Add icon   Add cover

Smart E-Commerce System

Spring Security

Complexity: Advanced | Time Estimate: 10-12 hours

Project Overview

This phase emphasizes **building secure, scalable, and production-ready web backends** that can safely handle authentication, authorization, and cross-origin requests. Learners will apply **Spring Security** concepts to secure their application's REST and GraphQL APIs using **JWT-based authentication**, **OAuth2 login (Google)**, and **Role-Based Access Control (RBAC)**. They will also explore **security-related DSA concepts** such as hashing, encryption, and secure token validation.

Project Objectives

By the end of this project, learners will be able to:

1. **Apply** Spring Security configurations to enforce authentication, authorization, and access control across REST and GraphQL APIs.
2. **Implement** JWT authentication, Google OAuth2 login, and secure password hashing using BCrypt.
3. **Configure** and **analyze** CORS and CSRF policies for different client interactions (Postman, browsers, and JavaFX).
4. **Apply** DSA concepts (hashing, encryption, and token validation) to strengthen data security and session integrity.
5. **Develop** and **test** role-based access control (RBAC) and secure endpoint communication for real-world deployment.

Epics and User Stories

Epic 1: Security Configuration and Access Policies

User Story 1.1

As a developer, I want to configure Spring Security filters so that I can protect all endpoints and define which resources require authentication.

Acceptance Criteria

- SecurityFilterChain configured with custom access rules.
- Public (login, registration) and restricted (admin, customer) endpoints defined.
- Passwords stored using `BCryptPasswordEncoder`.

User Story 1.2

As a frontend developer, I want to enable secure cross-origin requests so that external clients can interact with the backend.

Acceptance Criteria

- Global CORS configuration implemented to allow specific origins, methods, and headers.
- Configuration tested with both Postman and a web frontend.
- Unauthorized origins correctly rejected.

Epic 2: JWT-Based Authentication

User Story 2.1

As a user, I want to log in with valid credentials and receive a JWT token so that I can access restricted endpoints securely.

Acceptance Criteria

- `/auth/login` endpoint generates signed JWTs with claims (username, roles, expiry).
- Tokens validated on each protected request.
- Tampered or expired tokens rejected with 401 Unauthorized.

User Story 2.2

As a system analyst, I want to verify token structure and claims so that I can validate security implementation.

Acceptance Criteria

- JWT includes subject, issued time, and expiration claims.
- HMAC SHA-256 or RSA algorithm used for signature verification.
- Token payload can be viewed in Postman (decoded) for verification.

Epic 3: CSRF and Session Security

User Story 3.1

As a security engineer, I want to configure CSRF protection properly so that the system is secure against cross-site request forgery.

Acceptance Criteria

- CSRF protection disabled for stateless JWT APIs.
- Explanation provided for when CSRF should be enabled (stateful sessions, forms).
- CSRF token mechanism demonstrated in one form endpoint for illustration.

User Story 3.2

As a developer, I want to understand and document CSRF and CORS differences so that I can configure them correctly for frontend and backend communication.

Acceptance Criteria

- Technical documentation describing CORS and CSRF interaction included in README.
- Practical demonstration using Postman and browser client.

Epic 4: OAuth2 and Role-Based Access Control (RBAC)

User Story 4.1

As a user, I want to log in with my Google account so that I can access the system without manual signup.

Acceptance Criteria

- OAuth2 login integrated using Google provider.
- User details fetched from Google API and persisted.
- Roles assigned after successful OAuth2 authentication.

User Story 4.2

As an administrator, I want to restrict access to certain endpoints based on user roles so that I can enforce role-based permissions.

Acceptance Criteria

- Roles defined (ADMIN, CUSTOMER, STAFF).
- Endpoints annotated with `@PreAuthorize` or `@Secured`.
- Role-based access verified using Postman test cases.

Epic 5: DSA and Security Optimization

User Story 5.1

As a developer, I want to apply data structure and algorithm principles to improve security and performance.

Acceptance Criteria

- Hashing used for password security and token validation.
- Caching or lookup map implemented for temporary token storage or blacklisting.
- Optional use of in-memory map (hash map) for revoked tokens.

User Story 5.2

As an auditor, I want to view and analyze security event logs so that I can track login attempts and access patterns.

Acceptance Criteria

- Logging implemented for authentication success/failure events.
- Security reports include token usage and endpoint access frequency.
- Logs reviewed to detect unusual access or brute-force attempts.

Technical Requirements

	Area	Description
1	**Framework**	Spring Boot 3.x (Spring Security, JWT, OAuth2 Client, Validation, Cache)
2	**Language**	Java 21
3	**Authentication**	Username/password with JWT, OAuth2 login (Google)
4	**Authorization**	Role-Based Access Control (RBAC)
5	**Security Features**	CORS setup, CSRF demonstration, password hashing
6	**Password Encryption**	BCryptPasswordEncoder
7	**Database**	Existing database extended with user and role entities
8	**Testing & Interaction**	Postman or any web frontend for login and authorization testing
9	**Documentation**	OpenAPI documentation covering secured and public endpoints
10	**DSA Integration**	Hashing, token validation, map-based blacklisting, and secure lookup design

Deliverables

	Deliverable	Description
1	**Spring Security Integration**	Application configured with authentication and authorization filters.
2	**JWT Authentication System**	Fully functional login, token generation, and validation endpoints.
3	**CORS & CSRF Configuration**	Working setup with documentation and demonstrations for safe client access.
4	**OAuth2 (Google Login)**	Social login configured with user persistence and mapped roles.
5	**RBAC Enforcement**	Role-based access control implemented on key endpoints.
6	**Security Event Logging**	Logs capturing authentication and access details.

7	**DSA Implementation**	Applied hashing, caching, and lookup optimization within security flow.
8	**README & OpenAPI Docs**	Comprehensive documentation covering setup, testing, and configuration notes.

Evaluation Criteria

	Category	Description	Points
1	**Security Configuration (CORS & CSRF)**	Correct configuration and explanation of CORS, CSRF, and their use cases.	15
2	**JWT Implementation**	Functional JWT authentication with secure token generation and validation.	20
3	**OAuth2 (Google Integration)**	External login configured with consistent user mapping and persistence.	15
4	**RBAC and Role Enforcement**	Role-based restrictions implemented accurately across endpoints.	15
5	**DSA in Security**	Application of hashing, lookup, or caching for security logic.	15
6	**Testing & Logging**	Postman tests, security event logs, and error tracking implemented.	10
7	**Code Quality & Documentation**	Organized configuration, clear comments, and complete API documentation.	10
8	**Total**		**100 pts**