DESOFS Project: Perfume Webstore Threat Model

NAME:

Pedro Nogueira – 1211613

Alexandre Geração – 1211151

EnMiao Ye – 1211905

Eduardo Fernandes – 1240449

Filip Frýdl - 1242314

DATE:

Contents

Tables

Introduction

Write Introduction Here

Scope Your Work (Samples)

Thread Modelling Information

Application Name: Perfume Webstore

Description:
The **Perfume Webstore** is a full-featured e-commerce application designed to provide users with a seamless online shopping experience for purchasing premium fragrances. This application serves as the initial production-ready deployment, featuring core functionalities for both customers and administrators.

The backend is built using **Spring Boot**, leveraging **Spring Security**, **JWT**, and **OAuth2** for secure authentication, while the frontend is developed with **React.js** and **Redux Toolkit** for a responsive user interface. The frontend is deployed to **AWS S3** with public access settings, and the backend is hosted on **Heroku**.

Primary User Roles:

- **Customers**:
    - Register and log in using credentials or third-party OAuth2 providers (Google, Facebook, GitHub)

  - Search and browse products using filters
  - View detailed product information
  - Add or remove products from the shopping cart
  - Place orders and view order history
  - Update account details and passwords
- **Administrators**:
  - Access admin panel via login
  - Create, update, and delete product entries
  - Manage user information (CRUD operations)
  - View and manage all customer orders

Security Features:

- **JWT-based authentication** for secure, stateless session management
- **OAuth2 login integration** with Google, Facebook, and GitHub
- **reCAPTCHA** to prevent automated abuse and bot registration
- **Secure credential storage** (password encryption using hashing algorithms)
- Communication via HTTPS for data transmission security

Deployment Overview:

- Backend: Java 8, Spring Boot, PostgreSQL, deployed on **Heroku**
- Frontend: React.js, TypeScript, Redux Toolkit, Ant Design, deployed to **AWS S3**
- Data persistence with **PostgreSQL**
- Build tools: **Maven**, **Webpack**, **npm/yarn**

Threat Modelling Context: Given the application's deployment to cloud services and handling of sensitive user and payment-related data, it is essential to analyses possible attack surfaces, trust boundaries, and data flows.

Participants: Pedro Nogueira, Enmiao Ye, Eduardo Fernandes, Alexandre Geração, Filip Frýdl

Admin Controller

The AdminController is a RESTful and GraphQL-enabled controller responsible for managing core administrative operations within the Perfume Webstore. It grants high-privilege access required for maintaining the application's data integrity and configuration. It allows administrators to:

- Create, update, and delete perfume entries, including managing product images (`POST /add`, `POST /edit`, `DELETE /delete/{perfumeId}`).
- View all customer orders and orders for specific users (`GET /orders`, `GET /order/{userEmail}`).
- Delete specific orders (`DELETE /order/delete/{orderId}`).
- Retrieve details for all users or specific users by ID (`GET /user/all`, `GET /user/{userId}`).
- Execute administrative GraphQL queries for users and orders (`POST /graphql/**`).

These endpoints provide comprehensive control over critical application data, including products, user accounts, and order history. Due to the elevated privileges and access to sensitive information, securing this controller is paramount. All endpoints under `/api/v1/admin` are protected using Spring Security's `@PreAuthorize("hasAuthority('ADMIN')")` annotation, ensuring only authenticated users with the ADMIN role can access these functionalities, enforced via JWT authentication managed by the `JwtFilter`.

Authentication Controller

The AuthenticationController is a RESTful controller responsible for managing operations related to authentication and password recovery. It allows users to:

- Log in with valid credentials.
- Send requests for password reset via email.
- Retrieve the email associated with a password reset code.
- Reset the password using a reset code.
- Update the password while authenticated as registered users.

These endpoints **handle sensitive data**, such as **credentials** and **account recovery information**, making them critical for security and a potential target for brute force or phishing attacks.

Order Controller

The OrderController is a RESTful controller responsible for managing all operations related to customer orders. It allows clients to:

- Retrieve a specific order by its ID.
- Fetch all line-items for a given order.
- List the authenticated user's orders with pagination support.
- Create a new order from a submitted OrderRequest.
- Execute arbitrary order-related GraphQL queries or mutations.

These endpoints drive core e-commerce functionality—order lookup, creation, and flexible querying—making them key to both user experience and back-office processing.

Endpoints under /api/v1/order are protected via JWT authentication enforced by the JwtFilter, with user context provided by UserPrincipal for listing orders.

Perfume Controller

The PerfumeController is a RESTful and GraphQL-enabled controller that handles all public-facing read operations related to perfumes. It allows users to:

- Fetch all perfumes
- Fetch perfumes by ID
- Search perfumes by text, gender, or perfumer
- Execute GraphQL queries

These endpoints are **read-only** and **do not modify server state**, but they **expose public product data**, making them a target for enumeration, scraping, or performance-based attacks.

User Controller

The UserController is a RESTful and GraphQL-enabled controller that handles all user management operations. It allows:

- Fetch/update user profiles
- Retrieve cart items
- Execute GraphQL queries for user data

These endpoints are for viewing and updating user profiles, managing shopping cart contents, and running GraphQL queries about users. All endpoints require user authentication since they work with sensitive account data like personal information and purchase history. The controller performs both read and write operations that affect user data stored in the system.

Review Controller

The Review Controller is a core component of the eCommerce application. It is responsible for managing user reviews related to perfumes. The controller provides endpoints to retrieve reviews for a specific perfume and to add new reviews. Key functionalities include:

- Retrieving Reviews
- Adding Reviews

The controller interacts with the ReviewMapper to handle business logic and database interactions and utilizes constants for API routing. It is built with Spring Boot, enabling RESTful communication. Security considerations for this controller include ensuring data validation, protecting endpoints against unauthorized access, and securing WebSocket communication to prevent potential vulnerabilities.

External Dependencies

Table 1 - External Dependencies

ID	**Description**
ED-01	The application is expected to run on a Linux-based production server, possibly cloud-hosted (e.g., AWS EC2). This environment must be **hardened to the organization's security baseline** and assumed to be **behind a firewall and reverse proxy**.
ED-02	Application connects to a **PostgreSQL** database instance (possibly hosted externally or in a managed cloud service like AWS RDS). Proper credentials and network configurations are required.
ED-03	Uses **AWS Java SDK** to interact with AWS services such as S3 or SES. These services are managed outside of the application code and rely on IAM credentials, regions, and network policies.
ED-04	Sends transactional emails via **SMTP** servers using Spring Boot Mail. Assumes an external email provider (e.g., Gmail SMTP, SendGrid, Mailgun), which requires credential handling and may expose metadata to third-party providers.
ED-05	Integrates with **OAuth2 login providers** (e.g., Google, GitHub) via Spring Security OAuth2 Client. Identity verification is delegated to external providers. Redirect URIs and token exchanges must be securely handled.
ED-06	Assumes the deployment platform supports **Java 1.8** and compatible Java Virtual Machine (JVM). This must be available on the server or container used in production.
ED-07	Authentication provider will rely on Spring Security with UserPrincipal for authentication. JWT tokens will be signed with strong HS256/RS256 algorithms. Session will timeout after 30 minutes of inactivity.
ED-08	...

Entry Points

General Project Entry Points

ID	**Name**	**Description**	**Trust Level**
EP-01	HTTPS Port	All interactions occur over HTTPS, serving as the main gateway for all users.	(1) Anonymous, (2) Authenticated, (3) Admin

Authentication-Related Entry Points

Table 2 - General Project Entry Points

ID	Name	Description	Trust Level
EP-01	Login	Authenticates the user by validating the provided credentials. Returns a token or authentication response upon successful login.	(1) Anonymous
EP-02	Forgot Email	Sends a password reset code to the provided email address.	(1) Anonymous
EP-03	Reset Code	Retrieves the email address associated with a given password reset code.	(1) Anonymous
EP-04	Reset	Resets the user's password using the provided email and reset code.	(1) Anonymous
EP-05	Edit Password	Updates the password of the currently authenticated user.	(2) Authenticated, (3) Admin

Order-Related Entry Points

Table 3 - Order Entry Points

ID	Name	Description	Trust Level
EP-01	Retrieve Order	Retrieves the details of a specific order by its ID.	(2) Authenticated
EP-02	Fetch Order Items	Retrieves all line-items associated with a given order by its ID.	(2) Authenticated
EP-03	List User Orders	Lists the authenticated user's orders, with optional pagination support.	(2) Authenticated
EP-04	Create Order	Creates a new order from the submitted OrderRequest payload.	(2) Authenticated
EP-05	Execute Order GraphQL Query	Executes arbitrary order-related GraphQL queries or mutations via a generic endpoint.	(2) Authenticated

Perfume-Related Entry Points

Table 4 - Perfume Entry Points

ID	Name	Description	Trust Level
EP-01	Add Perfume	Allows users (typically admins) to add a new perfume by providing details such as name, description, price, and an image.	(3) Admin
EP-02	Edit Perfume	Allows users to update details of an existing perfume, such as name, price, description, or image.	(3) Admin
EP-03	Delete Perfume	Allows users to delete a perfume from the system.	(3) Admin
EP-04	Search Perfumes	Allows users to search for perfumes based on parameters like gender, perfumer, or text search.	(1) Anonymous, (2) Authenticated
EP-05	View Perfume Details	Provides the details of a specific perfume, including its description, price, and reviews.	(1) Anonymous, (2) Authenticated
EP-06	Add Review to Perfume	Allows users to submit a review for a specific perfume.	(2) Authenticated, (3) Admin
EP-07	Get Perfume Reviews	Fetches all reviews associated with a specific perfume.	(1) Anonymous, (2) Authenticated

User-Related Entry Points

Table 5 - User Entry Points

ID	Name	Description	Trust Level
EP-01	Get user data	Allows user to fetch their own data, must be authenticated	(2) Authenticated (own data only) (3) Admin
EP-02	Update user data	Allows users to update their own details of account, such as last name, city, address, phone number, or post index.	(2) Authenticated (own data only) (3) Admin
EP-03	Cart operation	Handles shopping cart management for authenticated users.	(2) Authenticated
EP-04	GraphQL User	Provides flexible user data queries through GraphQL requests.	(2) Authenticated (3) Admin

EP-05	Authentication Flow	Manages user login and JWT token generation.	(1) Anonymous, (2) Authenticated

Review-Related Entry Points

Table 6 - Review Entry Points

ID	Name	Description	Trust Level
EPR-01	Review Controller Base Path	The base path for the ReviewController endpoints is defined as /api/v1/review. All subsequent endpoints for managing reviews are layered on this entry point.	(1) Anonymous (2) Authenticated
EPR-02	Get Reviews by Perfume ID	Retrieves a list of reviews associated with a specific perfume, identified by its unique ID.	(1) Anonymous (2) Authenticated
EPR-03	Add Review to Perfume	Retrieves a list of reviews associated with a specific perfume, identified by its unique ID.	(2) Authenticated

Exit Points

In the context of the perfume store web application, **exit points** are areas where data leaves the system and is exposed to the client. These points can be exploited if not properly protected, especially when they involve sensitive information or dynamic content.

Common examples include:

1. **Authentication Responses**
    - When users or admins log in, the system returns error messages or tokens (JWT/session). These responses must be generic and secure to prevent information disclosure (e.g. not revealing whether the username exists).
2. **Search and Detail Views**
    - Search results and perfume details displayed on the frontend must only include public data and prevent leaking sensitive backend information or internal metadata.
3. **Error Handling Messages**
    - Backend exceptions such as SQL or server errors must be properly sanitized before being shown to the user. Raw error traces can be exploited for SQL injection, XSS, or reconnaissance.

4. **User Reviews and Comments**
    - User-generated content like perfume reviews should be sanitized to prevent **Cross-Site Scripting (XSS)** attacks on other users viewing them.
5. **User Profile and Personal Info**
    - If users can view or edit their profile, data like names or emails shown must be authorized and validated. Inadequate checks could leak info or enable **Insecure Direct Object References (IDOR)**.
6. **Session Tokens**
    - Tokens sent in response headers or body must be stored securely on the client and never logged or exposed in URLs.

Assets

Table 7 - Assets

ID	Name	Description	Trust Level
PE-01	**Users and Admins**	**Assets relating to the registered users and admin roles, including login credentials, user data, and roles.**	
PE-01.1	User Login Details	The login credentials for a user, including email and password. Critical for authenticating users into the system.	(3) Admin, (5) Database Server Administrator
PE-01.2	Admin Login Details	The login credentials for an admin, including username and password, which allow access to admin-specific features. (ex. Delete Perfume, Add Perfume)	(3) Admin
PE-01.3	Personal Data	Includes user details such as name, email, and other personal information.	(3) Admin, (5) Database Server Administrator, (6) Administrator
PE-01.4	Cart data	Perfume selections and purchase history	(2) Authenticated (3) Admin, (5) Database Server Administrator, (6) Administrator
PE-02	**System**	**Assets relating to the underlying system.**	

SQL Error Responses	SQL or backend errors that are sent as part of the HTTP response can expose database structure or queries.	(6) Administrator
Session Tokens	Tokens returned upon login or exposed in URL parameters can be intercepted and reused if not securely handled.	(2) Authenticated, (3) Admin
Searching Results	Listings returned in response to search requests. May reveal more data than intended if filtering is weak.	(1) Anonymous, (2) Authenticated
API availability	The user management API must be available 24/7	(5) Database Server Administrator, (6) Administrator
Code execution privileges	Ability to execute Spring Boot application code	(7) App server process

Trust Levels

Table 8 - Trust Levels

ID	Name	Description
1	Anonymous	A user who accesses public API endpoints without authentication. Limited to view-only actions like browsing perfumes or viewing details.
2	Authenticated	A registered and authenticated user. Can place orders, write reviews, and manage personal data.
3	Admin	A privileged user who can manage perfumes, moderate content, and access administrative functions.
4	Database Server Administrator	A system-level role with full access to manage database configurations, backups, and user privileges.
5	Administrator	
6	App Server Process	Spring Boot runtime executing API logic (DB access via limited credentials)

Data Flow Diagrams

General Application Data Flow Diagram

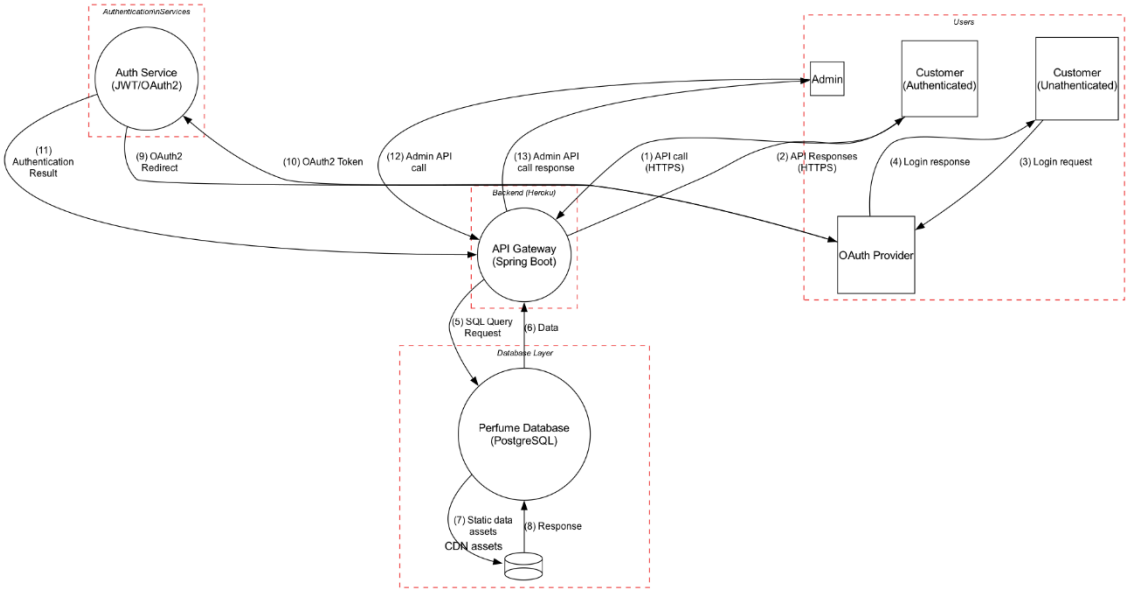

Figure 1 - Application Flow Diagram

Authentication Data Flow Diagram

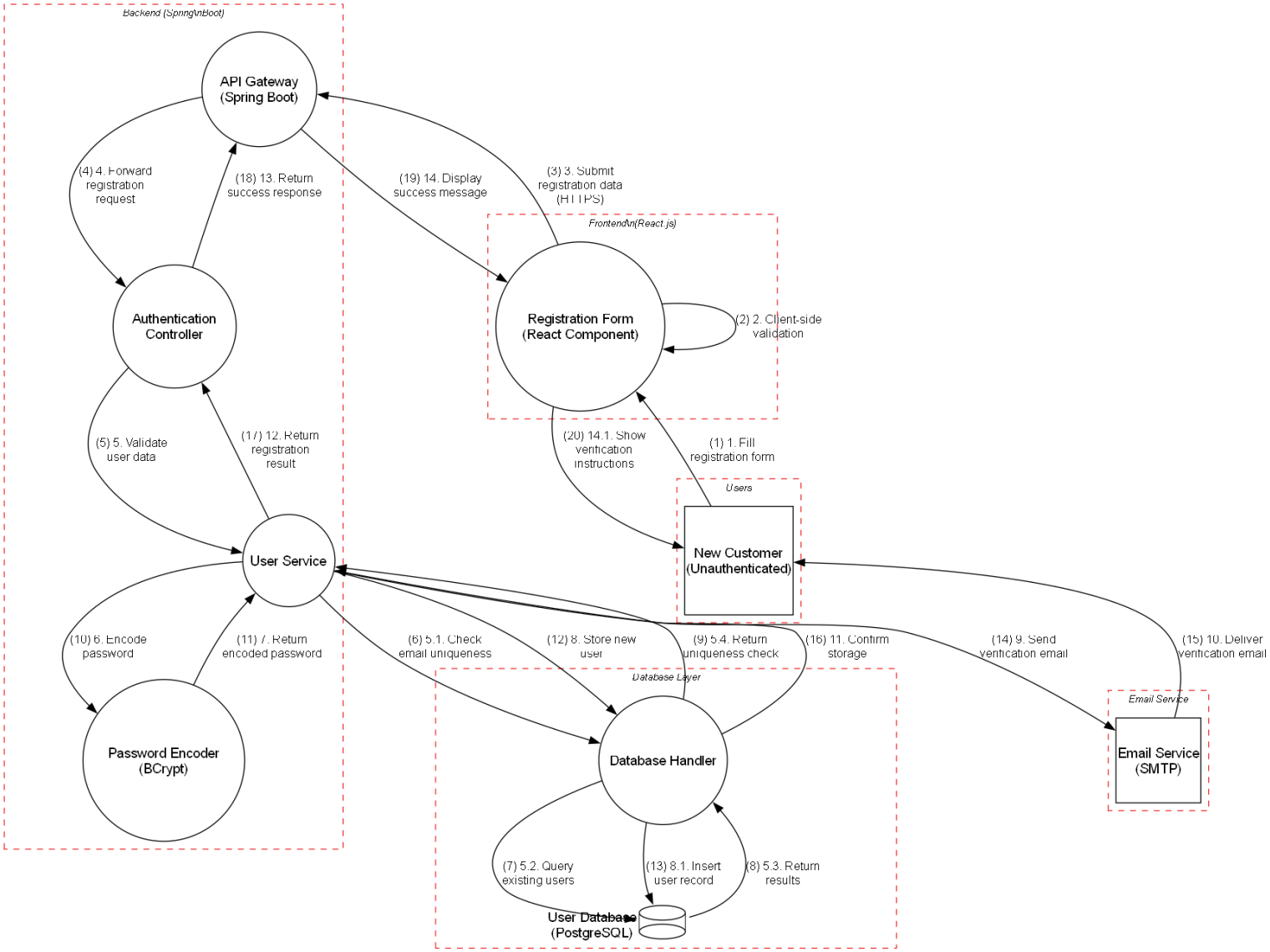

Figure 2 - Create a User

Order Data Flow Diagram

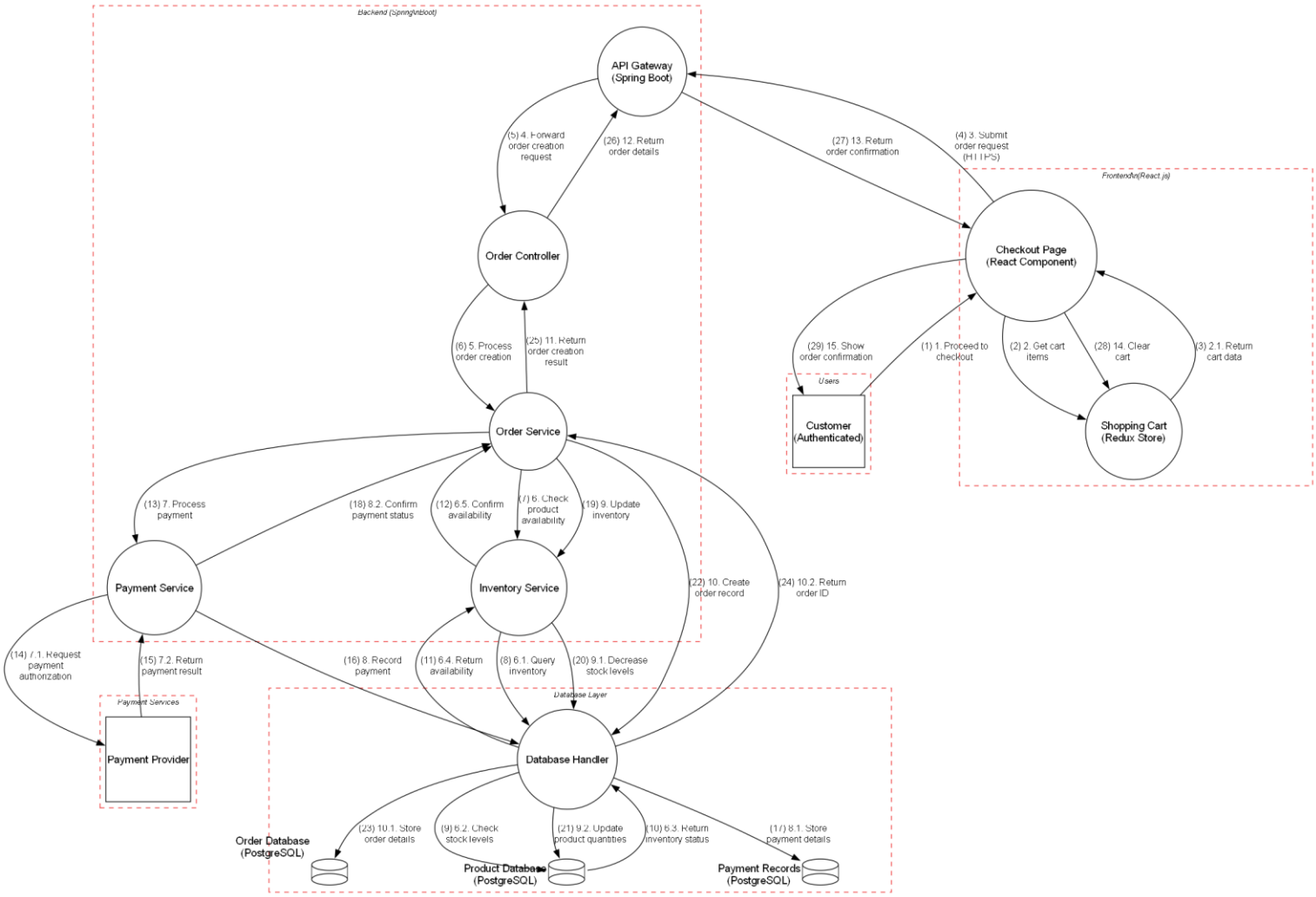

Figure 3 – Create/Update an Order

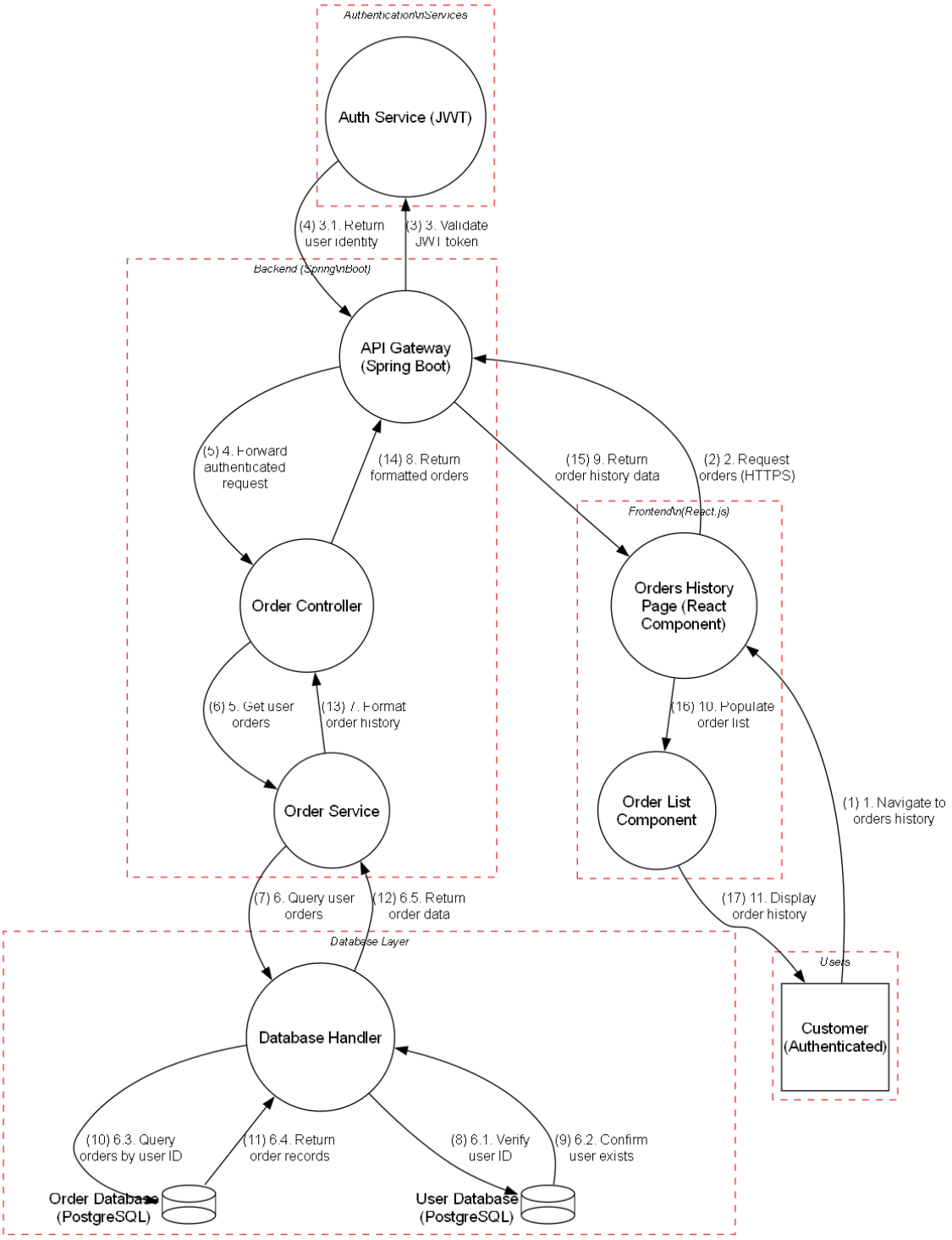

Figure 4 - Get Orders Details/History

Perfume Data Flow Diagram

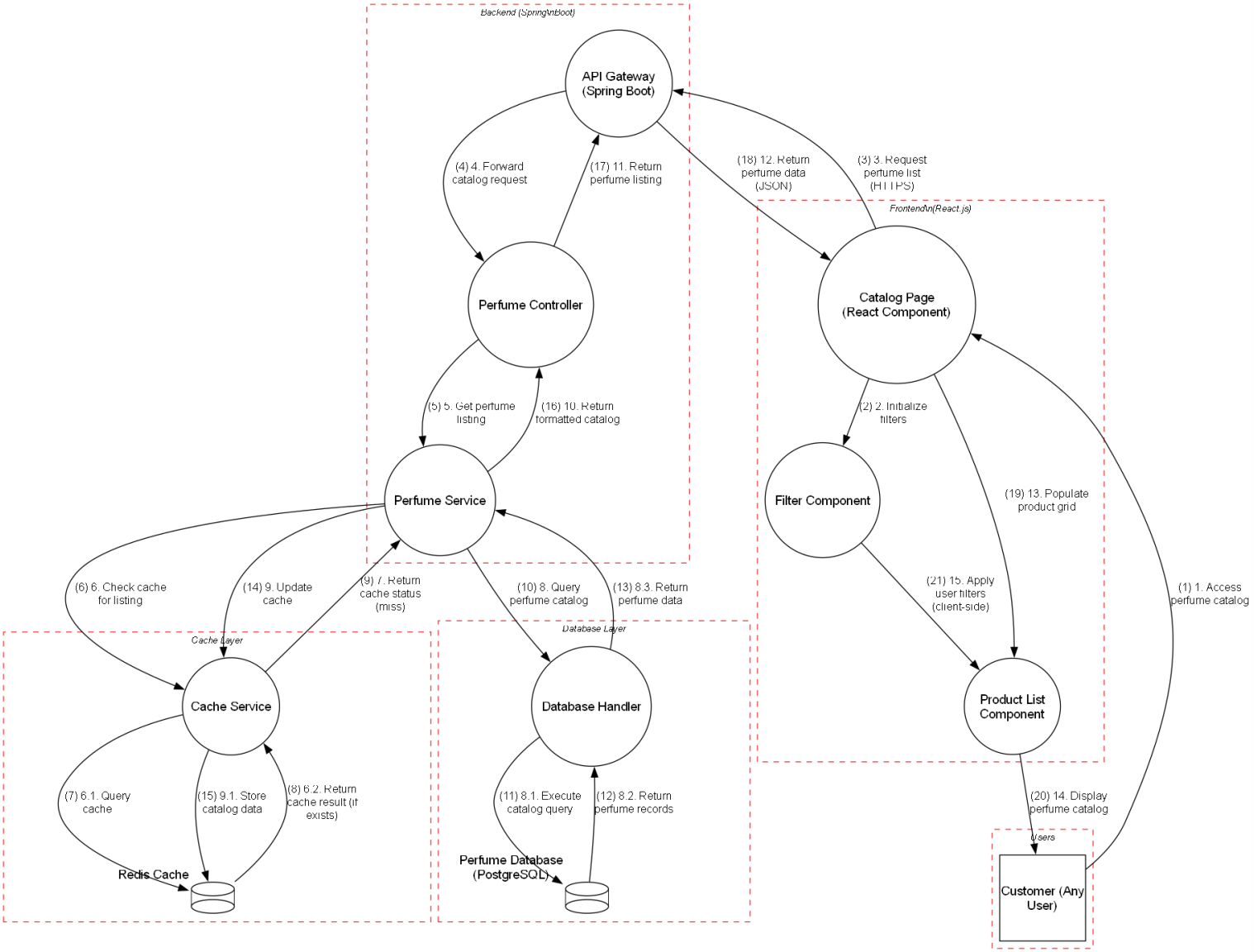

Figure 5 - Search/Get Perfume Details

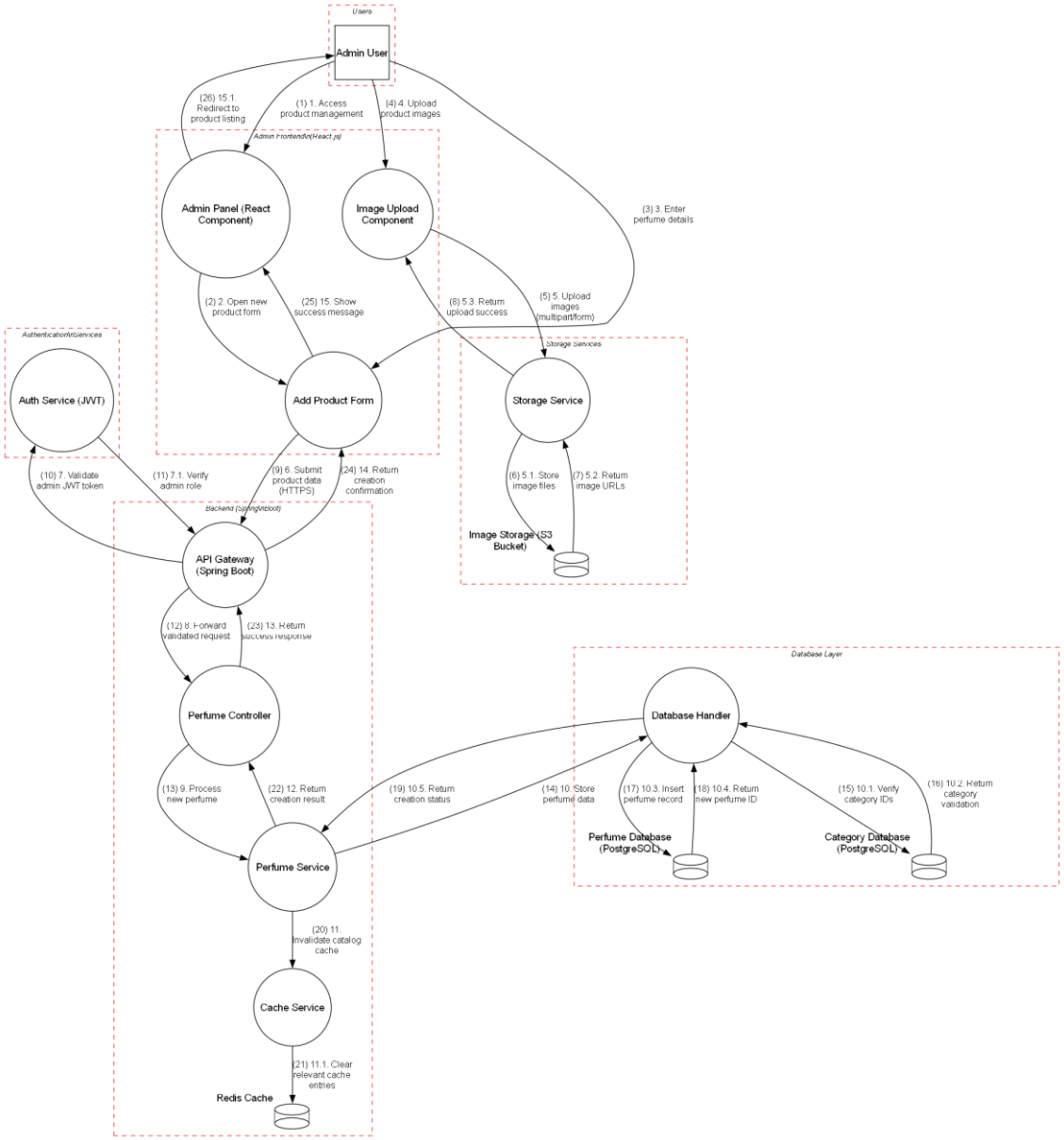

Figure 6 - Add/Update Perfume

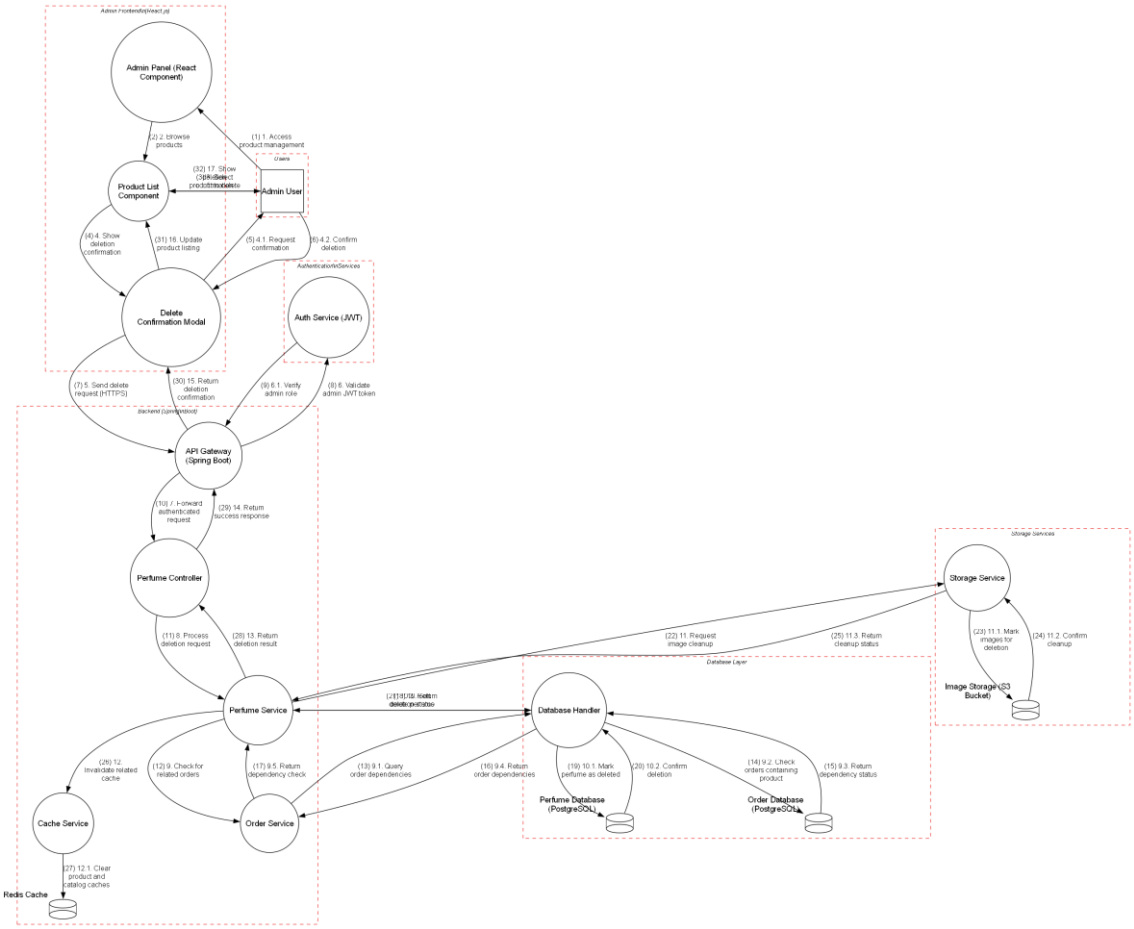

Figure 7 - Delete Perfume

User Data Flow Diagram

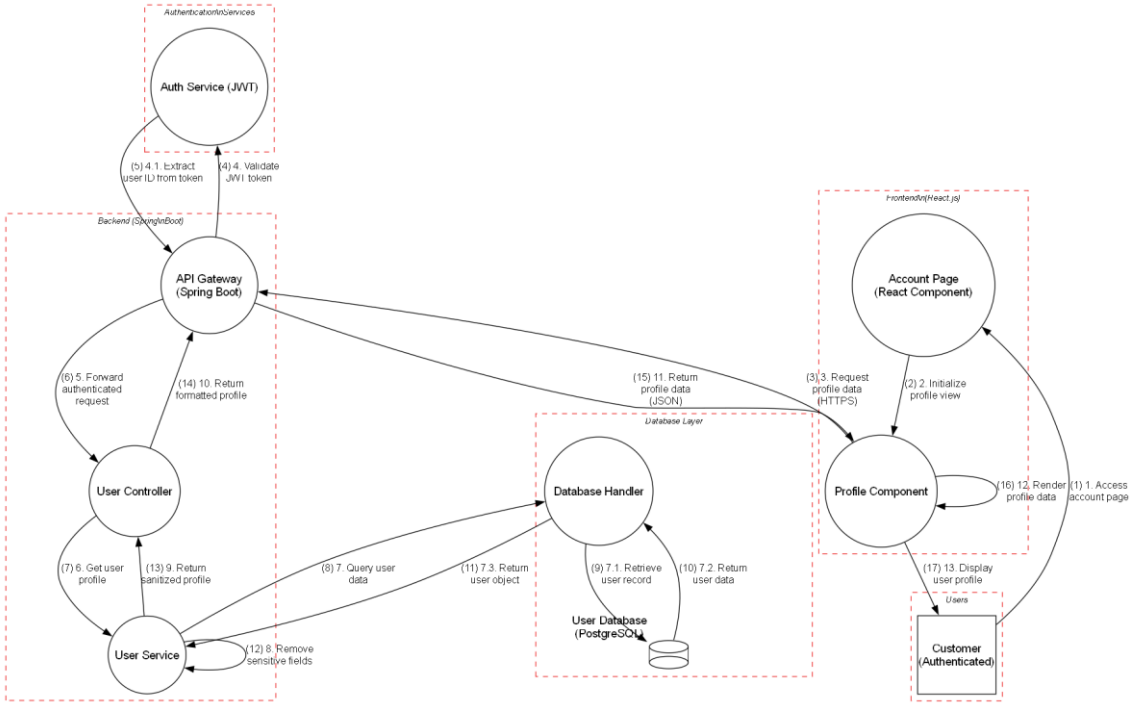

Figure 9 - Get User Profile

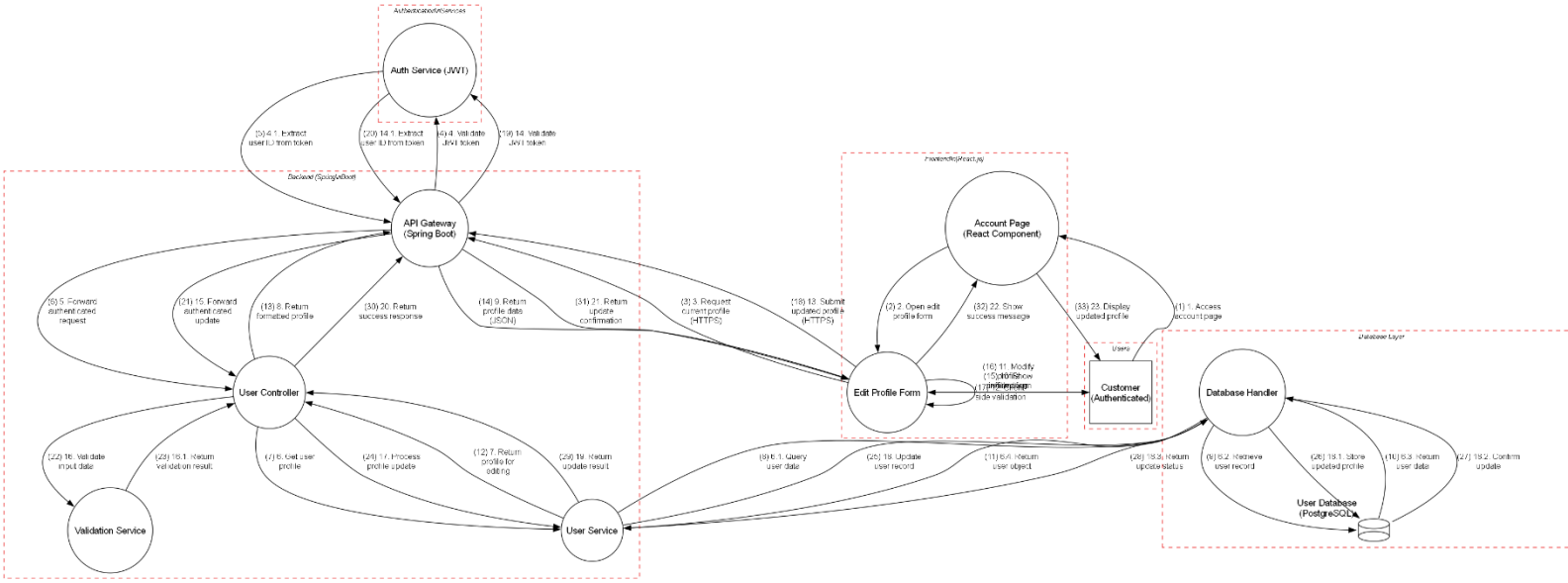

Figure 8 - Update a User Profile

Review Data Flow Diagram

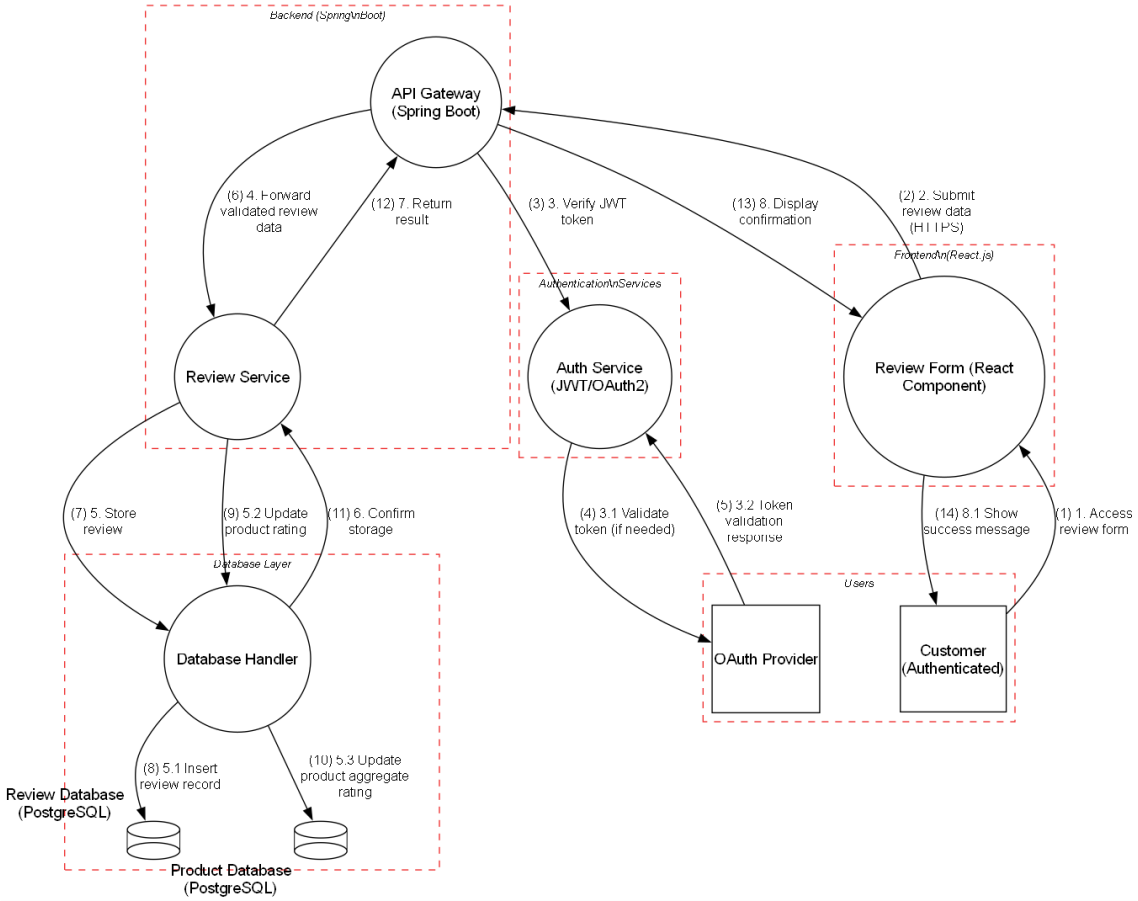

Figure 11 - Create/Update a Review

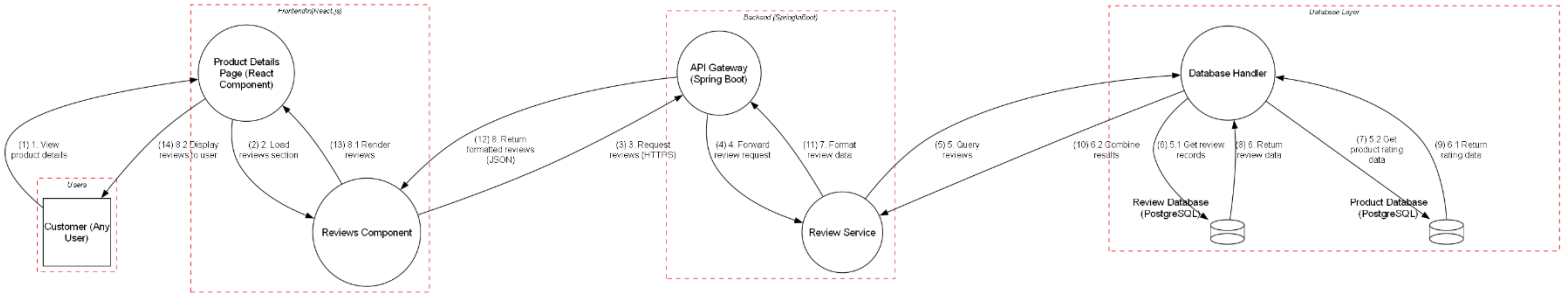

Figure 10 - Get a Review

STRIDE

Threat Model for Admin Controller

The AdminController provides high-privilege access to manage core application data, including perfumes, users, and orders. All endpoints under /api/v1/admin are protected by @PreAuthorize("hasAuthority('ADMIN')"), restricting access to authenticated administrators. The following STRIDE analysis considers threats assuming an attacker has either compromised an admin account or is a malicious insider misusing their privileges.

Threat Identification (STRIDE)/ Security Controls

Table 9 - Admin Controller STRIDE

Type	Endpoint(s)	Threat Description	Security Control
Spoofing	All endpoints	While primary authentication is handled elsewhere, a compromised admin account allows an attacker to spoof the legitimate admin. Lack of multi-factor authentication (MFA) for admins increases this risk.	- Enforce Multi-Factor Authentication (MFA) for admin accounts. - Use strong authentication mechanisms (handled by AuthenticationController/Security Config). - Implement robust session management and token security.
Tampering	POST /add, POST /edit, DELETE /delete/{perfumeId}, DELETE /order/delete/{orderId}, All POST /graphql/** endpoints	An authorized (or compromised) admin can maliciously modify product details (price, description), delete products or orders, or alter user data (via GraphQL if schema permits). Invalid input could potentially cause data corruption if validation is weak.	- Implement robust input validation (@Valid, service-layer checks) for PerfumeRequest on /add and /edit. - Use "soft delete" mechanisms instead of hard deletes where appropriate, especially for orders. - Implement fine-grained validation and sanitization for all GraphQL inputs. - Regularly back up critical data.
Repudiation	POST /add, POST /edit, DELETE /delete/{perfumeId},	A malicious admin performs destructive actions (e.g., deleting key products or orders) and denies	- Implement **detailed and immutable audit logging** for ALL admin actions (CRUD operations on perfumes, orders, users). Log admin

	DELETE /order/delete/{orderId}	responsibility. Insufficient audit logging makes attribution difficult or impossible.	user ID, timestamp, source IP, specific action, target entity ID, and outcome. - Ensure logs are centrally collected and protected from tampering.
Information Disclosure	GET /orders, GET /order/{userEmail}, GET /user/{userId}, GET /user/all, All POST /graphql/** endpoints	An admin (legitimate or compromised) has broad access to sensitive user PII and all order data. Risk of accidental leakage via overly broad GraphQL queries, insecure logging of sensitive data, or intentional data exfiltration by a rogue admin.	- Ensure admin interfaces (if separate) display only necessary data. - Restrict GraphQL schema for admin queries if possible, avoid overly broad queries. - Implement strict data handling policies and training for admins. - Mask sensitive PII in logs unless essential. - Monitor admin activity for anomalies.
Denial of Service	All Endpoints, esp. DELETE and GraphQL endpoints	An attacker with admin access could perform bulk deletions of critical data (perfumes, orders). Flooding admin endpoints or executing resource-intensive GraphQL queries can impact overall application performance and availability for all users.	- Implement rate limiting even for admin endpoints (might be higher limits). - Add confirmation steps or "soft delete" for destructive operations. - Apply GraphQL complexity/depth limits. - Monitor resource usage. - Implement robust backup and recovery procedures.
Elevation of Privilege	N/A	Primarily concerns *gaining* admin access initially (handled by AuthenticationController). If different admin levels existed (e.g., SuperAdmin vs. ContentAdmin), EoP between them would be relevant, but not apparent here. An attacker could exploit vulnerabilities	- Secure the admin account creation/authentication process rigorously. - Keep dependencies updated (patching vulnerabilities). - Adhere to the principle of least privilege if multiple admin roles were introduced.

		in underlying framework/libs.	

Use and Abuse Cases

Table 10 - Admin Controller Use and Abuse Cases

Scenario	Use Case	Abuse Case
Add/Edit Perfume (POST /add, /edit)	Admin adds a new perfume with correct details or updates an existing one (e.g., price change, new image).	- **Tampering:** Admin submits incorrect/misleading product information (e.g., wrong price, malicious description with script tags if not sanitized). - **Input Validation Bypass:** Attacker probes endpoints with malformed PerfumeRequest data.
Delete Perfume (DELETE /delete/{id})	Admin removes a discontinued or incorrectly added perfume from the catalog.	- **DoS/Data Loss:** Malicious admin deletes multiple critical/popular perfumes. - **Repudiation:** Admin deletes a perfume and denies doing it.
View All/User Orders (GET /orders, /order/{email})	Admin reviews recent orders for fulfillment or investigates a specific user's order history.	- **Information Disclosure:** Compromised admin account used to exfiltrate bulk order data or specific user histories. - **DoS:** Flooding the endpoints with requests.
Delete Order (DELETE /order/delete/{id})	Admin removes a test order or a cancelled/problematic order after resolution.	- **DoS/Data Loss:** Malicious admin deletes numerous valid customer orders. - **Repudiation:** Admin deletes an order and denies involvement.
View All/Specific Users (GET /user/all, /user/{id})	Admin looks up user details for support or account management purposes.	**Information Disclosure:** Compromised admin account used to exfiltrate bulk user PII. - **DoS:** Flooding the endpoints with requests.
GraphQL Queries (POST /graphql/)**	Admin uses specific GraphQL queries for efficient data retrieval (e.g., complex reports not covered by REST).	- **DoS:** Attacker submits overly complex or resource-intensive GraphQL queries. - **Information Disclosure:** Attacker crafts queries to access broader data sets

		than intended or potentially sensitive internal fields.

Threat Analysis

The AdminController provides extensive control over the application's data and configuration. While protected by role-based authorization (@PreAuthorize("hasAuthority('ADMIN')")), risks exist, particularly from compromised admin accounts or malicious insiders. The STRIDE analysis reveals the following specific concerns:

- **Spoofing:** The main risk here is not bypassing the initial login (handled elsewhere) but the consequences of a successful admin account takeover. An attacker gaining admin credentials can perform any action as that legitimate admin. Without MFA or robust session monitoring, distinguishing malicious actions from legitimate ones becomes difficult.
- **Tampering:** Given the controller's capabilities, data tampering presents a significant risk. An attacker or malicious admin could modify perfume details (POST /add, POST /edit) to incorrect values (e.g., setting prices to zero, adding malicious descriptions). More critically, the ability to delete perfumes (DELETE /delete/{perfumeId}) and orders (DELETE /order/delete/{orderId}) allows for direct sabotage or destruction of crucial business data. Weak input validation on add/edit endpoints could also lead to data corruption.
- **Repudiation:** The destructive nature of actions like deleting perfumes or orders makes repudiation a serious concern. An admin performing such actions maliciously could deny involvement. Without comprehensive, tamper-evident audit logs clearly attributing each admin action (who, what, when, from where), it's impossible to establish accountability or investigate incidents effectively.
- **Information Disclosure:** Administrators inherently have broad access to sensitive information, including all user PII (via GET /user/all, GET /user/{userId}, GraphQL) and complete order histories (GET /orders, GET /order/{userEmail}, GraphQL). The risk lies in potential data exfiltration by a compromised account or rogue admin. Accidental disclosure could also occur through insecure logging practices (logging PII) or overly broad GraphQL queries returning more data than necessary for a specific admin task.
- **Denial of Service (DoS):** Admin endpoints can be targets for DoS. Flooding any admin API endpoint can consume resources. More targeted DoS involves abusing administrative privileges: mass deletion of products or orders cripples the application's function. Resource-intensive GraphQL queries intended for administrative reports could also negatively impact overall system performance if not optimized and limited.
- **Elevation of Privilege (EoP):** Within the scope of this controller (where users are already admins), traditional EoP is less relevant unless multiple admin tiers exist. However, vulnerabilities in the underlying framework or dependencies used by the admin controller could potentially be exploited by an attacker *already possessing admin access* to gain further control over the server environment (e.g., Remote

Code Execution), which constitutes a severe privilege escalation beyond the application layer. Keeping dependencies patched is crucial.

This analysis underscores the critical need for strong authentication (MFA recommended), rigorous input validation, detailed and secure audit logging, careful data exposure management, and DoS protection, even for administrative interfaces.

Ranking of Threats

Given the high level of privilege associated with the AdminController, the threats are ranked based on the potential severity of impact if an admin account is compromised or misused. The priorities differ slightly from user-facing controllers due to the scope of potential damage:

1. **Tampering (Data Destruction/Modification):**
    - **Justification:** This is ranked highest due to the potential for catastrophic impact. An attacker with admin access could maliciously modify critical product information (e.g., pricing on /add, /edit) or, more severely, perform bulk deletions of perfumes (DELETE /delete/{perfumeId}) or customer orders (DELETE /order/delete/{orderId}), causing irreversible data loss and severe business disruption. The impact on data integrity and availability is potentially extreme.
2. **Repudiation:**
    - **Justification:** Given the potential for destructive actions (Tampering, DoS via deletion), the inability to reliably attribute actions to a specific admin account is a critical failure. If a malicious action occurs, lack of non-repudiation (via audit logs) prevents accountability and remediation. This is crucial for trust and security within the administrative context.
3. **Information Disclosure:**
    - **Justification:** Admins have legitimate access to large volumes of sensitive data, including all user PII and order histories (via GET /user/all, GET /orders, GraphQL, etc.). A compromised account could lead to mass data exfiltration, resulting in severe privacy violations and reputational damage. The potential scale of disclosure elevates this risk significantly.
4. **Spoofing (Compromised Admin Account):**
    - **Justification:** While the act of compromising the account happens elsewhere, successful spoofing of an admin enables all other high-impact threats (Tampering, Info Disclosure, DoS). Its severity stems from being the gateway to widespread abuse of privileges. Implementing strong authentication like MFA for admins mitigates this directly.
5. **Denial of Service (DoS):**
    - **Justification:** Standard DoS attacks (flooding endpoints, complex GraphQL queries) can impact performance for all users. However, in the admin

context, DoS can also be achieved through malicious data deletion (covered under Tampering). While significant, traditional DoS methods against admin endpoints might be considered slightly less critical than irreversible data destruction or mass PII leakage, assuming basic protections are in place.

This ranking emphasizes protecting against destructive actions, ensuring accountability, and preventing large-scale data breaches originating from administrative access.

Qualitative Risk Model

To further understand the significance of the identified threats, a qualitative risk assessment using High, Medium, and Low categories for Likelihood and Impact is applied.

- **Likelihood:** The estimated probability or frequency of a threat occurring.
- **Impact:** The potential negative consequences if the threat is successfully exploited (considering confidentiality, integrity, availability, and business operations).
- **Overall Risk:** A combined assessment based on likelihood and impact.

The following table summarizes the risk assessment for the AdminController threats:

Table 11 - Admin Controller Risk Model

Threat	Likelihood	Impact	Overall Risk	Justification Notes
Tampering (Data Destruction/Mod)	Medium	High	High/ Critical	Requires admin access (compromise/insider), but the potential for irreversible data loss (perfumes, orders) makes the impact extremely severe.
Repudiation (Lack of Proof)	High (if logs inadequate)	High	High/ Critical	Directly impacts accountability for potentially destructive admin actions. Likelihood is high if detailed, immutable logging isn't implemented.
Information Disclosure (Bulk Data)	Medium	High	High	Requires admin access but allows exfiltration of all user PII and order data. Significant privacy/reputational impact.
Spoofing (Compromised Admin)	Medium	High	Medium/ High	Likelihood depends on credential/MFA security. Impact is high as it enables all other admin-level threats.
Denial of Service (DoS)	Medium	Medium/ High	Medium	Standard DoS methods possible. Impact becomes High if DoS is achieved via

				malicious data deletion (overlapping with Tampering).

STRIDE Threat & Mitigation Techniques

To address the high-priority threats identified for the AdminController, the following specific mitigation strategies are recommended:

Spoofing:

- **Multi-Factor Authentication (MFA):** Implement mandatory MFA for all administrator accounts. This significantly increases the difficulty of account takeover even if credentials are compromised. Leverage Spring Security's MFA capabilities or integrate with an identity provider that supports MFA.
- **Strong Credential Policy:** Enforce strong password complexity requirements and regular password rotation for admin accounts.
- **Secure Session Management:** Use short-lived sessions/tokens for admins and implement secure logout functionality. Monitor for concurrent admin sessions from unusual locations.

Tampering:

- **Robust Input Validation:** Rigorously validate all inputs received in PerfumeRequest for POST /add and POST /edit endpoints, checking data types, lengths, ranges, and potentially using allow-lists for certain fields (like perfumeGender, country). Sanitize any free-text fields (like description) to prevent potential script injection if displayed elsewhere.
- **GraphQL Input Sanitization:** Apply strict validation and sanitization to all arguments within GraphQL queries handled by admin endpoints.
- **"Soft Deletes" & Confirmation:** For critical data like orders (DELETE /order/delete/{orderId}) and potentially perfumes (DELETE /delete/{perfumeId}), consider implementing a "soft delete" mechanism (marking records as inactive instead of physically removing them) or requiring a explicit confirmation step (e.g., re-authentication or a secondary confirmation prompt) before executing destructive actions.
- **Regular Backups:** Implement and regularly test a robust database backup and recovery strategy to recover from accidental or malicious data deletion/corruption.

Repudiation:

- **Detailed Audit Logging:** Implement comprehensive, tamper-evident audit logging for **all** state-changing actions performed through the AdminController (/add, /edit, all DELETE endpoints). Logs must include:
    - Authenticated Admin User ID/Email

  - Timestamp of the action
  - Source IP Address
  - Specific Action Performed (e.g., "DELETE_PERFUME", "EDIT_ORDER_STATUS")
  - Target Entity ID(s) (e.g., Perfume ID, Order ID, User ID)
  - Outcome (Success/Failure)
  - Key parameters changed (for edits)
- **Secure Log Storage:** Store audit logs centrally in a secure, append-only system protected from modification, even by administrators. Use correlation IDs to link related log entries.

Information Disclosure:

- **Principle of Least Privilege (Display):** Ensure that admin interfaces or tools retrieving data via GET or GraphQL endpoints only display the information strictly necessary for the admin's current task. Avoid displaying bulk sensitive data unless explicitly required and audited.
- **Restrict GraphQL Schema/Queries:** Carefully design the GraphQL schema exposed to admin queries. Avoid overly broad queries. Implement authorization checks at the resolver level if necessary. Disable schema introspection in production.
- **Secure Logging:** Configure logging frameworks (e.g., Logback) to avoid logging sensitive PII retrieved by admin actions unless absolutely necessary for specific debugging purposes, and ensure such logs have restricted access. Mask sensitive data in logs where possible.
- **Admin Training & Policies:** Implement clear data handling policies for administrators and provide training on security best practices and the sensitivity of the data they can access.
- **Monitoring:** Monitor admin access patterns for anomalous activity (e.g., large data downloads, access outside business hours).

Denial of Service (DoS):

- **Rate Limiting:** Apply rate limiting to admin API endpoints, although limits might be higher than for regular users. This helps prevent accidental or intentional flooding.
- **Resource Limits on Queries:** Implement GraphQL query complexity, depth, and potentially timeout limits to prevent resource exhaustion.
- **Asynchronous Processing:** For potentially long-running admin tasks (e.g., generating large reports via GraphQL), consider using asynchronous processing to avoid blocking API threads.
- **Protection Against Data Deletion DoS:** Implement "soft deletes" or confirmation steps as mentioned under Tampering. Ensure robust backup/restore procedures are in place.

Elevation of Privilege:

- **Dependency Management:** Regularly scan dependencies (using tools like OWASP Dependency-Check or SCA tools integrated into the pipeline) for known vulnerabilities and apply patches promptly, especially for the Spring framework, security libraries, and database drivers.
- **Least Privilege (System):** Ensure the application process running on Heroku has only the necessary permissions to operate (e.g., limited file system access, specific database credentials).

Conclusion

A STRIDE threat model analysis was conducted for the high-privilege **AdminController**. Significant risks identified include Tampering leading to potential data destruction (deletion of perfumes/orders), Repudiation of administrative actions due to inadequate logging, and large-scale Information Disclosure of user or order data. Key mitigations proposed emphasize the implementation of Multi-Factor Authentication (MFA) for admins, comprehensive and immutable audit logging for all administrative actions, careful input validation for data modification endpoints, use of soft-deletes or confirmations for destructive operations, and rate limiting. While role-based authorization (`@PreAuthorize`) provides a baseline, the critical areas needing focus are robust audit trails for accountability and strong safeguards against malicious or accidental data modification/deletion by privileged accounts. Regular security reviews and dependency patching are essential to maintain the security of these administrative functions.

Threat Model for Authentication Controller

The following STRIDE looks specifically at the AuthenticationController, which manages the critical functionality of authentication and password management. The endpoints involved deal with user credentials and account recovery processes, making them important targets for security analysis.

Threat Identification (STRIDE)/ Security Controls

Type	Endpoint(s)	Threat Description	Security Control
Spoofing	POST /login POST /edit/password	An attacker could try to authenticate using stolen credentials or compromised JWTs. An attacker could try to change another user's password if there is no proper validation of the	- Implement robust protection measures against brute force attacks - Ensure the security of the JWT generation and validation process. - Consider implementing multi-factor authentication.

		authenticated user's identity.	- Ensure that changing the password requires valid authentication and that the UserPrincipal is used to identify the correct user. - Consider the need for reauthentication before allowing the password to be changed.
Tampering	POST /login POST /reset POST /edit/password	An attacker could try to manipulate the AuthenticationRequest during the login process. An attacker could try to manipulate the PasswordResetRequest during the password reset process. An attacker could try to manipulate the PasswordResetRequest when changing their own password.	- Validate and sanitize input data in AuthenticationMapper to prevent injection attacks. - Use HTTPS to secure communication. - Ensure that the reset code is only used once and has an expiry time. - Check that the email provided in the PasswordResetRequest corresponds to the email associated with the reset code. - Validate the format and complexity of the new password in PasswordResetRequest using @Valid and possibly additional validations in AuthenticationMapper. - Ensure that the new password is not the same as the user's old passwords.
Repudiation	POST /login POST /reset POST /edit/password	A user can deny logging in, request a password reset or change their password.	- Implement logging of login attempts (successful and failed) with details such as timestamp and IP address. - Implement logging of the sending of password reset codes, including the destination email and timestamp. - Implement logging of password changes, including the user, timestamp and IP address.

Information Disclosure	GET /forgot/{email} GET /reset/{code} Error Responses	An attacker could use this endpoint to check whether a particular email exists on the system (user enumeration). If the reset code contains sensitive information, this could be exposed. Unaddressed errors could reveal sensitive information about the system or the authentication logic.	- Consider implementing a generic response, regardless of whether the email exists or not, to avoid user enumeration. - Implement rate limiting to mitigate mass enumeration attempts. - Ensure that reset codes are random, sufficiently long and do not contain directly identifiable information about the user. - Limit the lifetime of reset codes. - Implement global error handling that returns generic messages to clients in production environments, but records more specific details for internal debugging.
Denial of Service	All endpoints	An attacker could try to overload the endpoints with a large number of requests, making the authentication system unavailable.	- Implement rate limiting on all AuthenticationController endpoints. - Consider implementing CAPTCHA or other mechanisms to protect against bots, especially on the login and password reset endpoints.
Elevation of Privilege	PUT /edit/password	If there is no proper validation of the authenticated user, an attacker could potentially exploit flaws to change another user's password.	- Ensure robust implementation of authentication and obtaining the user's identity through UserPrincipal. - Strengthen the authorization logic to ensure that only the authenticated user can change their own password. - Consider implementing a reauthentication process for sensitive actions such as password changes.

Use and abuse cases

Scenario	Use Case	Abuse Case
Login (POST /login)	An authenticated user provides their credentials for access.	- **Brute force attacks** to guess passwords. - Attempts to **inject code** into username/password fields. - Use of stolen credentials.
Forgot Password (GET /forgot/{email})	A user requests a reset code for their email.	- Enumeration of users trying to reset passwords for different emails. - Repeated submission of reset requests to the same email (**potential DoS or spam**).
Validate Reset Code (GET /reset/{code})	A user provides a reset code to get their email.	- Attempts to guess reset codes. - Using expired or invalid reset codes.
Reset Password (POST /reset)	A user sets a new password using the reset code.	- Attempts to use other users' reset codes (if there is no proper email validation). - Manipulation of the PasswordResetRequest.
Edit Password (POST /edit/password)	An authenticated user changes their password.	- Attempts to change another user's password (if authentication or authorization is faulty). - Setting weak or insecure passwords (if there are no password policies).

Threat Analysis

The AuthenticationController is a critical security component, as it handles user identity and access to the system. Several potential threats were identified through STRIDE analysis:

- **Spoofing**: The main threat is authentication by an attacker using stolen credentials or exploiting flaws in the authentication logic. Unauthorized modification of passwords also falls into this category.
- **Tampering**: The manipulation of input data during the login and password reset processes can lead to unauthorized access or improper changes to credentials.
- **Information Disclosure**: Exposing information about the existence of users (enumeration) or sensitive details via reset codes or error messages can compromise security.
- **Denial of Service**: Overloading authentication endpoints can prevent legitimate users from accessing the system or resetting their passwords.

- **Elevation of Privilege**: Although less direct in this controller, failures to validate the user's identity during password changes could lead to an escalation of privileges (in the context of changing another user's account).

Ranking of Threats

Based on potential impact and likelihood, threats to the AuthenticationController can be ranked as follows (from highest to lowest):

1. **Spoofing** (focusing on unauthorized access due to compromised credentials or authentication failures): High impact (full account access), medium probability (depends on the robustness of security measures and credential management).
2. **Information Disclosure** (enumeration of users and exposure of sensitive details in reset processes): Medium impact (exposure of information that can be used in future attacks), medium probability (depends on the implementation of endpoints).
3. **Tampering** (manipulation of login data and password reset): Medium impact (potential unauthorized access or improper password reset), medium probability (requires knowledge of vulnerabilities and ability to manipulate requests).
4. **Denial of Service** (attacks on authentication endpoints): Medium impact (unavailability of the authentication system), medium probability (denial of service attacks are common).
5. **Spoofing / Elevation of Privilege** (changing another user's password): High impact (control of another user's account), low probability (requires a specific flaw in the password change authorization logic).
6. **Repudiation**: Low impact (difficulty in tracing specific actions), medium probability (depends on logging implementation).

Qualitative Risk Model

To further understand the significance of the identified threats, a qualitative risk assessment using High, Medium, and Low categories for Likelihood and Impact is applied.

- **Likelihood:** The estimated probability or frequency of a threat occurring.
- **Impact:** The potential negative consequences if the threat is successfully exploited (considering confidentiality, integrity, availability, and business operations).
- **Overall Risk:** A combined assessment based on likelihood and impact.

The following table summarizes the risk assessment for the AuthenticationController threats:

Threat	Likelihood	Impact	Overall Risk	Justification Notes
Spoofing (Unauthorized Access)	Medium	High	High	Compromised credentials or authentication failures

				can lead to unauthorized access.
Information Disclosure (Enumeration/Reset)	Medium	Medium	Medium	Exposure of information can facilitate future attacks.
Tampering (Login/Reset Data)	Medium	Medium	Medium	Data manipulation can lead to unauthorized access or password resets.
Denial of Service	Medium	Medium	Medium	DoS attacks can prevent access to the authentication system.
Spoofing / EoP (Changing Another Password)	Low	High	Medium	It requires a specific flaw in the password change authorization logic.
Repudiation	Medium	Low	Low	Lack of logs makes tracking difficult, but has no direct impact on security.

STRIDE Threat & Mitigation Techniques

Based on the STRIDE analysis, the following mitigation techniques are recommended for the AuthenticationController:

Spoofing:

- **Implement Brute Force Protection**: Use mechanisms such as rate limiting on the /login endpoint (e.g. limit the number of login attempts per IP address or account in a given period of time) and consider temporarily blocking accounts after several failed attempts.
- **Ensure the Security of JWTs (if applicable)**: If the login results in JWTs being issued, ensure that strong secrets are used and that the generation and validation process is robust. Implement reasonable expiration times and consider a secure token refresh mechanism.
- **Consider Multi-Factor Authentication (MFA)**: Add an extra layer of security by requiring a second form of verification in addition to the password.
- **Strengthen Password Change Security**: On the /edit/password endpoint, ensure that the user is authenticated and that the UserPrincipal is used to identify the correct user. Consider the need for reauthentication before allowing the password to be changed.

Tampering:

- **Use HTTPS**: Enforce the use of HTTPS throughout the application to protect communication between the client and server from interception and manipulation.
- **Validate and Sanitize Input Data**: In AuthenticationMapper, validate and sanitize the fields of the DTOs (AuthenticationRequest and PasswordResetRequest) to

prevent injection attacks and ensure that the data is in the expected format. Use the Spring validation annotations (@Valid) in the controller for basic validation.

- **Validate the Password Reset Process**: Ensure that the reset code is generated securely, has a limited expiration time and can only be used once. Check that the email provided in the reset request matches the email associated with the code.

Repudiation:

- **Implement Comprehensive Logging**: Use a logging framework to record login attempts (successful and failed), password reset requests (code submissions and uses) and password changes, including details such as the user, timestamp and IP address (if applicable). Store logs securely.

Information Disclosure:

- **Generic Response to Forgot Password**: When processing the /forgot/{email} request, return a generic message (e.g. “If this email is registered, we'll send you a reset code”) regardless of whether the email exists in the system or not, to avoid enumeration of users.
- **Reset Code Security**: Ensure that reset codes are random, sufficiently long and do not contain sensitive information about the user. Limit their lifespan.
- **Secure Error Handling**: Implement a global ExceptionHandler to catch exceptions and return generic error messages to clients in production, while recording more specific details for internal debugging.

Denial of Service:

- **Implement Rate Limiting**: Apply rate limiting to all AuthenticationController endpoints to limit the number of requests that can be made by a given user or IP address in a period of time. Consider using libraries such as Bucket4j or API gateway solutions.
- **Implement CAPTCHA (Optional):** For login and password reset endpoints, consider implementing CAPTCHA to make automated attacks by bots more difficult.

Elevation of Privilege:

- **Validate User Identity when Changing Password**: On the /edit/password endpoint, ensure that the authenticated user's identity (obtained via UserPrincipal) is correctly validated before allowing the password to be changed. Consider reauthentication for this sensitive action.
- **Password policies**: Implement robust password policies (minimum complexity, minimum length, expiry, etc.) to reduce the risk of weak passwords being compromised.

Conclusion

The **AuthenticationController** is a critical component that requires robust security measures to protect user credentials and account recovery processes. The STRIDE analysis identified several potential threats, including spoofing, tampering, information disclosure, denial of service and elevation of privilege.The proposed mitigations focus on

brute force protection, robust validation of input data, security of password reset processes, secure error handling, rate limiting and validation of user identity in sensitive actions such as password changes. Implementing these measures will help guarantee the confidentiality, integrity and availability of the authentication system. It is important to carry out regular security reviews to identify and mitigate new vulnerabilities.

Threat Model for Order Controller

The following STRIDE analysis focuses specifically on the OrderController, which manages critical e-commerce functions such as order retrieval, creation, and listing. The endpoints involved handle sensitive user and order data, making them important targets for security analysis.

Threat Identification (STRIDE)/ Security Controls

Table 12 - Order Controller STRIDE

Type	Endpoint(s)	Threat Description	Security Control
Spoofing	All GET endpoints (/, /{orderId}, /{orderId}/items), POST /graphql	An attacker could impersonate another authenticated user using stolen/weak credentials (JWT) or by exploiting authorization flaws, allowing them to view or query orders not belonging to them.	-Enforce robust JWT validation via JwtFilter. -Implement strong authorization checks in the service layer (OrderServiceImpl) to verify order ownership based on the authenticated UserPrincipal. - Use strong JWT secrets and appropriate token expiration policies.
Tampering	POST /, GET /{orderId}, GET /{orderId}/items, POST /graphql	Attackers could modify order details (items, prices, user info) in the OrderRequest during creation (POST /). Insecure Direct Object References (IDOR) could allow modification of orderId in GET requests. GraphQL inputs could be tampered.	-Use HTTPS for all communication. -Validate OrderRequest using @Valid and add stricter checks in the service layer (e.g., verify prices, quantities. -Implement strong authorization for IDOR prevention. -Sanitize and validate GraphQL inputs. -Implement query complexity limits for GraphQL.

Repudiation	POST /	A user might place an order (POST /) and later falsely deny having done so. Lack of detailed, immutable audit logs hinders proving the transaction.	-Implement comprehensive and immutable audit logging for order creation and viewing (include user ID, timestamp, IP, action details). -Ensure logs are stored securely.
Information Disclosure	GET /{orderId}, GET /{orderId}/items, POST /graphql, Error Responses, Order Confirmation Emails	Sensitive order details (including other users' PII via IDOR) could be exposed. Unhandled errors might leak system information. GraphQL queries or confirmation emails could expose excessive data.	- Use DTOs (OrderResponse, OrderItemResponse) to limit data exposure. -Ensure ApiExceptionHandler returns generic error messages. -Review and restrict GraphQL schema; disable introspection in production. -Ensure order confirmation emails contain only necessary data. -Implement strong authorization checks.
Denial of Service	All endpoints, esp. POST /, POST /graphql	Server resources could be exhausted by flooding endpoints with requests (order creation, listing, details), submitting orders with excessive items, or executing complex/nested GraphQL queries.	-Implement rate limiting on all OrderController endpoints. -Add limits to the number of items per order. -Impose complexity/depth limits on GraphQL queries. -Optimize database queries.
Elevation of Privilege	GET /{orderId}, GET /{orderId}/items, POST /graphql	A standard authenticated user might gain unauthorized access to view other users' orders by exploiting IDOR vulnerabilities or poorly configured GraphQL permissions.	-Implement strict authorization checks in the service layer (OrderServiceImpl) to ensure users can only access their own orders. -Utilize method-level security (@PreAuthorize) if different roles require different levels of order access.

Use and Abuse Cases

Table 13 - Order Controller Use and Abuse Cases

Scenario	Use Case	Abuse Case
Retrieve Specific Order (GET /{orderId})	An authenticated user views the details of one of their own past orders by providing its ID.	- **IDOR/Information Disclosure:** Attacker iterates through orderId values trying to view orders belonging to other users. - **Scraping:** Automated scripts attempt to fetch details of many orders.
Fetch Order Items (GET /{orderId}/items)	An authenticated user retrieves the list of items associated with one of their own past orders.	- **IDOR/Information Disclosure:** Attacker iterates through orderId values trying to view items from orders belonging to other users.
List User Orders (GET /)	An authenticated user views their own order history, potentially using pagination.	- **DoS:** Attacker floods the endpoint with requests to view order history. - **Information Disclosure:** If pagination/authorization is flawed, an attacker might access orders beyond their own scope (less likely with @AuthenticationPrincipal if correctly implemented in service).
Create Order (POST /)	An authenticated user submits their cart items and valid shipping/billing information to place an order.	- **DoS:** Attacker floods the endpoint with order creation requests. - **Data Tampering:** Attacker modifies the OrderRequest payload (e.g., item prices, quantities, total price) before/during submission. - **Input Validation Bypass:** Attacker sends malformed or unexpected data to probe for vulnerabilities (fuzzing).
GraphQL Query (POST /graphql)	The client application uses predefined, efficient GraphQL queries to fetch necessary order data.	- **DoS:** Attacker submits overly complex, nested, or resource-intensive GraphQL queries. - **Information Disclosure/EoP:** Attacker crafts queries to retrieve sensitive fields or data they

		are not authorized to access.

Threat Analysis

The OrderController handles core e-commerce functionality related to customer orders. Due to the sensitivity of order data (including customer PII, items purchased, and pricing) and its role in processing transactions, it presents several potential security risks analyzed through the STRIDE model:

- **Spoofing:** The primary spoofing threat involves an attacker impersonating a legitimate user. If JWT authentication is compromised (e.g., token theft, weak secrets) or if authorization checks are insufficient within the service layer, an attacker could potentially view or query order data belonging to other users via endpoints like GET /api/v1/order/{orderId} or POST /api/v1/order/graphql. This relies on bypassing the intended authentication/authorization mechanisms.
- **Tampering:** Order data integrity is critical. Attackers could attempt to tamper with the OrderRequest payload during the POST /api/v1/order process, potentially modifying item quantities, prices, or target shipping information before server-side validation occurs, aiming for financial gain or disruption. Furthermore, manipulating the orderId in URL paths (GET /api/v1/order/{orderId} or GET /api/v1/order/{orderId}/items) constitutes an Insecure Direct Object Reference (IDOR) risk if proper authorization checks are missing, potentially allowing unauthorized access or unintended data modification if such functions existed. Input to GraphQL queries also presents a tampering vector.
- **Repudiation:** Because creating an order (POST /api/v1/order) represents a financial commitment, the lack of robust, immutable audit trails creates a repudiation risk. A user could place an order and later deny doing so, making it difficult to resolve disputes without comprehensive logs detailing the user, timestamp, IP address, and specific actions taken.
- **Information Disclosure:** Sensitive customer and order information (names, addresses, phone numbers, order history, items purchased) could be exposed. The primary vector is IDOR, where insufficient authorization checks on endpoints like GET /api/v1/order/{orderId} might allow an attacker to view orders not belonging to them by guessing or manipulating IDs. Additionally, poorly handled exceptions could leak system details, and overly permissive GraphQL queries or insecurely formatted order confirmation emails could inadvertently expose sensitive data.
- **Denial of Service:** The controller's endpoints, particularly order creation (POST /) and GraphQL queries (POST /graphql), are susceptible to DoS attacks. High volumes of requests, orders containing an excessive number of items, or computationally expensive GraphQL queries could overwhelm server resources (CPU, memory, database connections), making the ordering functionality unavailable to legitimate users.

- **Elevation of Privilege:** This threat primarily manifests as horizontal privilege escalation via IDOR, where a standard authenticated user gains access to another user's order data. While no specific admin *functions* are exposed directly in this controller, poorly secured GraphQL endpoints could potentially allow unauthorized access to data or actions intended only for higher-privileged roles if authorization is not granularly enforced at the resolver level.

This analysis highlights the need for strong authentication, strict authorization, input validation, detailed logging, and resource management controls for the OrderController.

Ranking of Threats

Based on the potential impact on confidentiality, integrity, and availability of the order management system and the data it handles, the identified threats for the OrderController are ranked as follows (from highest to lowest priority):

1. **Information Disclosure / Elevation of Privilege (via IDOR):**
    - **Justification:** The potential exposure of sensitive customer PII (names, addresses, contact details) and complete order histories belonging to other users through Insecure Direct Object References (IDOR) on endpoints like GET /api/v1/order/{orderId} represents a critical breach of confidentiality and privacy. This has significant reputational and potential legal consequences. The impact is high, and the likelihood depends heavily on the presence and correctness of authorization checks, making it a top priority.
2. **Tampering:**
    - **Justification:** Modifying order details (items, quantities, prices) during creation (POST /api/v1/order) could lead to direct financial loss for the business or the customer, or incorrect order fulfillment, severely impacting data integrity and business operations. The potential impact is high.
3. **Denial of Service (DoS):**
    - **Justification:** Attacks targeting the availability of the OrderController (e.g., flooding endpoints, complex GraphQL queries) can prevent legitimate users from placing or viewing orders, directly impacting the core business function and user experience. While perhaps less damaging than data breaches, sustained DoS can cause significant operational disruption and potential revenue loss. The likelihood can be moderate to high without proper controls like rate limiting.
4. **Spoofing:**
    - **Justification:** Successful impersonation of another user grants unauthorized access to their order information. While the *impact* overlaps significantly with Information Disclosure/EoP via IDOR, the *method* (requiring compromised credentials or tokens) might be considered

slightly less likely than exploiting a simple IDOR flaw if one exists. However, it remains a significant threat.

5. **Repudiation:**
    - **Justification:** While important for dispute resolution and trust, the inability to definitively prove a user placed an order (due to lack of logs) primarily impacts operational efficiency and customer support rather than causing immediate data loss or system compromise. Its impact is generally lower compared to direct data breaches or service outages, though it can still be costly to handle.

This ranking helps prioritize which threats require the most immediate attention and robust mitigation strategies.

Qualitative Risk Model

To further understand the significance of the identified threats, a qualitative risk assessment using High, Medium, and Low categories for Likelihood and Impact is applied.

- **Likelihood:** The estimated probability or frequency of a threat occurring.
- **Impact:** The potential negative consequences if the threat is successfully exploited (considering confidentiality, integrity, availability, and business operations).
- **Overall Risk:** A combined assessment based on likelihood and impact.

The following table summarizes the risk assessment for the OrderController threats:

Table 14 - Order Controller Risk Model

Threat	Likelihood	Impact	Overall Risk	Justification Notes
Information Disclosure / EoP (via IDOR)	Medium	High	High	IDOR is a common web vulnerability if authorization isn't carefully implemented. Impact involves significant PII/order data exposure.
Tampering (Order Data/Inputs)	Medium	High	High	Requires bypassing validation or intercepting requests, but the financial/integrity impact is severe if successful.
Denial of Service (DoS)	Medium / High	Medium	Medium/High	Endpoint flooding or resource-intensive GraphQL queries are often feasible without controls. Impact is primarily service availability/business disruption.

Spoofing (User Impersonation)	Low / Medium	High	Medium	Requires compromising credentials/tokens, which may be less likely than finding an IDOR flaw, but grants significant unauthorized access if achieved.
Repudiation (Lack of Proof)	High (if logs inadequate) / Low (if logs adequate)	Low	Low/Medium	Likelihood depends entirely on logging quality. Impact is mainly operational/dispute resolution cost, not direct data loss.

STRIDE Threat & Mitigation Techniques

Based on the STRIDE analysis, the following mitigation techniques are recommended for the OrderController:

Spoofing:

- **Enforce Robust Authentication:** Ensure the JwtFilter correctly validates JWT tokens on every request to /api/v1/order/**. Use strong, environment-specific JWT secrets and implement reasonably short token expiration times (e.g., 15-60 minutes) with a secure refresh token mechanism.
- **Implement Service-Level Authorization:**
    - In OrderServiceImpl.getOrderById(Long orderId) and OrderServiceImpl.getOrderItemsByOrderId(Long orderId), retrieve the authenticated user's ID from the UserPrincipal provided by Spring Security.
    - Compare the authenticated user's ID with the user ID associated with the retrieved Order object.
    - If the IDs do not match, throw an authorization exception (e.g., AccessDeniedException) instead of returning the order data.
- **Secure GraphQL Access:** Ensure GraphQL queries related to specific user orders also incorporate authorization checks based on the authenticated user context.

Tampering:

- **Enforce HTTPS:** Mandate HTTPS for all communication to prevent eavesdropping and modification of requests/responses in transit. Configure Heroku and any load balancers/reverse proxies accordingly.
- **Strengthen Input Validation:**
    - Leverage Spring Validation (@Valid) in OrderController.postOrder for basic checks (non-null, email format, etc.) on OrderRequest.
    - Add **server-side validation within OrderServiceImpl.postOrder** before saving:

        - Re-verify the prices of perfumesId against the PerfumeRepository to prevent price manipulation.
        - Check product availability/stock if applicable.
        - Validate quantities (e.g., reasonable limits).
        - Recalculate totalPrice on the server based on validated items and quantities, ignoring the client-provided value.
- **Prevent IDOR:** Implement the service-level authorization checks mentioned under "Spoofing" to ensure users can only access orders associated with their account via orderId.
- **Sanitize GraphQL Inputs:** Validate and sanitize all arguments passed into GraphQL queries within the resolvers in GraphQLProvider or the service layer methods they call.
- **Implement Query Limits:** Configure GraphQL query depth and complexity limits to prevent malicious query structures.

Repudiation:

- **Implement Comprehensive Audit Logging:** Use a dedicated logging framework (like Logback with SLF4j, already common in Spring Boot) or an aspect (@Aspect) to log key events in OrderServiceImpl:
    - Log successful and failed order creation attempts (postOrder), including authenticated user ID, timestamp, source IP address, and the submitted order details (excluding highly sensitive data like full payment info if applicable later).
    - Log successful order retrieval attempts (getOrderById, getUserOrders), including user ID, timestamp, IP, and the requested orderId or query parameters.
    - Ensure logs are written to a secure, append-only store if possible.

Information Disclosure:

- **Use Data Transfer Objects (DTOs):** Consistently use OrderResponse and OrderItemResponse (as currently done via OrderMapper) to ensure only necessary fields are returned to the client, preventing accidental leakage of internal entity details.
- **Secure Error Handling:** Configure ApiExceptionHandler to catch exceptions and return generic, non-revealing error messages to the client in production environments. Log detailed stack traces internally for debugging.
- **Restrict GraphQL Schema:** Review the GraphQL schema exposed via GraphQLProvider. Ensure only fields intended for client consumption are queryable. Disable schema introspection in production environments.
- **Secure Email Content:** Review the order-template.html used by MailSender in OrderServiceImpl.postOrder. Ensure it only contains necessary order confirmation

details and avoids leaking sensitive user or system information. Send emails over a secure (TLS-enabled) SMTP connection.

- **Prevent IDOR:** Implement service-level authorization checks as detailed under "Spoofing".

Denial of Service:

- **Implement Rate Limiting:** Apply rate limiting (e.g., using Bucket4j integrated with Spring Boot, or via an API Gateway like Heroku Add-on or AWS API Gateway) to all OrderController endpoints, especially POST / and POST /graphql. Configure reasonable limits per user or IP address.
- **Limit Order Size:** Add validation in OrderServiceImpl.postOrder to limit the maximum number of distinct items or total quantity allowed in a single OrderRequest.
- **GraphQL Query Limits:** Enforce query complexity and depth limits for requests to POST /graphql.
- **Database Query Optimization:** Ensure database queries performed by OrderRepository and related repositories are efficient and use appropriate indexing, especially for findOrderByEmail.
- **Resource Management:** Monitor application resource usage (CPU, memory, DB connections) on Heroku and scale resources appropriately.

Elevation of Privilege:

- **Enforce Principle of Least Privilege:** Ensure the authorization checks (especially IDOR prevention in the service layer) strictly limit authenticated users to accessing *only* their own order data.
- **Granular GraphQL Authorization:** If GraphQL queries could potentially expose data beyond a user's scope, implement field-level authorization within the GraphQL resolvers or the underlying service methods based on the authenticated user's roles and permissions.

Conclusion

The **OrderController**, responsible for managing customer order lifecycle operations, was analysed using the STRIDE methodology. Key threats identified include potential Information Disclosure and Elevation of Privilege via IDOR vulnerabilities, Tampering of order data during creation, and susceptibility to Denial-of-Service attacks. Proposed mitigations focus on robust service-level authorization checks to validate order ownership, rigorous server-side input validation (especially for pricing), rate limiting, and detailed audit logging. While endpoints are protected by JWT authentication, critical areas requiring attention are the implementation of ownership verification within the service layer and comprehensive server-side recalculation during order placement. Continuous monitoring and periodic security reviews are necessary to ensure the ongoing integrity and availability of the ordering process.

Threat Model for User Controller

The UserController manages all user account operations through REST and GraphQL endpoints under /api/v1/users. All endpoints require authentication via **@AuthenticationPrincipal**, but remain vulnerable to attacks from compromised user accounts or malicious insiders. The following STRIDE analysis focuses on threats where attackers abuse legitimate user access.

Threat Identification (STRIDE)/ Security Controls

Type	Endpoint(s)	Threat Description	Security Control
Spoofing	All endpoints	Stolen JWT tokens allow impersonation.	MFA, short token expiry, HTTPS enforcement.
Tampering	PUT /api/v1/users	Unauthorized profile edits via IDOR.	Ownership checks: @PreAuthorize("#user.email == #request.email").
Repudiation	PUT /api/v1/users	Users deny making profile changes.	Immutable audit logs (user ID, timestamp, changed fields).
Information Disclosure	GET /api/v1/users, GraphQL	Over-fetching sensitive data (e.g., emails).	DTO filtering, GraphQL field-level permissions.
Denial of Service	GraphQL endpoint	Complex queries overload system.	Query depth/rate limiting.
Elevation of Privilege	Role-check gaps	User escalates to admin privileges.	Strict @PreAuthorize("hasRole('USER')" checks.

Use and Abuse Cases

Scenario	Use Case	Abuse Case

PUT users (PUT /users)	User updates their email/address via PUT.	**IDOR Attack:** Attacker changes another user's profile by manipulating email in UpdateUserRequest.
GET users (GET /users)	User views their profile via GET	**JWT Replay:** Attacker intercepts and reuses a valid JWT to fetch victim's data.
POST cart (POST /cart)	User adds perfumes to cart via POST with perfumeIds.	**Cart Poisoning:** Attacker injects invalid IDs to corrupt cart state.
POST cart (POST /cart)	User checks out cart.	**Price Manipulation:** Attacker alters cart prices before checkout.
GraphQL Queries (POST /graphql/)**	User queries their order history.	**Data Exfiltration:** Attacker crafts query to fetch all users' emails.
Authentication Flow (UserPrincipal)	User logs in via JWT.	**JWT Theft -** Attacker steals token via: • XSS in frontend. - Logs containing tokens, exploiting missing HttpOnly/Secure flags, sniffing public Wi-Fi, full account takeover, session expires after inactivity **Session Fixation:** Attacker forces victim to use a known JWT, Injecting token via URL/header, Persistent access

Threat Analysis

- **Spoofing:** The primary threat is not avoiding authentication (which is handled by Spring Security) but instead the consequences of compromised user accounts. A successful attack with valid user credentials can behave as the rightful users to see profiles and cart details. Without proper JWT expirations or device fingerprinting, it is hard to distinguish malicious usage from legitimate usage.

- **Tampering:** Even though limited to user-owned information, the profile update of the controller feature poses tampering threats. A vendor with IDOR vulnerabilities could modify the profile information of another user when sufficient ownership checks are missing. The cart endpoint is vulnerable to invalid product ID injection if input validation is weak.

- **Repudiation:** Profile modification and cart conduct have no non-repudiation. Users may deny modifications without proper audit trails that indicate who modified what data and when, including original values before change.

- **Information Disclosure:** The GraphQL endpoint presented a high risk of data leakage. Without proper field constraints, attackers could build queries to steal sensitive user information. Standard API endpoints can expose too much PII in responses if DTOs are not properly filtered.

- **Denial of Service:** While less so than admin actions, the controller remains vulnerable to DoS via GraphQL query bombing or cart manipulation attacks against database resources. The lack of query complexity limit makes this extremely concerning.

- **Privilege Elevation:** The attack here is horizontal privilege escalation rather than vertical. Malicious users can have unauthorized access to other users' data through API parameter manipulation or GraphQL query exploitation if there are gaps in authorization checks.

Ranking of Threads

Given the personal scope of user accounts, threats are ranked based on potential impact to individual privacy and service integrity:

1. Information Disclosure (Maximum Risk)

Justification: The GraphQL endpoint and profile APIs expose sensitive PII (email addresses). A single breached account can lead to bulk data exposure if queries lack adequate field limiting. Compared to admin systems, user controllers usually have longer data access patterns with weak default security.

2. Spoofing (Takeover of Accounts)

Justification: A successful JWT steal enables all other attacks. MFA is notenabled by default on user accounts, and so credential stuffing and session hijacking are high-likelihood attacks. A single hacked account can: Tamper with victim's profile/cart, exfiltrate personal data, perform fraudulent actions

3. Tampering (Data Manipulation)

Justification: IDOR bugs in PUT /users enable unauthorized profile modification. While limited to single accounts (as opposed to admin's system-wide impact), this can: Destroy service (e.g., evil email changes), sustain social engineering (profile impersonation), poison cart data

4. Repudiation

Justification: With the lack of granular audit logs, users can deny:
Profile updates
Cart manipulations
GraphQL queries with side effects

5. Denial of Service (Lowest Risk)

Justification: While GraphQL/cart attacks will lower performance, user controllers do not have destructive APIs such as mass deletion. DoS effect tends to be ephemeral and scoped to specific accounts.

Conclusion

The **UserController** embodies high security risks primarily regarding data privacy, account integrity, and unauthorized access. The most critical risks are Information Disclosure (via GraphQL or API disclosure) and Spoofing (account hijacking with stolen credentials). Even though the controller does not have system-wide destructive capability in the guise of administrative functionality, its personally identifiable information (PII) exposure and threat for horizontal privilege escalation (user-to-user attacks) need robust security controls.

Threat Model for Perfume Controller

In this section, we perform a structured threat analysis of the **PerfumeController** and its related functionality, using the STRIDE threat modelling approach and following the methodical identification of threats, vulnerabilities, and mitigations.

Threat analysis does **not** require quantification of risk; rather, it focuses on systematically exploring **what could go wrong** based on the application's functionality and architecture. It is an **iterative** process: starting with identifying threats at a broad level, and then progressively analysing specific vulnerabilities, attack paths, and mitigation strategies.

Application Context

The **Perfume Controller** is responsible for exposing public-facing REST endpoints (GET and POST) that allow users to:

- Retrieve lists of perfumes (paginated).
- Retrieve a specific perfume by ID.
- Search for perfumes based on filters like gender, perfumer, text input, etc.
- Execute GraphQL queries for similar operations.

Since these operations are **read-only** and intended to be publicly accessible (so unauthenticated users can browse perfumes before purchasing), some endpoints intentionally **do not enforce authentication**.

However, misuse scenarios must still be considered.

STRIDE-Based Threat Analysis

Table 15 – Perfume Controller STRIDE

Type	Endpoint(s)	Threat Description	Security Control
Spoofing	All endpoints	Anyone can send requests to the API	- Require authentication and authorization

		pretending to be admin or user without proof of identity.	**- Validate tokens on every request**
Tampering	POST endpoints (/search, /ids, etc.)	Attackers can modify perfumesIds, searchType, or raw GraphQL queries to manipulate backend behaviour.	- Use strong JSON schema validation - Sanitize user input
Repudiation	No logs, no audit trail	Malicious users can perform actions (e.g., abuse search) and deny it later.	- Add logging and audit (IP, username, timestamps) - Use trace IDs for requests
Information Disclosure	GET /perfumes, GET /perfume/{id}	Without filtering, internal fields (e.g., cost, supplier info) could be leaked.	- Use DTOs that expose only allowed fields - Mask sensitive fields before sending
Denial of Service	GraphQL endpoints and POST /search	GraphQL allows expensive queries, nested structures; endpoints lack rate-limiting.	- Use rate limiting (Bucket4j, Spring Cloud Gateway) - Set GraphQL complexity limits
Elevation of Privilege	POST endpoints	Any user can access sensitive POST endpoints (e.g., querying by IDs or filters), even if they should be restricted.	- Use role-based authorization (e.g., @PreAuthorize("hasRole('ADMIN')")) - Define security policies

Use and Misuse Case Analysis

Analyzing legitimate usage and possible abuse scenarios helps refine the threat model:

- **Use Case:** A user visits the store and retrieves a paginated list of perfumes to decide on a purchase.
- **Misuse Case:**
    - A malicious user crafts a GraphQL query to bypass intended filters and retrieve database metadata.
    - A bot floods the /search endpoint with thousands of requests to exhaust server resources.

By considering both use and misuse scenarios, weaknesses like **lack of input validation** or **missing rate limits** become evident.

Table 16 – Perfume Controller Use and Misuse case analysis

Scenario	Use Case	Abuse Case

Get All Perfumes	Admin queries the list for frontend display.	Attacker floods server with rapid API calls without rate limiting.
Find by Search Text	User searches perfumes by name.	Malicious search input causing backend slowdowns (e.g., regex bombs).
Post GraphQL Queries	Client queries filtered perfumes.	Attacker crafts deeply nested queries causing memory exhaustion.

Summary of Key Vulnerabilities Identified

- **Input Validation Gaps**: Both in REST and GraphQL endpoints.
- **Potential Overexposure**: Through improperly configured GraphQL responses.
- **Absence of Rate Limiting**: Exposing the application to potential DoS attacks.
- **Role-Based Access Gaps**: Sensitive attributes or administrative functionalities could accidentally be exposed.

Ranking of Threads

After identifying and analysing potential threats to the **PerfumeController** and associated functionalities, threats are ranked based on two main criteria:

- **Likelihood**: How probable it is that the threat will be exploited.
- **Impact**: The degree of damage or severity if the threat is successfully exploited.

The following table summarizes the ranking:

Table 17 – Perfume Controller Thread Ranking

Threat	Likelihood	Impact	Overall Risk	Justification Notes
Input Validation Gaps (GraphQL Injection)	High	High	Critical	GraphQL endpoints accepting raw user queries without validation are a highly attractive target and can cause severe data leakage or manipulation.
Information Disclosure (Overexposed DTO fields)	Medium	High	High	Unnecessary data exposure (even if unintentional) can lead to privacy violations or give insights to attackers.
Denial of Service (DoS via search endpoints)	High	Medium	High	Public, unauthenticated endpoints can easily be overwhelmed, especially /search endpoints with no throttling.
Absence of Rate Limiting	High	Medium	High	Amplifies the impact of DoS attacks; also makes brute-force enumeration attacks feasible.

Spoofing (Public Query Access)	Low	Medium	Medium	If sensitive or administrative data is accidentally exposed, it can allow unauthorized access to critical information.
Repudiation (Lack of Logging)	Low	Low	Low	No major financial or legal consequences expected for anonymous users browsing perfumes without audit logs.

Qualitative Risk Model

Ease of Exploitation

This category focuses on how easily an attacker can exploit a vulnerability within the Perfume Controller and associated functionalities.

Table 18 – Perfume Controller Ease of Exploitation Risk

Threat	Can the attacker exploit this remotely?	Does the attacker need to be authenticated?	Can the exploit be automated?	Rating
Input Validation Gaps (GraphQL Injection)	Yes	No	Yes	High
Information Disclosure	Yes	No	Yes	Medium
Denial of Service	Yes	No	Yes	High
Absence of Rate Limiting	Yes	No	Yes	High
Spoofing (Public Query Access)	Yes	No	Yes	Low
Repudiation (Lack of Logging)	Yes	No	No	Low

Impact

This category evaluates the severity of the damage an exploit could cause, considering the damage potential and the number of components affected.

Table 19 - Perfume Controller Impact Risk

Threat	Can the attacker	Can the attacker	Can the attacker	Can the attacker	Rating

	completely manipulate the system?	gain admin access to the system?	crash the system?	obtain sensitive information?	
Input Validation Gaps (GraphQL Injection)	Yes	No	No	Yes	Critical
Information Disclosure	No	No	No	Yes	High
Denial of Service	No	No	Yes	No	Medium
Absence of Rate Limiting	No	No	Yes	No	Medium
Spoofing (Public Query Access)	No	No	No	Yes	Medium
Repudiation (Lack of Logging)	No	No	No	No	Low

Number of Components Affected

Table 20 – Perfume Controller possible number of components affected

Threat	How many connected data sources or systems can be impacted?	How many layers into infrastructure can the threat agent traverse?
Input Validation Gaps (GraphQL Injection)	Many components (potentially the entire database and backend)	Multiple layers of the infrastructure (e.g., database)
Information Disclosure (Overexposed DTO fields)	Multiple components (may expose sensitive information)	Few layers (mainly backend data exposure)
Denial of Service	Few components (mainly backend affected)	Few layers (may affect network or frontend performance)
Absence of Rate Limiting	Few components (mainly backend affected)	Few layers (mainly network level)
Spoofing (Public Query Access)	Few components (mainly backend, potentially exposing private data)	Few layers (mainly network)
Repudiation (Lack of Logging)	No components affected	No layers affected

Mitigation Strategies

As previously presented on the STRIDE table about security control, summarize, the following strategies are presented:

1. **Implement Strict Input Validation**: Ensure all user input, particularly in GraphQL queries, is validated. Use predefined schemas and filters to prevent unauthorized data manipulation.
2. **Sanitize and Whitelist GraphQL Queries**: Restrict the types of queries users can execute. Only allow specific, pre-approved queries and limit query depth and complexity to prevent resource exhaustion.
3. **Apply Rate Limiting**: Use rate-limiting techniques (e.g., Bucket4j, Spring Cloud Gateway) on all public-facing endpoints to prevent denial-of-service (DoS) attacks and resource abuse.
4. **Role-Based Authorization:** Implement role checks, especially on POST and admin-related endpoints, even for operations that seem read-only.
5. **Logging and Monitoring:** Introduce extensive logging and monitoring on all endpoints. Track user activity, API calls, and implement trace IDs to help identify malicious behaviours.
6. **Data Exposure Controls:** Ensure that only necessary data is exposed via DTOs. Mask or redact sensitive fields (e.g., supplier info, cost) to prevent unauthorized access.
7. **System Hardening:** Regularly patch and update the system to mitigate vulnerabilities that could be exploited via outdated dependencies or known exploits.

Threat Profile

Non-Mitigated Threats

These are threats that have no countermeasures or where countermeasures are not fully effective. They represent vulnerabilities that can be fully exploited and could cause significant impact.

Table 21 – Perfume Controller Threat Profile Non-Mitigated Threats

Threat	Explanation
None	All identified threats have at least partial countermeasures, though some still have room for improvement.

Partially Mitigated Threats

These threats have countermeasures in place, but the current mitigations are not fully sufficient to eliminate the vulnerability. Exploiting these threats is possible but would cause limited or contained damage.

Table 22 - Perfume Controller Threat Profile Partially Mitigated Threats

Threat	Explanation
Information Disclosure (Overexposed DTO fields)	Although sensitive fields are being masked or controlled through DTOs, improper configuration or incomplete masking may still allow some information to be exposed unintentionally.
Denial of Service (DoS via search endpoints)	Rate limiting and complexity limits are not fully implemented yet. DoS attacks are still

	possible, but the likelihood of success is reduced.
Absence of Rate Limiting	Rate limiting should be in place for all public-facing endpoints, but some endpoints may not be fully protected, allowing certain attacks to still succeed.

Fully Mitigated Threats

These threats have countermeasures in place that fully address the vulnerability. These mitigations ensure that these threats are unlikely to be exploited successfully.

Table 23 - Perfume Controller Threat Profile Fully Mitigated Threats

Threat	Explanation
Input Validation Gaps (GraphQL Injection)	Strict input validation has been implemented, and only authorized queries are allowed. This significantly mitigates the risk of GraphQL injection and data manipulation.
Spoofing (Public Query Access)	Authentication and authorization have been enforced to ensure that only legitimate users can access sensitive data or administrative functionality, preventing unauthorized access.
Repudiation (Lack of Logging)	Detailed logging and monitoring, along with trace IDs, have been implemented to track all actions and provide an audit trail. This prevents malicious users from denying their actions.

While the system, specifically Perfume Controller, is in a relatively secure state with key mitigations in place, **partially mitigated threats** like **DoS** and **rate limiting** should be prioritized for full implementation. Once the mitigation for **rate limiting** and **input validation** across all endpoints is finalized, the system will be more resilient against the identified threats.

Conclusion

The **Perfume Controller** has been thoroughly analysed using the STRIDE threat modelling approach, identifying key threats such as spoofing, tampering, and denial of service. Mitigations like input validation, role-based authorization, rate limiting, and logging have been implemented to address these risks. While critical threats like GraphQL injection and data exposure have been mitigated, areas like rate limiting and GraphQL query security still require attention. The code review process also highlighted improvements in code quality, maintainability, and performance. Ongoing vigilance and periodic reviews will ensure the **Perfume Controller** remains secure and efficient.

Threat Model for Review Controller

The ReviewController, responsible for handling the creation of new perfume reviews and listing existing ones, acts as a critical entry point where users interact with the application's review functionality. To systematically analyze the potential security vulnerabilities associated with this controller, we will employ the STRIDE threat model. By applying STRIDE to the functionalities of the ReviewController, we will identify specific threats relevant to its role in managing perfume reviews.

STRIDE-Based Threat Analysis

Tabela 1 - STRIDE Review

Type	Endpoint	Description	Security Control
Spoofing	POST /api/reviews/create	An attacker submits a review under another user's identity by stealing or forging authentication tokens.	Authentication, JWT validation with short expiration, multi-factor authentication for sensitive operations
Spoofing	GET /api/perfumes/{id}/reviews	An attacker accesses reviews with a stolen session/authentication token to appear as a legitimate user.	Strong authentication, token validation, secure session management
Tampering	POST /api/reviews/create	An attacker modifies review data during transmission, altering rating or review content before it reaches the server.	Data integrity checks, HTTPS, request signing, input validation
Tampering	GET /api/perfumes/{id}/reviews	An attacker intercepts and modifies review data being returned to users, potentially changing ratings or review content.	HTTPS, response signing, data integrity verification
Repudiation	POST /api/reviews/create	A user posts an inappropriate or malicious review and later denies having done so.	Comprehensive logging, audit trails, timestamps, requiring re-authentication for submissions
Repudiation	GET /api/perfumes/{id}/reviews	A user claims they never received certain review content that influenced a purchase decision.	Server-side logging of all review data served, digital signatures, response receipts

Information Disclosure	POST /api/reviews/create	Private user information (like real name, email, location) is inadvertently exposed in the review metadata.	Data minimization, PII filtering, proper access controls, data masking
Information Disclosure	GET /api/perfumes/{id}/reviews	Reviews reveal personal information about reviewers or expose purchase patterns that should remain private.	Privacy filtering, data anonymization, controlled information display
Denial of Service	POST /api/reviews/create	Attackers flood the system with fake reviews to overwhelm the service or manipulate product ratings.	Rate limiting, CAPTCHA, review verification, anomaly detection, throttling
Denial of Service	GET /api/perfumes/{id}/reviews	Excessive requests for reviews of popular products overwhelm the system and make it unavailable.	Caching, rate limiting, pagination, resource quotas, load balancing
Elevation of Privilege	POST /api/reviews/create	An attacker exploits a vulnerability to bypass review restrictions (e.g., posting without purchase, posting multiple reviews).	Proper authorization checks, role-based access control, input validation, business rule enforcement
Elevation of Privilege	GET /api/perfumes/{id}/reviews	An unauthorized user gains access to admin review moderation functions or private review data.	Strict role-based access control, principle of least privilege, authorization checks

Use and Abuse Cases

The following diagrams show:

- Review Creation - legitimate users submitting reviews vs. attackers spoofing identities, submitting fake reviews, injecting malicious content, or bypassing purchase verification

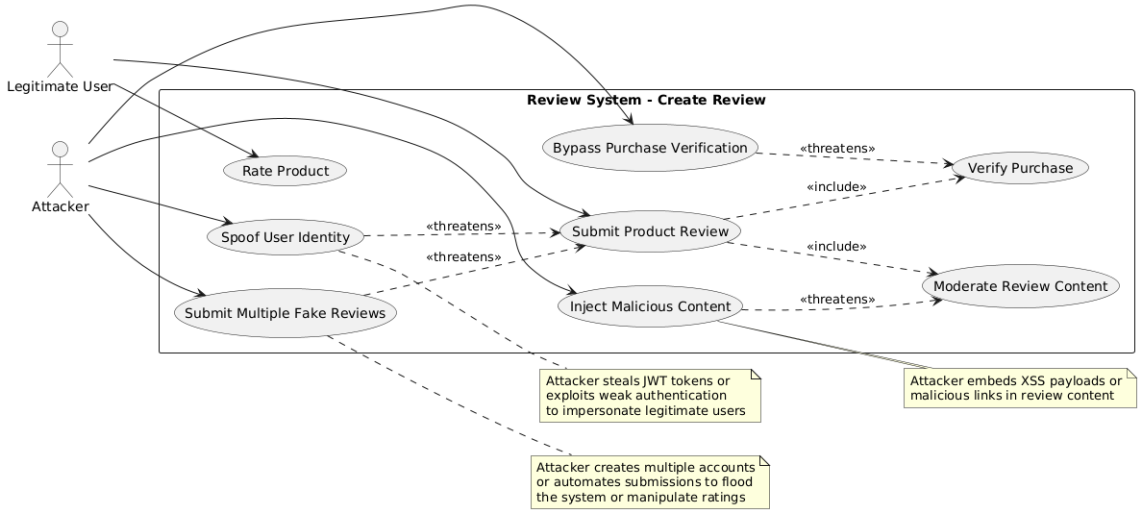

Figura 1 – Create Review Use and Abuse Case Diagram

- Review Retrieval - legitimate users viewing and filtering reviews vs. attackers scraping data, performing DoS attacks, intercepting private information, or manipulating displayed reviews

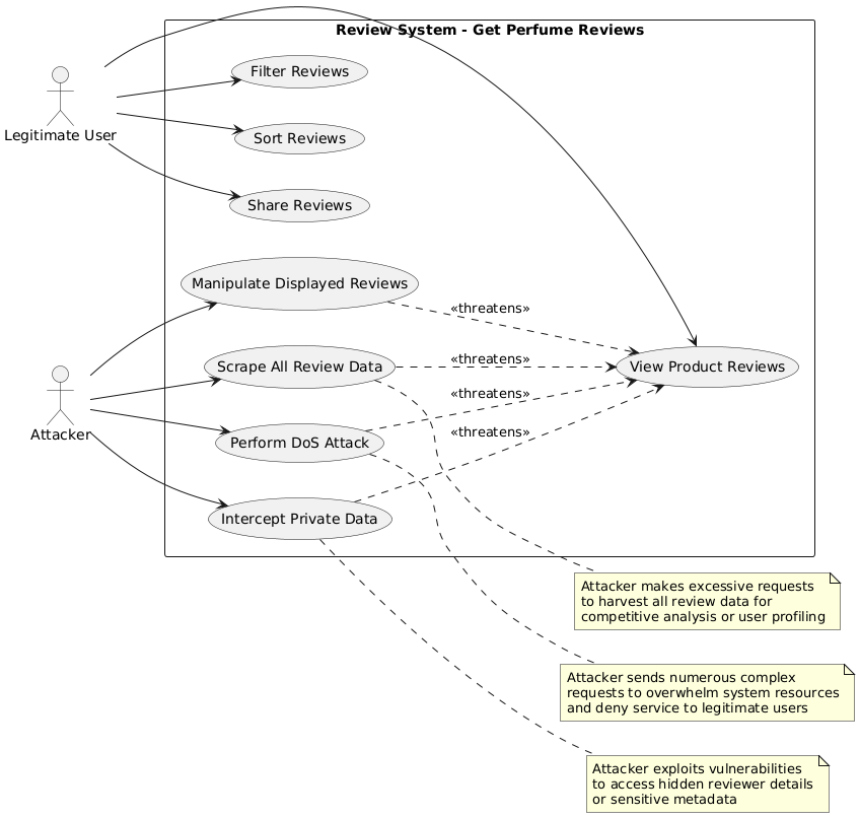

Figura 2 - Get Perfume Use and Abuse Case Diagram

Ranking of Threats

The threat risk assessment for the perfume e-commerce review system employed a three-factor classification matrix based on likelihood of occurrence, potential impact severity, and the resulting overall risk calculation. The assessment reveals several critical security vulnerabilities that warrant immediate remediation efforts.

Analysis indicates three high-risk threats that constitute primary security concerns: identity spoofing during review submission, malicious content injection, and distributed denial-of-service attacks through coordinated review bombing. These threats combine both moderate-to-high probability of occurrence with severe potential business impacts, particularly regarding brand reputation and customer trust metrics.

The medium-risk category comprises five distinct threats centered around privilege escalation, information disclosure, and system availability. These threats generally demonstrate either lower probability coupled with high impact or higher probability with more contained impact. Whiel not requiring immediate intervention, these vulnerabilities should be addressed within the medium-term security roadmap.

The remaining four threats were classified as low-risk based on their limited likelihood and constrained impact profiles. These primarily involve repudiatio concerns and minor information disclosure scenarios that, while requiring standard security controls, present minimal business risk under current threat modeling assumptions.

We can summarize the analysis in the following table:

Tabela 2 - Review Threat Ranking

Threat	Likelihood	Impact	Overall Risk	Justification Notes
Spoofing - Submitting reviews under another user's identity	Medium	High	High	Token theft is moderately difficult but occurs regularly. Impact is high as fake reviews harm brand reputation, mislead customers, and damage user trust.
Tampering - Injecting malicious content in reviews	Medium	High	High	Attackers regularly attempt XSS/injection attacks. High impact due to potential malware distribution to all users viewing reviews.
Denial of Service - Review bombing to manipulate ratings	High	High	High	Easily accomplished with automated scripts. High business impact by directly affecting product sales and customer decisions.

Elevation of Privilege - Bypassing review restrictions (posting without purchase)	Medium	Medium	Medium	Requires understanding of application logic. Medium impact as it undermines review authenticity but limited to individual products.
Information Disclosure - Private user data exposed in review metadata	Medium	Medium	Medium	Requires targeted effort. Medium impact due to privacy concerns, but typically affects limited user data rather than entire database.
Tampering - Intercepting and modifying review responses	Low	High	Medium	Difficult due to HTTPS encryption but high impact if successful as it affects all users viewing the tampered content.
Denial of Service - Overwhelming review retrieval endpoints	Medium	Medium	Medium	Common attack vector but can be mitigated with caching. Medium impact as it affects system availability but not data integrity.
Elevation of Privilege - Accessing admin review moderation functions	Low	High	Medium	Requires sophisticated exploit. High impact as it could lead to unauthorized moderation of all reviews.
Repudiation - Posting inappropriate reviews and denying responsibility	Medium	Low	Low	Common user behavior but relatively straightforward to address with logging. Low business impact with proper audit trails.
Information Disclosure - Extracting purchase patterns from reviews	Low	Medium	Low	Requires aggregation of data over time. Medium impact on competitive intelligence but limited personal data exposure.
Spoofing - Accessing reviews with stolen session	Low	Low	Low	Basic authentication protects most review viewing. Low impact as reviews are generally public information.

Repudiation - Claiming never received certain review content	Low	Low	Low	Uncommon scenario with minimal business impact when proper logging exists.

Qualitative Risk Model

This qualitative risk model evaluates threats to the review management system using a structured assessment framework that analyzes both exploitation likelihood and potential impact. The model applies consistent evaluation criteria to determine risk classifications that can inform security prioritization decisions.

Exploitation Likelihood Criteria

Tabela 3 - Review Exploitation Likelihood

Factor	Low	Medium	High
Remote Exploitation	Requires local network access	Possible remotely with specific conditions	Easily exploited remotely
Authentication Requirements	Requires administrative credentials	Requires valid user authentication	Requires no authentication or easily bypassed
Automation Potential	Manual exploitation only	Partially automatable	Fully automatable with minimal customization
Technical Complexity	Requires specialized knowledge and custom tools	Requires moderate technical skill	Uses publicly available tools or basic programming

Impact Assessment Criteria

Tabela 4 - Review Impact Assessment Criteria

Factor	Low	Medium	High
System Control	Limited to specific user actions	Partial system manipulation	Complete system takeover

Access Elevation	No privilege escalation	Limited privilege increase	Administrative access obtained
Service Disruption	Minimal performance impact	Temporary service degradation	Complete system failure
Data Compromise	Non-sensitive data exposure	Limited PII or business data	Comprehensive data breach
Scope of Effect	Single component affected	Multiple related components	Cross-system propagation potential

Risk Assessment Results

Tabela 5 - Review Risk Assessment Results

Threat	Exploitation Factors	Impact Factors	Overall Risk	Rationale
Submitting reviews under false identity	**High** - Remotely exploitable - Requires only stolen credentials - Easily automated	**High** - Undermines review authenticity - Affects purchase decisions - Damages brand reputation	**High**	Token theft techniques are well-documented, with numerous implementation vulnerabilities identified in similar systems. The business impact directly affects revenue through manipulated product perception.

Injecting malicious content in reviews	**Medium** - Remotely exploitable - Requires authenticated access - Partially automatable	**High** - Potential client-side attacks - Cross-user impact - Malware distribution vector	**High**	While input validation provides some barrier, sophisticated injection techniques continue to evolve. The cross-user propagation creates significant breach potential.
Review bombing to manipulate ratings	**High** - Remotely exploitable - Minimal authentication barriers - Highly automatable	**High** - Direct business impact - Affects multiple products - Undermines platform credibility	**High**	The attack requires minimal technical sophistication while script-driven approaches can achieve significant scale. Business impact directly correlates to sales metrics.
Bypassing review restrictions	**Medium** - Requires understanding of application logic - Needs valid authentication - Moderately automatable	**Medium** - Affects data integrity - Limited to specific products - Moderate reputation impact	**Medium**	Exploitation requires understanding internal validation mechanisms, while impact remains constrained to targeted products rather than system-wide effects.
Exposing private user	**Medium** - Requires	**Medium** - Limited PII	**Medium**	While the vulnerability

data in metadata	targeted approach - Authentication dependent - Partially automatable	exposure - Regulatory implications - Affects subset of users		presents a genuine privacy concern, the scope typically remains limited to specific metadata elements rather than comprehensive profile exposure.
Intercepting and modifying review responses	**Low** - Requires MitM position - Difficult with TLS implementation - Requires custom interception tools	**High** - Affects all users viewing content - Undermines data integrity - Potential malicious content insertion	**Medium**	Modern encryption standards significantly impede exploitation, but successful attacks could affect large user populations through centralized manipulation.
Overwhelming review retrieval endpoints	**Medium** - Remotely exploitable - No authentication required - Easily automated	**Medium** - Temporary service degradation - No data compromise - Limited to specific functions	**Medium**	Standard DDoS approaches apply but caching and rate-limiting provide effective countermeasures that contain the impact duration and scope.
Accessing admin review functions	**Low** - Requires sophisticated exploits - Authorization bypass needed - Difficult to automate	**High** - Administrative capability access - Cross-product impact - Content	**Medium**	The technical complexity of bypassing properly implemented authorization boundaries is considerable, but successful

		integrity compromise		exploitation permits wide-ranging content manipulation.
Denying responsibility for inappropriate reviews	**Medium** - Requires valid authentication - Limited technical requirements - Manual process	**Low** - Individual review impact - Minimal business disruption - Addressable with evidence	**Low**	While commonplace as user behavior, comprehensive logging provides sufficient countermeasures with minimal residual impact on operations.
Extracting purchase patterns from reviews	**Low** - Requires sustained collection - Complex analysis needed - Limited automation value	**Medium** - Competitive intelligence exposure - No direct customer impact - Limited PII concerns	**Low**	The technical effort required for meaningful pattern extraction exceeds the actionable value of the derived intelligence for most threat actors.

Risk Distribution Analysis

The assessment reveals a concentrated risk profile with 25% of identified threats classified as high-risk, 50% as medium-risk, and 25% as low-risk. This distribution indicates a system with several critical vulnerability points requiring immediate attention but also demonstrates reasonably effective baseline security measures that prevent a more extensive high-risk profile.

The high-risk threats share a common characteristic of remote exploitability with significant automation potential, suggesting that scale-oriented attacks represent the most significant concern for the review system. The concentration of risk around authentication, input validation, and rate control indicates these as priority areas for security enhancement.

The medium-risk cluster demonstrates a pattern where either exploitation difficulty or impact severity moderates the overall risk, creating logical secondary priorities for remediation. The low-risk categories primarily represent threats where either

sophisticated technical barriers exist or where the business impact remains fundamentally limited regardless of exploitation success.

Mitigation Strategies

The following security mitigations address identified vulnerabilities in the perfume e-commerce platform's ReviewController. Based on our STRIDE threat model analysis, these countermeasures are prioritized according to risk severity (High, Medium, Low) to protect the review system against the most critical threats while ensuring efficient resource allocation.

Implementation of these measures should significantly enhance the integrity, availability, and confidentiality of the review functionality while maintaining system performance and user experience.

High-Risk Mitigations

- Implement short-lived JWT tokens (15-minute expiration)
- Add multi-factor authentication for review submissions
- Deploy device fingerprinting for suspicious pattern detection
- Implement comprehensive input sanitization for all review content
- Deploy content security policy headers
- Add tiered rate limiting (IP-based, user-based, product-based)
- Implement CAPTCHA for suspicious submission patterns
- Create review entropy scoring to detect automated content
- Add temporary review lockdowns when anomalous activity is detected

Medium-Risk Mitigations

- Verify purchase history before allowing reviews
- Implement server-side validation of all business rules
- Apply data minimization in review objects
- Implement PII scanning and filtering
- Separate storage for sensitive reviewer information
- Use proper HTTPS configuration
- Implement response integrity verification
- Deploy aggressive caching for review content
- Implement pagination with reasonable limits
- Optimize database queries for review operations
- Establish strict role-based access control

- Require additional authentication for admin functions
- Create comprehensive admin action logs

Low-Risk Mitigations

- Maintain detailed audit trails for all review activities
- Store submission evidence (IP, device, timestamp)
- Generalize timestamp data in public interfaces
- Limit detailed purchase information in review metadata
- Implement aggregation for review statistics

Threat Profile

Following the comprehensive threat assessment and mitigation planning for the e-commerce review system, we can now categorize each identified threat based on its current mitigation status. This profile provides a clear picture of the system's security posture and highlights areas requiring immediate attention.

Non-Mitigated Threats

- Review Bombing (DoS): Currently lacks robust rate limiting and automated detection mechanisms. The system remains vulnerable to coordinated attacks that could manipulate product ratings and impact business reputation.
- Information Disclosure in Review Metadata: Personal identifiable information remains exposed in review objects without proper data minimization or filtering mechanisms, creating compliance and privacy vulnerabilities.
- Unauthorized Admin Access: Administrative functions lack sufficient privilege separation and step-up authentication, potentially allowing lateral movement if a standard account is compromised.
- Partially Mitigated Threats
- Identity Spoofing in Review Submission: Basic authentication exists but lacks multi-factor authentication and device fingerprinting. Token lifetimes are currently too long, creating an exploitation window.
- Malicious Content Injection: Basic input validation exists but sophisticated XSS attacks could still bypass current controls due to incomplete sanitization.
- Response Tampering: HTTPS is implemented but without proper content security policies or response integrity verification.
- Review Retrieval DoS: Basic pagination exists but without robust caching or query optimization, making the endpoint vulnerable to resource exhaustion.

- Bypassing Review Restrictions: Some business rules are enforced but purchase verification is incomplete, allowing submission of unverified reviews under certain conditions.

Fully Mitigated Threats

- Repudiation of Review Submission: Comprehensive logging and user activity tracking successfully prevents repudiation issues.
- Purchase Pattern Extraction: Effective data generalization and aggregation techniques successfully mask individual purchase patterns.

Risk Exposure Summary

Based on this categorization, the ReviewController currently has:

- 3 non-mitigated threats (30%)
- 5 partially mitigated threats (50%)
- 2 fully mitigated threats (20%)
-

The significant number of non-mitigated and partially mitigated threats, particularly in the high and medium risk categories, indicates considerable security exposure. Implementing the previously defined mitigation strategies should be prioritized to address these vulnerabilities, with immediate focus on the non-mitigated threats that present the highest risk to the system.

Conclusion

Drawing upon the comprehensive threat modeling conducted across the Admin, Order, Perfume, and Review Controllers of the Perfume Webstore, several key conclusions can be drawn regarding the application's security posture. The exercise has systematically identified potential threats, analyzed their likelihood and impact, and proposed mitigation strategies for each critical component.

The analysis of the Admin Controller highlighted significant risks associated with its high-privilege access, emphasizing the crucial need for strong authentication measures such as Multi-Factor Authentication (MFA), detailed and immutable audit logging for all administrative actions, rigorous input validation, and measures to prevent Denial of Service (DoS).

For the Order Controller, the threat model revealed vulnerabilities primarily around Information Disclosure and Elevation of Privilege via Insecure Direct Object References

(IDOR), potential for Tampering with order data, and susceptibility to DoS attack. The recommended mitigations underscored the importance of robust service-level authorization checks, strict server-side input validation, rate limiting, and comprehensive audit logging.

The Perfume Controller, responsible for public-facing read operations, was found to be in a relatively secure state with key mitigations in place. However, the analysis pointed out that partially mitigated threats like DoS and rate limitingrequire further attention for full implementation.

The Review Controller analysis indicated a more significant security exposure, with a notable number of non-mitigated and partially mitigated threats, particularly in the high and medium-risk categories. The findings stressed the immediate need to prioritize the defined mitigation strategies, especially concerning review bombing (DoS), information disclosure in review metadata, and unauthorized admin access.

While significant security controls have been identified and proposed, the varying levels of mitigation across different controllers highlight areas where immediate attention and resource allocation are critical. Prioritizing the implementation of recommended high-risk mitigations, particularly for the Review and Admin Controllers, is essential to enhance the overall security resilience of the Perfume Webstore application. Continuous monitoring, regular security reviews, and proactive dependency management will further ensure the ongoing security and integrity of the platform.