlew bootRun
```

Wait for:

```
✓ Created S3 bucket: product-images
✓ Uploaded 35 product images to S3
✓ Created 35 products with LocalStack S3 images
```

4. Access the Application

- **Web Application**: http://localhost:8081
- **LocalStack S3**: http://localhost:4566
- **Elasticsearch**: http://localhost:9200
- **Kibana**: http://localhost:5601
- **Redis**: localhost:6379

📦 Docker Services

LocalStack S3 (Image Storage)

- **Port**: 4566
- **Bucket**: `product-images`
- **Images**: 35 colored SVG placeholders
- **URL Format**: `http://localhost:4566/product-images/product-{1-35}.svg`

Redis (Caching)

- **Port**: 6379
- **TTL**: 10 minutes
- **Cached Methods**:
  - `getAllProducts()`
  - `getAvailableProducts()`
  - `getProductById()`
  - `getProductByAsin()`
  - `getProductsByCategory()`
  - `searchProducts()`

ELK Stack (Logging)

- **Elasticsearch**: Port 9200
- **Logstash**: Port 5000
- **Kibana**: Port 5601
- **Log Format**: JSON with timestamps
- **Features**: Real-time log streaming and search

🗂️ Project Structure

```
src/
├── main/
│   ├── java/com/aerloki/personal/project/Personal/Project/
│   │   ├── model/
│   │   │   ├── Product.java          # Product entity
│   │   │   ├── User.java             # User entity
│   │   │   ├── Order.java            # Order entity
│   │   │   ├── OrderItem.java        # Order items
│   │   │   ├── CartItem.java         # Shopping cart items
│   │   │   └── CheckoutSession.java  # Checkout state
│   │   ├── repository/
│   │   │   ├── ProductRepository.java
│   │   │   ├── UserRepository.java
│   │   │   └── OrderRepository.java
│   │   ├── service/
│   │   │   ├── ProductService.java   # @Cacheable methods
│   │   │   ├── CartService.java
│   │   │   ├── OrderService.java
│   │   │   └── S3Service.java        # LocalStack S3 operations
│   │   ├── controller/
│   │   │   ├── HomeController.java
│   │   │   ├── ProductController.java
│   │   │   ├── CartController.java
│   │   │   ├── CheckoutController.java
│   │   │   └── OrderController.java
│   │   └── config/
│   │       ├── DataInitializer.java     # DB initialization
│   │       ├── S3Config.java            # LocalStack S3 config
│   │       ├── S3ImageInitializer.java  # S3 image upload
│   │       ├── RedisConfig.java         # Redis cache config
│   │       └── SecurityConfig.java
│   └── resources/
│       ├── application.properties
│       ├── logback-spring.xml         # Logstash logging config
│       └── templates/                 # Thymeleaf HTML templates
│           ├── index.html
│           ├── cart.html
│           ├── checkout-*.html
│           └── orders.html
```

🎯 Key Features Explained

LocalStack S3 Image Storage

Images are stored in a local S3-compatible service (LocalStack):

- Automatically creates `product-images` bucket on startup
- Uploads 35 unique colored SVG images

- Each image displays "Product {number}" with a different background color
- No external dependencies - all images served locally

Verify S3 images:

```
# View an image
curl http://localhost:4566/product-images/product-1.svg

# List all images
aws --endpoint-url=http://localhost:4566 s3 ls s3://product-images/
```

Redis Caching

Reduces database load and improves page performance:

- **Cache Keys**: Product queries cached for 10 minutes
- **Eviction**: Automatic after TTL expires
- **Benefit**: Subsequent page loads are nearly instant

Monitor cache:

```
# View cached keys
docker exec -i redis redis-cli KEYS "*"

# Clear cache
docker exec -i redis redis-cli FLUSHALL
```

ELK Stack Logging

All application logs are sent to Elasticsearch:

1. **Logback** captures logs in JSON format
2. **Logstash** receives logs via TCP (port 5000)
3. **Elasticsearch** stores and indexes logs
4. **Kibana** provides search and visualization

Access Kibana: http://localhost:5601

📊 Sample Data

Products (35 items)

- **Electronics**: Speakers, streaming devices, e-readers, headphones, smart watches, power banks, mice
- **Books**: Self-help, fiction, psychology, history
- **Home & Kitchen**: Pressure cookers, blenders, coffee makers, air fryers, robot vacuums
- **Sports & Outdoors**: Water bottles, yoga mats, fitness trackers, resistance bands

- **Fashion**: Jeans, handbags, sneakers, sunglasses, jackets, watches
- **Beauty & Personal Care**: Moisturizers, hair dryers, toothbrushes, serums
- **Gaming**: Consoles (PS5, Switch), controllers, headsets
- **Toys & Games**: LEGO sets, puzzles, board games
- **Pet Supplies**: Automatic feeders, grooming tools
- **Tools**: Drills, vacuums
- **Automotive**: Dash cams, floor mats
- **Office**: Standing desks, ergonomic chairs

Users (3 test accounts)

- john.doe@example.com
- jane.smith@example.com
- bob.jones@example.com

🚦 Starting & Stopping

Start All Services

```
# Start Docker services
docker-compose up -d

# Start application
./gradlew bootRun
```

Stop All Services

```
# Stop application (Ctrl+C in terminal)

# Stop Docker services
docker-compose down
```

Quick Start Script

```
# Use the provided script
./start-all.sh
```

Quick Stop Script

```
./stop-all.sh
```

🔍 Troubleshooting

LocalStack S3 Not Connecting

```
# Check LocalStack status
docker logs localstack

# Restart LocalStack
docker-compose restart localstack
```

Redis Connection Issues

```
# Check Redis status
docker exec -i redis redis-cli PING
# Expected output: PONG

# Restart Redis
docker-compose restart redis
```

Clear Cache After Changes

```
# Clear Redis cache
docker exec -i redis redis-cli FLUSHALL

# Clear product data
psql -h localhost -U aerloki -d ecommerce -c "DELETE FROM order_items;
DELETE FROM orders; DELETE FROM products;"
```

View Application Logs

```
# Via Kibana
open http://localhost:5601

# Via Docker
docker logs -f logstash
```

🌐 API Endpoints

Products

- `GET /api/products` - All available products
- `GET /api/products/{id}` - Product by ID

- `GET /api/products/asin/{asin}` - Product by ASIN
- `GET /api/products/category/{category}` - Products by category
- `GET /api/products/search?keyword={keyword}` - Search products

Web Pages

- `GET /` - Home page with products
- `GET /cart` - Shopping cart
- `GET /checkout` - Checkout flow (address, payment, review)
- `GET /orders` - Order history
- `GET /orders/{id}` - Order details

🔐 Security Note

Current configuration has security disabled for demo purposes. In production, you should:

1. Enable Spring Security authentication
2. Add proper user authentication
3. Secure API endpoints
4. Use HTTPS
5. Implement proper session management

📝 Configuration

Application Ports

- Application: **8081**
- LocalStack S3: **4566**
- PostgreSQL: **5432**
- Redis: **6379**
- Elasticsearch: **9200**
- Logstash: **5000**
- Kibana: **5601**

Database

- **URL**: jdbc:postgresql://localhost:5432/ecommerce
- **Username**: aerloki
- **Password**: (empty)

Redis Cache

- **Host**: localhost
- **Port**: 6379
- **TTL**: 600000ms (10 minutes)

🧪 Testing

Access different features:

1. **Browse Products**: http://localhost:8081
2. **Add to Cart**: Click "Add to Cart" on any product
3. **View Cart**: Click cart icon in header
4. **Checkout**: Complete multi-step checkout process
5. **View Orders**: Click "Returns & Orders" in header

📈 Performance Features

1. **Redis Caching**: Product queries cached for 10 minutes
2. **Connection Pooling**: HikariCP for database connections
3. **LocalStack S3**: Fast local image serving
4. **Lazy Loading**: JPA entities optimized with proper fetch strategies

🪲 Known Issues

- LocalStack S3 images require the LocalStack container to be running
- Redis cache should be cleared after database changes
- Some Spring Data Redis warnings (expected with multi-module setup)

📚 Additional Documentation

- `SETUP_GUIDE.md` - Detailed setup instructions
- `ELK_SETUP_GUIDE.md` - ELK stack configuration
- `ELASTICSEARCH_SEARCH_GUIDE.md` - Elasticsearch usage
- `QUICK_START.md` - Quick start guide

🤝 Contributing

This is a personal learning project demonstrating:

- Spring Boot best practices
- Microservices patterns (S3, Redis, ELK)
- E-commerce application architecture
- RESTful API design

📄 License

This is a demo project for educational purposes.

🎓 Learning Objectives

This project demonstrates:

- **Spring Boot** application structure
- **LocalStack** for local AWS service emulation
- **Redis** for application caching
- **Docker Compose** for multi-container setup
- **ELK Stack** for centralized logging
- **JPA/Hibernate** for database operations

- **Thymeleaf** for server-side rendering
- **RESTful API** design patterns

Author: aerloki
Created: 2025
Version: 2.0.0 (with LocalStack S3 & Redis caching)