ShopHub E-commerce Platform

Sprint 6 — Final Documentation

Course: Engenharia de Software 2025/2026

Authors:

108548	Diogo Martins	(Scrum Master)
108583	Luis Sousa	(Product Owner)
108598	Ricardo Dias	(Service Analyst)
108647	Sérgio Correia	(DevOps Engineer)
113085	Simão Almeida	(QA Engineer)

December 10, 2025

Abstract

ShopHub is a modern, full-stack e-commerce platform developed as part of the Software Engineering course at Universidade de Aveiro. The system implements enterprise-grade technologies and best practices, providing a complete online shopping experience with product browsing, user authentication, shopping cart management, order processing, and an AI-powered conversational assistant.

The platform architecture follows a microservices-oriented approach with a React-based frontend, Spring Boot backend, PostgreSQL database, and integration with external services including Keycloak for authentication, Ollama for AI capabilities, and Flagsmith for feature flag management. The system implements comprehensive observability through OpenTelemetry, Prometheus, and Grafana, ensuring production-ready monitoring and alerting.

This report documents the complete system architecture, user personas, core functionality through user stories, deployment procedures, and provides comprehensive onboarding instructions for new developers. The project demonstrates the application of software engineering principles including SOLID design, CI/CD automation, zero-downtime deployments, and MLOps practices for the AI chatbot component.

Acknowledgments

This work was developed under the guidance of Professor João Almeida and Assistant Raquel Paradinha, whose supervision and feedback contributed significantly to the project outcomes. Acknowledgement is extended to the open-source community for the tools and frameworks that enabled the implementation of this platform.

Contents

Chapter 1

Introduction

1.1 Context

ShopHub is a modern, full-stack e-commerce platform developed as part of the Software Engineering course at Universidade de Aveiro (2025-2026). The project implements enterprise-grade technologies and best practices to create a production-ready online shopping system.

The platform provides a complete shopping experience with product browsing, user authentication, shopping cart management, order processing, and an AI-powered conversational assistant. ShopHub supports multiple user personas (customers, administrators, content managers, warehouse staff) with distinct requirements and access controls, demonstrating real-world complexity in authorization and workflow management.

1.2 Motivation

Modern software engineering requires practical experience with distributed systems, DevOps practices, and production-grade infrastructure. Traditional academic projects often focus on isolated features without addressing the complexity of real-world deployment, observability, and operational challenges.

ShopHub addresses this gap by implementing a complete e-commerce platform that mirrors industry standards. The project provides hands-on experience with the full software development lifecycle: from architecture design and implementation to automated deployment and production monitoring. By tackling challenges such as zero-downtime deployments, distributed tracing, authentication systems, and AI integration, the team develops skills directly applicable to professional software engineering roles.

1.3 Document Structure

Chapter 2 defines user personas and core user stories. Chapter 3 describes system architecture. Chapter 4 provides developer onboarding. Chapter 5 details production configuration. Chapter 6 concludes with achievements and future work.

Chapter 2

Personas and Core User Stories

This chapter defines the users who interact with ShopHub and the core functionality that serves their needs. Understanding these users and their requirements is essential for designing a system that addresses real-world e-commerce workflows across different roles and responsibilities.

2.1 User Personas

2.1.1 Ana Silva — Customer

Ana is a 28-year-old marketing manager living in Lisbon with high tech savviness and an annual income of €35,000. She shops online 2-3 times per month and prefers mobile browsing during her commutes and breaks. Ana values convenience and efficiency, expecting seamless digital experiences that respect her time.

> *"I don't have time to dig through cluttered websites. I need to find what I want, buy it, and get back to work."*

Goals & Motivations: Ana wants to find products quickly using intuitive search and filtering. She needs to compare products and read customer reviews before making decisions. She also appreciates personalized product recommendations based on her browsing and purchase history.

How ShopHub Helps: ShopHub provides fast, intuitive search and filtering that helps Ana find exactly what she needs. The AI-powered chatbot answers product questions instantly without waiting for customer support. Comprehensive product details and authentic customer reviews help her make confident purchase decisions.

2.1.2 João Costa — Administrator

João is a 35-year-old e-commerce manager in Porto with very high tech savviness and 8 years of experience in e-commerce operations. He manages a team of 5 staff members and is responsible for the entire platform operation. João constantly seeks ways to optimize workflows and improve efficiency through data-driven decision making.

> *"I need real-time visibility into our operations. Manual processes and delayed reports cost us sales."*

Goals & Motivations: João wants to manage the product catalog efficiently with streamlined tools. He needs to monitor sales and inventory in real-time to prevent stockouts and identify opportunities. He wants to analyze business metrics and trends to make informed decisions.

How ShopHub Helps: ShopHub provides a comprehensive admin dashboard with real-time visibility into all operations. Low stock alerts help João prevent stockouts and manage inventory proactively. Sales reports and analytics give him the data needed for informed business decisions. The streamlined order management interface improves team productivity and order processing speed.

2.1.3 Maria Santos — Content Manager

Maria is a 30-year-old digital marketing specialist in Coimbra with medium-high tech savviness. She holds a Marketing degree and has 5 years of experience in content creation. Maria manages all promotional content for the platform, including landing pages and blog posts, constantly testing new strategies to improve engagement and conversions.

> *"I shouldn't need a developer to change a banner or publish a blog post. Content moves fast, systems should too."*

Goals & Motivations: Maria wants to create engaging landing pages that drive conversions. She needs to publish blog content regularly to improve SEO and engage customers. Managing promotional banners and campaigns efficiently is crucial for her marketing calendar. She wants to update content without requiring technical assistance.

How ShopHub Helps: ShopHub provides an intuitive CMS interface that empowers Maria to work independently without developer assistance. The landing page builder with markdown support gives her creative flexibility while maintaining consistency. The blog management system with scheduling capabilities lets her plan campaigns in advance.

2.1.4 Pedro Almeida — Warehouse Staff

Pedro is a 42-year-old warehouse supervisor in Aveiro with medium tech savviness and 15 years of experience in logistics and fulfillment. He works 8-hour shifts, 5 days per week, overseeing daily warehouse operations and ensuring orders are picked, packed, and shipped accurately.

> *"Every minute I spend hunting for order details or fixing inventory mistakes is a minute an order sits waiting."*

Goals & Motivations: Pedro wants to process orders efficiently with simple workflows. He needs to track inventory accurately in real-time to avoid picking errors and stockouts. Minimizing picking and packing errors is crucial for customer satisfaction. He also needs to coordinate seamlessly with shipping carriers to ensure timely deliveries.

How ShopHub Helps: ShopHub provides a dedicated warehouse interface optimized for fulfillment operations. Digital order management eliminates paperwork and reduces errors. Real-time inventory updates across all systems prevent discrepancies. A clear order status workflow with state transitions keeps everyone aligned.

2.2 User Story 1: Browse Product Catalogs

As a customer, I want to browse products organized by categories so that I can discover items relevant to my interests.

Acceptance Criteria:

- **AC-1:** Given the customer visits the home page or catalog section, when they navigate through categories or scroll the catalog, then the system displays available products grouped by category.
- **AC-2:** Given the catalog contains multiple pages, when the customer scrolls or clicks "next page", then additional products are loaded and displayed.
- **AC-3:** Given filters (price, brand, size) are available, when the customer applies them, then the product list updates according to the selected filters.

2.3 User Story 4: Add Items to Shopping Cart

As a customer, I want to add products to my shopping cart so that I can purchase multiple items in a single transaction.

Acceptance Criteria:

- **AC-1:** Given the customer opens a product page, when they click to add the product to the cart, then the cart stores that product along with any previous ones.
- **AC-2:** Given the customer adds a product to the cart, when the action is successful, then the cart icon updates with the new total quantity.
- **AC-3:** Given the product is out of stock, when the customer tries to add it to the cart, then the system prevents the action and shows an error message.

2.4 User Story 12: Manage Product Listings

As a store administrator, I want to manage product listings in the catalog so that customers can view accurate product information and availability.

Acceptance Criteria:

- **AC-1:** Given the administrator is logged into the dashboard, when they add, edit, or delete a product, then the system updates the catalog with the product's name, description, price, images, stock, and attributes (size, color, etc.).
- **AC-2:** Given a product exists, when it is managed, then it must contain name, description, price, images, stock, and attributes (size, color, etc.).

2.5 User Story 20: Manage Blog Posts

As a content manager, I want to create and publish blog posts so that I can engage customers with relevant content and improve SEO.

Acceptance Criteria:

- **AC-1:** Given a Content Manager is logged in, when they create a new blog post, then they can define the title, content, and metadata fields.
- **AC-2:** Given the blog post is published, when a visitor accesses it, then all content displays correctly.
- **AC-3:** Given a Content Manager is creating a blog post, when they assign categories and tags, then these are saved and displayed correctly.

2.6 User Story 22: View New Orders

As a warehouse staff member, I want to view new orders awaiting fulfillment so that I can prepare shipments efficiently.

Acceptance Criteria:

- **AC-1:** Given the warehouse staff is logged into the system, when they navigate to the "New Orders" section, then the system displays a list of all pending orders with their order IDs, customer details, and required products.
- **AC-2:** Given the list of new orders is long, when the warehouse staff uses filters (date, status, customer), then the system narrows down the displayed orders.
- **AC-3:** Given new orders arrive while the staff is viewing the list, when the page refreshes or notifications appear, then the new orders are highlighted at the top of the list.

2.7 User Story 28: Conversational Assistant

As a customer, I want to interact with a conversational assistant so that I can get product recommendations and support.

Acceptance Criteria:

- **AC-1:** Given the user opens the application, when they access the conversational assistant interface, then the assistant is visible and ready to receive input.
- **AC-2:** Given the user asks a question, when they submit the query, then the assistant provides a relevant response within 3 seconds.
- **AC-3:** Given the user interacts with the assistant, when any conversation occurs, then all interactions are logged for analytics purposes.

Chapter 3

System Architecture

3.1 Architectural Overview

ShopHub implements a traditional three-tier monolithic architecture, deployed as containerized services via Docker and orchestrated with Terraform. The core application consists of a single React SPA frontend, a Spring Boot backend monolith with layered architecture (controllers, services, repositories, entities), and a PostgreSQL relational database. The system integrates with external services: Keycloak for OAuth2 authentication, Flagsmith for feature flag management, and Ollama for AI-powered chatbot capabilities. Figure 3.1 shows the complete system structure.

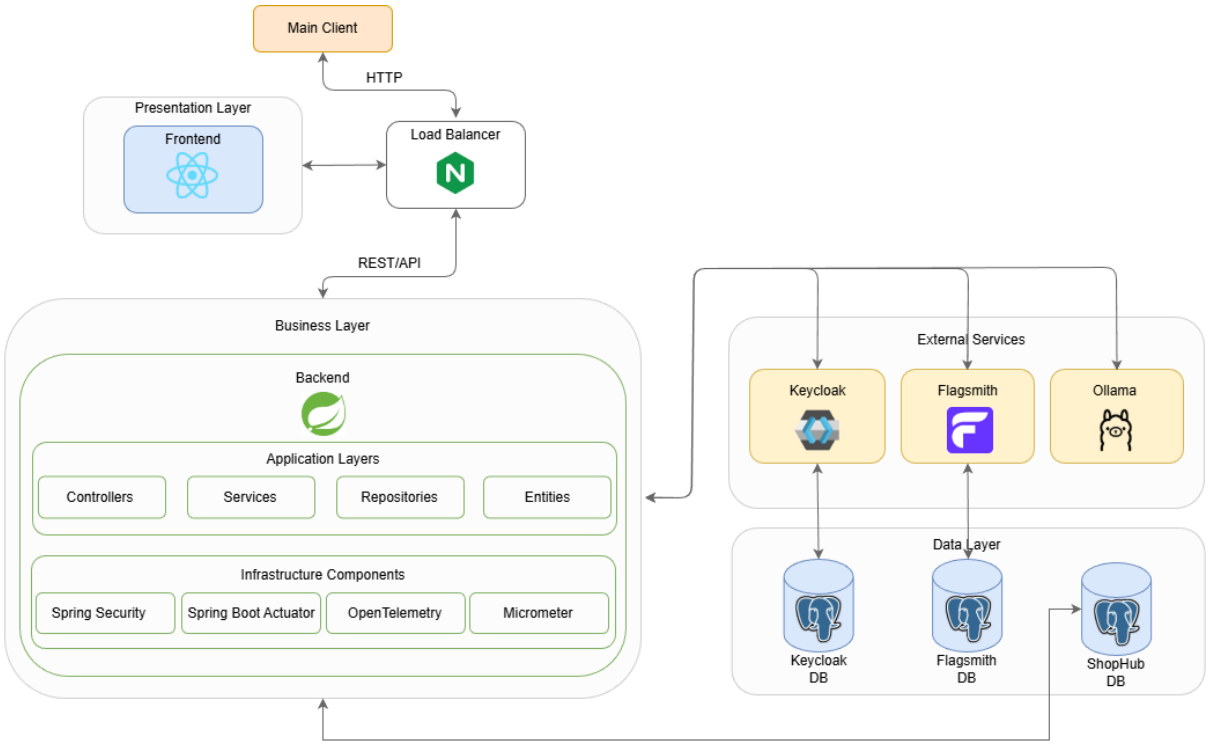

Figure 3.1: ShopHub System Architecture

3.1.1 Presentation Layer

The frontend is a React 19.2.0 single-page application built with TailwindCSS, shadcn component library and Lucide icons. It integrates Keycloak JS for OAuth2 authentication, Flagsmith SDK for client-side feature flags, and OpenTelemetry Web SDK for telemetry. The application communicates with the backend through RESTful APIs using Axios, with all requests authenticated via JWT bearer tokens. The frontend is served as a static build deployed in NGINX containers.

3.1.2 Business Layer

The backend is a Spring Boot 3.2.0 monolith running on Java 17, organized in a traditional layered architecture with controllers, services, repositories, and JPA entities. Controllers expose REST endpoints and handle HTTP requests. Services implement business logic including product management, order processing, shopping cart operations, and chatbot integration. Repositories provide data access through Spring Data JPA.

The backend integrates with external services: Keycloak for JWT token validation and authentication, Flagsmith for server-side feature flag evaluation, and Ollama for AI-powered chatbot responses using the Gemma3 model with RAG (Retrieval-Augmented Generation).

Key infrastructure components include Spring Security configured as an OAuth2 resource server, Spring Boot Actuator for health checks and metrics endpoints, OpenTelemetry Java agent for distributed tracing, and Micrometer for metrics collection.

3.1.3 Data Layer

The data layer consists of three separate PostgreSQL 15 instances for isolation and security. The application database (`shophub`) stores core business data including products, categories, orders, shopping carts, users, reviews, blog posts, landing pages, and chatbot conversation history. The Keycloak database maintains authentication data including user credentials, roles, sessions, and client configurations. The Flagsmith database stores feature flag configurations, environments, and analytics.

3.1.4 External Services

Keycloak

Keycloak 25.0.6 serves as the OAuth2/OIDC identity provider for the platform. It manages the ShopHub realm with four distinct roles: customer, admin, content_manager, and warehouse_staff. Keycloak issues JWT tokens upon successful authentication and exposes a JWKS endpoint for token signature validation by the backend.

Flagsmith

Flagsmith provides feature flag management, enabling runtime feature toggles without code deployments. The service supports A/B testing, environment-based configurations, and gradual feature rollouts. Both frontend and backend integrate with Flagsmith to evaluate feature states dynamically.

Ollama

Ollama hosts the Gemma3:1b large language model for the conversational chatbot. The backend ProductContextService implements Retrieval-Augmented Generation by enriching user queries with relevant product data before sending prompts to Ollama. Conversations are persisted in the database for analytics and model improvement.

Load Balancer

NGINX acts as a reverse proxy and load balancer, serving as the single entry point. It distributes traffic across two backend replicas and two frontend replicas using round-robin load balancing. NGINX also handles static file serving and health check routing.

3.2 Dependencies

As illustrated in Figure 3.2, the system is organized according to the following dependency graph.

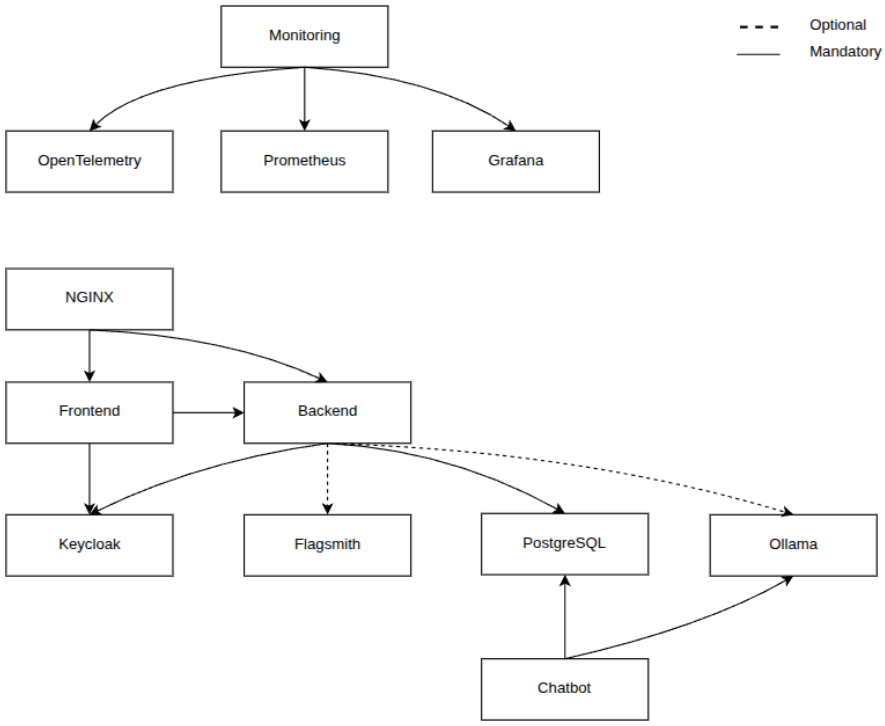

Figure 3.2: Dependencies Graph

- **Backend** depends on: PostgreSQL, Keycloak, Flagsmith (optional, fails gracefully), Ollama (optional, chatbot fails gracefully)
- **Frontend** depends on: Backend API, Keycloak
- **NGINX** depends on: Frontend replicas, Backend replicas
- **Chatbot** depends on: Ollama, PostgreSQL
- **Monitoring** depends on: OpenTelemetry Collector, Prometheus, Grafana

3.3 Critical Flows

3.3.1 Request Flow

All user requests enter through NGINX, which load balances between frontend replicas. The frontend serves the React SPA, which authenticates users via Keycloak and then makes API calls to the backend through the load balancer. Backend requests are authenticated by validating JWT tokens against Keycloak's JWKS endpoint. The backend queries PostgreSQL for data persistence, checks feature flags via Flagsmith, and invokes Ollama for chatbot requests. All responses flow back through the same path to the user.

3.3.2 Authentication Flow

When a user logs in, the frontend redirects to Keycloak's login page. After successful authentication, Keycloak issues a JWT token containing user identity and roles. The frontend stores this token and includes it in the Authorization header for all API requests. The backend validates the token signature using Keycloak's public keys and extracts user roles for authorization decisions. Role-based access control restricts endpoints based on user roles: customer, admin, content_manager, or warehouse_staff.

3.3.3 Chatbot Interaction Flow

When a user submits a question, ChatbotService manages the conversation session and saves the user message. ProductContextService implements simplified RAG by extracting keywords and searching the product repository to find up to 5 matching products via SQL LIKE queries. PromptService renders a template combining product context, conversation history (last 5 messages), and the user query. OllamaService sends the enriched prompt to Ollama's API with retry logic. The response is validated by ChatOutputValidator for content appropriateness before being persisted with performance metrics and returned to the frontend.

Chapter 4

Developer Onboarding

This chapter provides a comprehensive guide for setting up the ShopHub development environment on a clean Ubuntu/Debian Linux system. The setup process uses automated deployment through GitHub Actions, requiring minimal manual configuration.

4.1 Requirements

Before beginning the installation process, ensure the following requirements are met:

- Ubuntu or Debian Linux operating system
- GitHub account with access to the repository
- Server IP address for configuration

Note: All configuration settings including IP addresses, passwords, and API keys are managed through a `.env` file that will be configured during the installation process.

4.2 Installation

4.2.1 Step 1: Install Docker

Docker is the primary containerization platform for ShopHub. Install Docker using the official installation script:

```
curl -fsSL https://get.docker.com -o get-docker.sh
sudo sh get-docker.sh --version 28.0.4
sudo usermod -aG docker $USER
newgrp docker
```

4.2.2 Step 2: Install Unzip

The unzip utility is required for extracting archived files during deployment:

```
sudo apt install unzip
```

4.2.3 Step 3: Setup GitHub Actions Runner

Navigate to https://github.com/detiuaveiro/group-project-es2526_203/settings/actions/runners/new and execute the commands displayed on the page.

During the runner configuration, provide the following values when prompted:

- **Runner group**: Press Enter to accept the default
- **Runner name**: `es-vm-ecommerce`
- **Labels**: `Linux,X64,es-vm-ecommerce`
- **Work folder**: Press Enter to accept the default

After configuration, install and start the runner as a system service:

```
sudo ./svc.sh install
sudo ./svc.sh start
```

4.2.4 Step 4: Install GitHub CLI

The GitHub CLI enables interaction with GitHub from the command line. Install it using the following commands:

```
sudo mkdir -p /etc/apt/keyrings
curl -fsSL https://cli.github.com/packages/githubcli-archive-keyring.gpg | \
    sudo tee /etc/apt/keyrings/githubcli-archive-keyring.gpg \
    > /dev/null
echo "deb [arch=$(dpkg --print-architecture) \
    signed-by=/etc/apt/keyrings/githubcli-archive-keyring.gpg] \
    https://cli.github.com/packages stable main" | \
    sudo tee /etc/apt/sources.list.d/github-cli.list \
    > /dev/null
sudo apt update
sudo apt install gh -y
```

Authenticate with your GitHub account:

```
gh auth login
```

Follow the interactive prompts to complete the authentication process.

4.2.5 Step 5: Clone Repository

Clone the ShopHub repository to your local environment:

```
git clone https://github.com/detiuaveiro/group-project-es2526_203.git
cd group-project-es2526_203
```

4.2.6 Step 6: Configure Environment

Run the automated setup script to configure your environment variables:

```
./scripts/setup-env.sh
```

The setup script performs the following actions:

- Automatically detects your server's IP address
- Creates `.env` file from `.env.example` template
- Configures all IP-related settings including CORS origins and Keycloak redirect URIs

When prompted for the IP address, you can press Enter to use the automatically detected value or enter a custom IP address if needed.

After configuration is complete, commit the environment changes to the repository:

```
git add .
git commit -m "Configure environment variables"
git push
```

4.2.7 Step 7: Monitor the Deployment

Monitor the deployment progress in real-time:

```
gh run watch
```

The deployment process will build and deploy all services including the frontend, backend, databases, Keycloak, Flagsmith, and Ollama.

4.3 Verification

Once the deployment completes successfully, access the ShopHub application by navigating to:

`http://YOUR_SERVER_IP`

You should see the ShopHub homepage, indicating that the installation and deployment were successful.

Chapter 5

Production Configuration

Production host: Ubuntu Server 24.04 LTS at `131.163.96.54`, Docker Engine 29.x, Terraform 1.14.x, and a self-hosted GitHub Actions runner labeled `es-vm-ecommerce`.

5.1 Environment Variables

Workflows load `.env` at the repository root before building, deploying, or verifying. Key variables:

Variable	Purpose
`DEPLOY_HOST_IP`	Public IP used in generated URLs and Terraform configuration (e.g., `131.163.96.54`).
`POSTGRES_PASSWORD`	Password for the application database user (`shophub_user`).
`CORS_ALLOWED_ORIGINS`	Comma-separated origins allowed by the backend.
`BACKEND_PORT_1`, `BACKEND_PORT_2`	Ports for the two backend replicas (default 5000/5001).
`FRONTEND_PORT_1`, `FRONTEND_PORT_2`	Ports for the two frontend replicas (default 3000/3001).
`FLAGSMITH_ENABLED`	Enables/disables Flagsmith integration in the backend.
`FLAGSMITH_ENVIRONMENT_KEY`	Server-side Flagsmith API key.
`REACT_APP_FLAGSMITH_ENVIRONMENT_ID`	Frontend environment ID used by the React SDK.
`REACT_APP_FLAGSMITH_API_URL_PATH`	API path for Flagsmith (default `/api/v1`).
`REACT_APP_FLAGSMITH_DEFAULT_IDENTITY`	Optional default identity for anonymous users.
`REACT_APP_FLAGSMITH_ENABLE_ANALYTICS`	Enables frontend analytics in Flagsmith.
`REACT_APP_KEYCLOAK_REALM`	Realm name used by the React app (`ShopHub`).
`REACT_APP_KEYCLOAK_CLIENT_ID`	Client ID for the SPA (`shophub-frontend`).
`KEYCLOAK_ADMIN`, `KEYCLOAK_ADMIN_PASSWORD`	Credentials used when bootstrapping Keycloak.
`OLLAMA_MODEL`, `OLLAMA_PORT`	Model and port for the Ollama service (defaults `gemma3:1b` on 11434).

5.2 CI/CD Pipeline

The orchestrating workflow is `pipeline.yml`:

- **infrastructure** job runs the infrastructure workflow.
- **build** job builds the Spring Boot JAR and React app, then builds images.
- **test** job runs after image build: backend Maven `clean verify` inside a container with Allure report generation and artifact upload; frontend tests run with Node 18.
- **deploy** job pulls those images, renders `terraform.tfvars` from `.env`, cleans old app containers, and applies Terraform to recreate database, backend, frontend, and NGINX load balancer on `shophub-network`.
- **verify** job performs smoke tests against the load balancer and direct replica ports and reports the deployed images and URLs.

5.3 Monitoring and Observability

- **Health checks** are enforced in the pipeline and runtime: backend Actuator `/actuator/health`, frontend HTTP 200, Keycloak `/health/ready`, Flagsmith `/health`, and NGINX `/health`.
- **Logs** via `docker logs -f` on all containers (backend, frontend, Keycloak, Flagsmith, Ollama, NGINX).

5.4 Health Checks and Operations

- Backend replicas: `:5000/actuator/health` and `:5001/actuator/health`.
- Frontend replicas: `:3000` and `:3001`.
- Load Balancer: `/health`.

5.5 Production URLs

Service	URL
Frontend (via NGINX LB)	`http://131.163.96.54`
Backend API (via NGINX LB)	`http://131.163.96.54/api`
Keycloak Admin	`http://131.163.96.54:8080`
Flagsmith Admin	`http://131.163.96.54:8000`
Ollama API	`http://131.163.96.54:11434`

Chapter 6

Conclusion

ShopHub demonstrates the practical application of software engineering principles to build a production-grade e-commerce platform. The project successfully integrates enterprise technologies including containerization with Docker, infrastructure-as-code with Terraform, OAuth2 authentication via Keycloak, feature flag management with Flagsmith, and AI-powered assistance using Ollama with RAG. The implementation follows a traditional three-tier architecture, automated CI/CD pipelines, and comprehensive observability through OpenTelemetry, Prometheus, and Grafana.

The development process revealed the complexity of real-world systems beyond simple feature implementation. Challenges included coordinating multi-container deployments, managing database migrations without downtime, implementing zero-downtime deployments, debugging distributed tracing issues, and integrating LLM capabilities with product data. The Agile methodology with clearly defined roles enabled the team to navigate these challenges through iterative delivery and continuous feedback.

This documentation provides both a technical reference for the implemented architecture and a comprehensive onboarding guide for developers joining the project, capturing its complete system design, deployment procedures, and operational practices.

Appendix A

Local Deployment with SSH Tunnels

If you wish to run the project locally with tunnels, follow the commands below.

Important Notes:

1. The IP address `131.163.97.55` used throughout these commands must be replaced with the correct IP address for your deployment.
2. Use the correct GitHub account credentials.
3. For detailed installation instructions, refer to: https://github.com/luisbfsousa/Deploy_Local/blob/main/INSTALLATION_GUIDE_LOCALHOST.md

```
# System Update and Base Package Installation
sudo apt update
sudo apt upgrade -y
sudo apt install -y \
    curl \
    wget \
    git \
    vim \
    htop \
    net-tools \
    ca-certificates \
    gnupg \
    lsb-release

# Firewall Configuration
sudo apt install -y ufw &&
sudo ufw allow 22/tcp &&
sudo ufw allow 80/tcp &&
sudo ufw allow 443/tcp &&
sudo ufw allow 3000:3001/tcp &&
sudo ufw allow 5000:5001/tcp &&
sudo ufw allow 8080/tcp &&
sudo ufw allow 8000/tcp &&
sudo ufw enable &&
sudo ufw status verbose

# Docker Installation
```

```
sudo mkdir -p /etc/apt/keyrings

curl -fsSL https://download.docker.com/linux/ubuntu/gpg | \
    sudo gpg --dearmor -o /etc/apt/keyrings/docker.gpg

echo \
  "deb [arch=$(dpkg --print-architecture) \
  signed-by=/etc/apt/keyrings/docker.gpg] \
  https://download.docker.com/linux/ubuntu \
  $(lsb_release -cs) stable" | \
  sudo tee /etc/apt/sources.list.d/docker.list > /dev/null

sudo apt update
sudo apt install -y docker-ce docker-ce-cli containerd.io \
    docker-buildx-plugin docker-compose-plugin

docker --version
sudo usermod -aG docker $USER
newgrp docker
docker run hello-world
docker compose version

# Docker Compose Installation
sudo curl -L "https://github.com/docker/compose/releases/latest/\
download/docker-compose-$(uname -s)-$(uname -m)" \
    -o /usr/local/bin/docker-compose

sudo chmod +x /usr/local/bin/docker-compose
docker-compose --version

# Git Configuration
git --version
git config --global user.name "luisbfsousa"
git config --global user.email "luisbfsousa@ua.pt"

# Java Development Kit Installation
sudo apt install -y openjdk-17-jdk
java --version
echo 'export JAVA_HOME=/usr/lib/jvm/java-17-openjdk-amd64' \
    >> ~/.bashrc
echo 'export PATH=$PATH:$JAVA_HOME/bin' >> ~/.bashrc
source ~/.bashrc

# Node.js Installation
curl -fsSL https://deb.nodesource.com/setup_18.x | sudo -E bash -
sudo apt install -y nodejs
node --version
npm --version

# Terraform Installation
wget -O- https://apt.releases.hashicorp.com/gpg | \
```

```
    sudo gpg --dearmor -o \
    /usr/share/keyrings/hashicorp-archive-keyring.gpg

echo "deb [signed-by=/usr/share/keyrings/hashicorp-archive-keyring.gpg] \
https://apt.releases.hashicorp.com $(lsb_release -cs) main" | \
    sudo tee /etc/apt/sources.list.d/hashicorp.list

sudo apt update
sudo apt install -y terraform
terraform --version

# SSH Key Generation and Repository Clone
ssh-keygen -t ed25519 -C "luisbfsousa@ua.pt"
nano .ssh/id_ed25519.pub
ssh-add ~/.ssh/id_ed25519  # optional

git clone git@github.com:luisbfsousa/Deploy_Local.git
cd Deploy_Local
ls -la

# Production Environment Configuration
cat > frontend/.env.production << EOF
REACT_APP_KEYCLOAK_URL=http://131.163.97.55:8080
REACT_APP_KEYCLOAK_REALM=ShopHub
REACT_APP_KEYCLOAK_CLIENT_ID=shophub-frontend
REACT_APP_API_URL=http://131.163.97.55/api
REACT_APP_FLAGSMITH_ENVIRONMENT_ID=dwuYC8a6HTirU2SQNkz7rX
REACT_APP_FLAGSMITH_API_URL=http://131.163.97.55:8000/api/v1
REACT_APP_FLAGSMITH_DEFAULT_IDENTITY=
REACT_APP_FLAGSMITH_ENABLE_ANALYTICS=false
EOF

# Development Environment Configuration
cat > frontend/.env.development << EOF
REACT_APP_KEYCLOAK_URL=http://localhost:8080
REACT_APP_KEYCLOAK_REALM=ShopHub
REACT_APP_KEYCLOAK_CLIENT_ID=shophub-frontend
REACT_APP_API_URL=/api
REACT_APP_FLAGSMITH_ENVIRONMENT_ID=CqLZfVKsZdP8MyvESjpvnv
REACT_APP_FLAGSMITH_API_URL=http://localhost:8000/api/v1
REACT_APP_FLAGSMITH_DEFAULT_IDENTITY=
REACT_APP_FLAGSMITH_ENABLE_ANALYTICS=false
EOF

# Start Core Services
docker compose up -d postgres keycloak keycloak-postgres \
    flagsmith flagsmith-postgres flagsmith-task-processor ollama

docker compose ps

# Configure Ollama
```

```
docker exec shophub-ollama ollama pull gemma
docker exec shophub-ollama ollama list
docker exec shophub-ollama ollama run gemma "teste"

# Backend Build
cd backend
sudo apt install -y maven
mvn clean package -DskipTests
ls -lh target/shophub-backend-1.0.0.jar

# Frontend Build
cd ../frontend
npm install
npm run build
ls -lh build/
cd ..

# Docker Image Creation
docker build -t shophub-backend:latest \
    -f backend/Dockerfile backend/

docker build -t shophub-frontend:latest \
    -f frontend/Dockerfile.prod frontend/

docker images | grep shophub

# Start Application Services
docker compose up -d postgres keycloak keycloak-postgres \
    flagsmith flagsmith-postgres flagsmith-task-processor ollama

docker compose ps

docker compose up -d backend frontend

docker compose ps

# SSH Tunnel Configuration
ssh -L 3000:localhost:3000 \
    -L 5000:localhost:5000 \
    -L 5001:localhost:5001 \
    -L 8000:localhost:8000 \
    -L 8080:localhost:8080 \
    -L 9090:localhost:9090 \
    -L 11434:localhost:11434 \
    scrummaster@131.163.97.55
```