Blue text on a black background

Description automatically generated

**BAKU HIGHER OIL SCHOOL**

**INFORMATION TECHNOLOGY DEPARTMENT**

**“Information Security” Division**

**Midterm Project Documentation**

**Security Practices on CaseStudy “SecureWeb”**

**Names of students in the project group:**

Natig Mammadov, Umman Mammadov, Mirhabil Sariyev

**Course:** WEB Programming and Security

**Instructor:** Elnur Bagirov

**Group number:** IS22

# 1. Advanced Session Management

**Objective**

The goal of this feature in our Java based program is to enhance the security and manageability of user sessions within the web application. The two main enhancements are:

1. **"Remember Me" functionality**: Allowing users to remain logged in across sessions securely.
2. **Session activity logging**: Monitoring and detecting anomalies in session behavior to prevent unauthorized access and potential attacks.

1.1. "Remember Me" Functionality Using Secure Cookies

The "Remember Me" feature allows users to persist in their login session beyond the typical expiration time of a session cookie. This functionality is particularly useful for users who prefer not to log in every time they visit the application.

**Design and Implementation**

1. **Secure Cookie Storage**:
   * A cookie containing a token is used to identify users who have opted into the "Remember Me" functionality.
   * The token is stored in a secure, **HttpOnly** cookie with the **Secure** flag enabled, ensuring that it is only transmitted over HTTPS connections and is inaccessible to JavaScript, reducing the risk of Cross-Site Scripting (XSS) attacks.
   * The cookie has an expiration time (for example, 5 days) after which the session will expire, and the user will need to log in again.
   * With the token associated with a unique, randomly generated session identifier stored server-side, access to protected pages is restricted to authenticated users through validation.

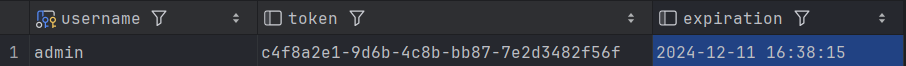
A screenshot of a login screen

Description automatically generated

A computer screen shot of code

Description automatically generated

1. **Token Generation**:
   * The token contains the user's unique ID, roles, and any relevant session data. It is signed with a secret key to ensure its integrity and authenticity.
   * When a user logs in and opts for the "Remember Me" option, the application generates a secure token. The token is then stored in the browser’s cookie, and it is also stored in the database with a corresponding user ID.



1. **Session Token Validation**:
   * On subsequent requests, the server checks for the presence of the "Remember Me" cookie.
   * If the cookie is found, the server validates the token by verifying the signature, checking for expiration, and ensuring that it matches the user’s session stored in the database.
   * If the token is valid, the user is automatically logged in. If invalid (e.g., expired or tampered), the user is asked to log in again.

The saved token can be found in Browser’s developer tools -> inspection -> application -> cookies:



**Security Considerations**

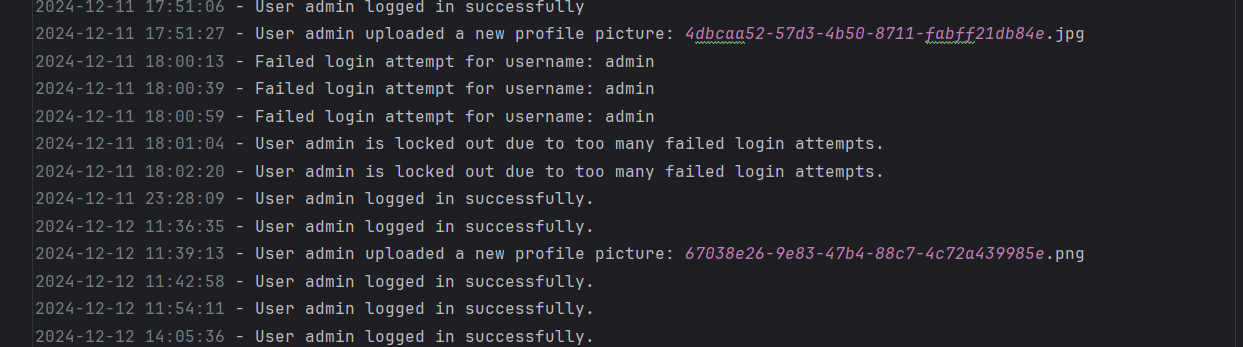
* **Token Regeneration**: Each time a user logs in with the "Remember Me" option, the application generates a new token to prevent old tokens from being reused.
* **Session Expiration**: The session has a finite lifespan, and the token is automatically invalidated after the expiration date (for example, 5 days).
* **Token Integrity**: The JWT is signed using a strong hashing algorithm (like HMAC-SHA256) and cannot be tampered with. If the token signature is invalid, the user is prompted to log in again.
* **Secure Cookies**: Cookies are set with the HttpOnly and Secure flags to prevent client-side JavaScript from accessing the cookie, and to ensure cookies are transmitted only over HTTPS connections.

1.2. Session Activity Logging to Detect Anomalies

Session activity logging is a critical part of detecting suspicious activity, such as session hijacking, brute-force attacks, and unauthorized access. This logging ensures that any unusual or potentially malicious session activities are captured, and alerts can be triggered for investigation.

**Security Design and Implementation**

1. **Logging Key Session Events**:
   * **Login Attempts**: Successful and failed login attempts are logged, including the user ID, timestamp, and source IP address.
   * **Session Creation and Destruction**: Each session creation (login) and destruction (logout or session timeout) event is recorded, including the session ID and user information.
   * **Session Activity**: Every significant user action (e.g., uploading a file, changing settings) is logged with the user ID, session ID, timestamp, and action type.
   * **Failed Access Attempts**: Access attempts to unauthorized resources (accessing an admin-only page without proper permission) are logged.
2. **Log Format**:
   * Logs are stored in a structured format, such as JSON or plain text, for easy querying and analysis.
   * Each log entry includes:
     + **User ID**: The ID of the user performing the action.
     + **Session ID**: The ID of the session the user is using.
     + **Action**: A description of the action (e.g., “login attempt”, “file upload”, etc.).
     + **Timestamp**: The exact time of the action.
     + **IP Address**: The IP address from which the request originated.
     + **Outcome**: The result of the action (success or failure).
3. **Anomaly Detection**:
   * Anomalies, such as multiple failed login attempts from the same IP address, sudden session expiration, or inconsistent session behavior, are flagged as suspicious.
   * Multiple failed login attempts within a short time (e.g., brute-force attack).
   * Same session ID being used on different devices or IP addresses.
4. **Log Storage**:
   * Logs are stored securely, either in a database or in log files with restricted access.
   * To prevent tampering, logs are **write-only** for the application, with access restricted to administrators or security personnel.
   * Logs should be retained for a specific period (e.g., 5 days) for auditing purposes and to comply with data retention policies.



**Conclusion**

The advanced session management feature ensures that user sessions are secure, persistent (for "Remember Me" functionality), and monitored for any suspicious activity. By utilizing secure cookies and session activity logging, the system can mitigate risks like session hijacking and brute-force attacks while providing administrators with tools to detect and respond to security incidents effectively.

# 2. User Role-Based Access Control (RBAC)

**Objective**

To implement role-based access control (RBAC) in the application, restricting user access to features based on their roles. The three roles are:

Admin: Full access to all features.

Moderator: Can manage uploaded files.

User: Limited to uploading files.

**Roles and Permissions**

Admin:

Full access to all application features:

* user management
* file uploads
* file deletions
* system settings.

Can assign roles to other users.

Moderator:

Semi-access to application features:

* view
* approve uploaded files
* delete uploaded files.

Cannot manage user accounts or access admin settings.

User:

Limited access to application features:

* only upload files.

No permissions for file management or system settings.

**Implementation**

*Role Assignment*:

Roles are assigned at user creation or through an admin interface.

Each user has a role attribute, which is stored in the database (role: 'Admin', role: 'Moderator', role: 'User').

A screen shot of a computer code

Description automatically generated

*Access Control:*

Token Authentication: Roles are included in the token after successful login. The token is validated on each request.

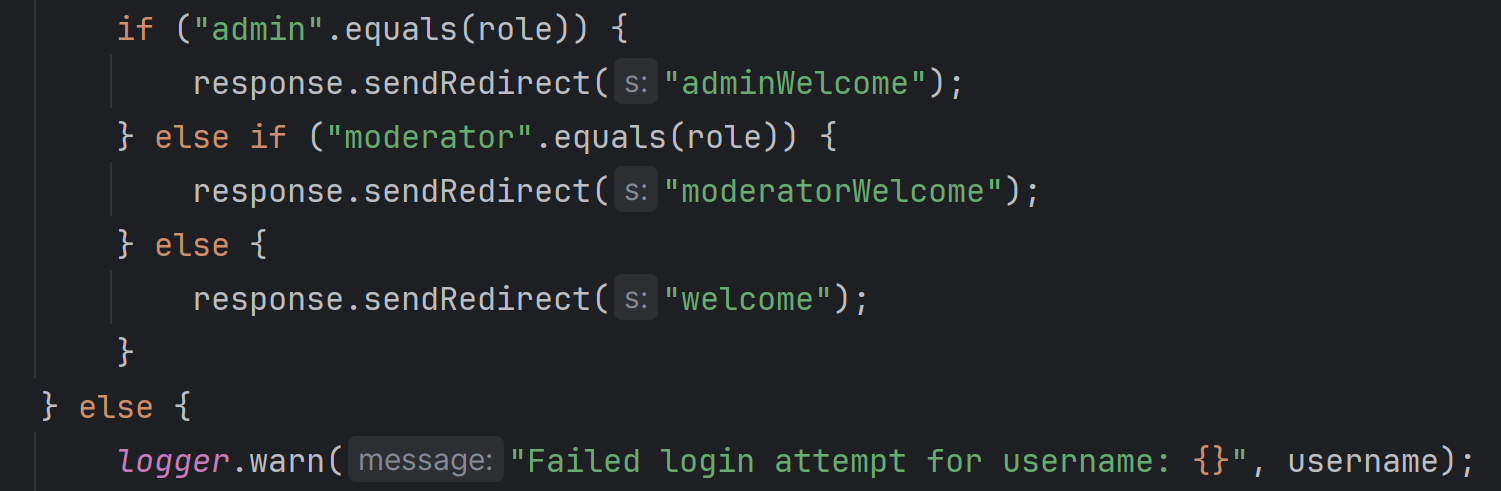
*Authorization Filters:*

Role-based authorization filters are used to restrict access to endpoints. For example:

Admin-specific endpoints are protected by checking if the user’s role is Admin.

Moderator endpoints are protected by checking if the user’s role is Moderator or Admin.

User endpoints are accessible by anyone with the User role.



*Access Control Logic:*

When a user attempts to access a resource, the system checks their role embedded in the JWT token.

The system denies access to resources that the user’s role is not authorized to access.

A screen shot of a computer program

Description automatically generated

**Security Considerations**

Role Validation: Roles are securely stored and validated on each request to ensure proper access control.

Least Privilege: Users only have the minimum privileges required for their role, reducing the risk of privilege escalation.

Preventing Unauthorized Access: Unauthorized access to resources (e.g., an Admin-only page accessed by a User) is prevented by role-based authorization checks.

**Conclusion**

RBAC ensures that users can only perform actions relevant to their role, protecting sensitive resources and maintaining a clear separation of duties. By using roles (Admin, Moderator, User) and authorization checks, the system can enforce proper access control across the application.

# 3. Enhanced File Upload Security

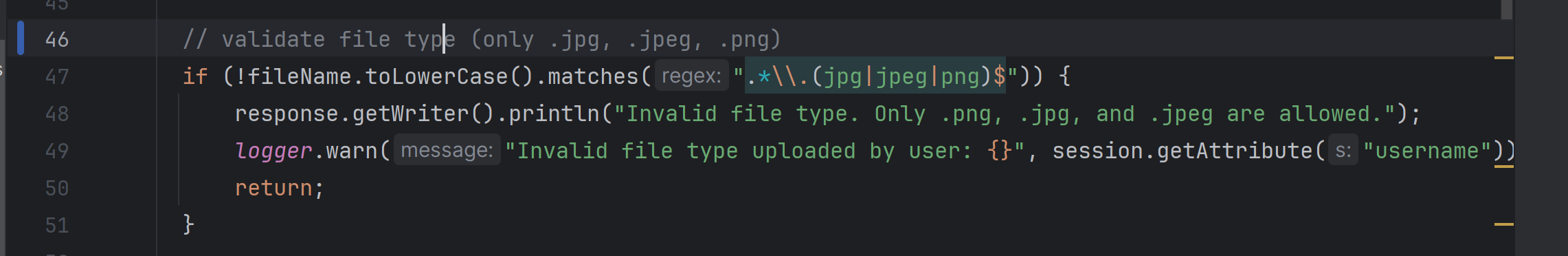
**Objective**

To improve the security of the file upload feature by restricting allowed file types, scanning files for malware, and enforcing file size limits. This will prevent malicious files from being uploaded and ensure that only safe, authorized content is processed.

**Implementation Overview:**

File upload security ensures secure file uploads by restricting accepted file types to images (JPG, PNG) and enforcing a maximum file size of just a few MB. Files that do not meet these criteria are rejected, and users are notified. Additionally, all upload activities are logged for auditing and security purposes.

We restricted file uploads to only image files (JPG, PNG) by checking the file's MIME type and extension. This ensures that users cannot upload potentially dangerous files like executable files. We use regex to filter out the filename.



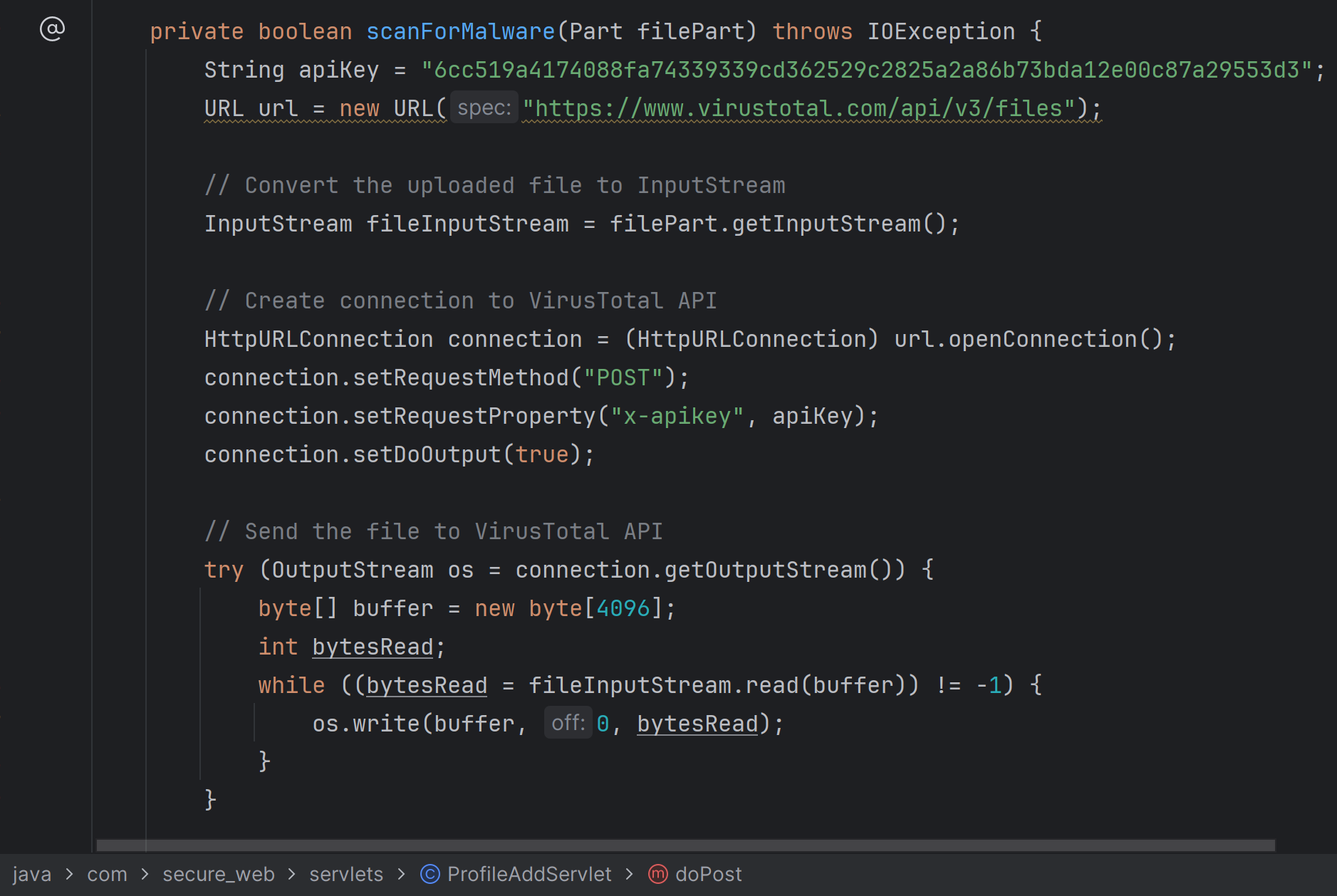
**Security Considerations**

*Avoid Executable Files*: Ensure that no executable file types (for example, .exe or .bat) can be uploaded. This can be done by checking both the file extension and MIME type.

*Scan files for malwares*: Check the uploaded file for malware detection.

A black background with green and blue text

Description automatically generated



*Limit Max Upload Size*: Prevent user uploading bigger files (limiting the image file size).

A screen shot of a computer program

Description automatically generated

**Conclusion**

Enhancing file upload security involves multiple layers of protection: validating file types, scanning for malware, and enforcing file size restrictions. By implementing these controls, we reduce the risk of malicious file uploads, ensuring that only valid and safe files are processed. Additionally, secure handling and storage practices are critical to safeguarding the application from potential threats.

# 4. Secure API Development

**Objective**

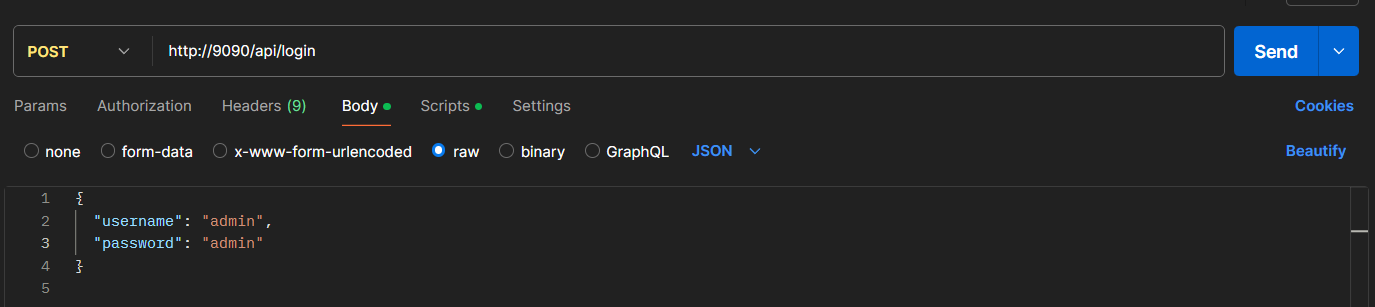
To expose secure RESTful APIs for user login and file uploads while implementing rate limiting to prevent abuse. The goal is to ensure that the API is secure, efficient, and resistant to common attacks like brute force or denial of service (DoS).

4.1. User Login API

**Objective**: Implement a secure endpoint for user login that validates credentials and returns an authentication token.

**Login Endpoint**:

* Users submit their credentials (username/email and password) via a POST request.



* The server validates the credentials and, if successful, generates a **JWT** that is returned to the client for subsequent authenticated requests.

A close up of a screen

Description automatically generated

**JWT Token**:

* **JWT** contains user information, such as ID and roles, and is signed using a secret key to ensure its integrity.
* The token is sent in the **Authorization** header (Bearer <token>) in subsequent requests.

**Security Considerations**:

* Use **HTTPS** for all communications.
* Store passwords securely using hashing algorithm.
* Implement rate limiting (explained below) to protect against brute force attacks.

4.2. File Upload API

**Objective**: Provide a secure file upload endpoint for authenticated users.

**Implementation:**

1. **File Upload Endpoint**:
   * Users submit files via a POST request.

A screenshot of a computer

Description automatically generated

* + The server checks the **JWT token** to ensure the user is authenticated.
  + The file is validated for type, size, and security (e.g., malware scanning) before being stored

A black background with white text

Description automatically generated

1. **File Validation**:

* Restrict file types (e.g., images only), scan for malware, and ensure files are within size limits (e.g., 5MB).

1. **Security Considerations**:

* Require authentication via **JWT**.
* Use **role-based access control (RBAC)** to ensure only authorized users can upload files.
* Protect the endpoint with **rate limiting** (explained below).

**Conclusion**

The secure API development process involves careful handling of user authentication via JWT tokens, secure file upload validation, and proactive rate limiting to prevent abuse. By applying role-based access control (RBAC) and ensuring all endpoints are protected via HTTPS, the application can maintain both security and usability for users while minimizing the risk of attacks like brute-force or denial-of-service.

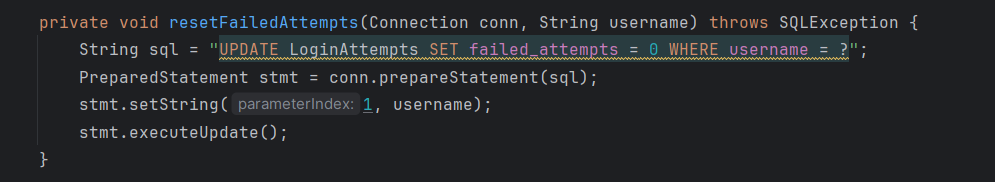
# 5. Comprehensive Logging and Monitoring

**Objective**

To implement an advanced logging and monitoring system that tracks significant user actions, sends alerts for suspicious activities (e.g., multiple failed login attempts), and stores logs securely for future auditing and incident response.

**Implementation:**

1. **Types of Logs to Track**:
   * **Login Attempts**: Track both successful and failed login attempts, including the username, timestamp, IP address, and login outcome (success or failure).



* + **Failed or mal-intended attempts:** If a user constantly tries to login or does DoS attack to the system the user gets locked out after a few failed attempts.

A computer screen shot of a program code

Description automatically generated

A screen shot of a computer program

Description automatically generated

* + **File Uploads**: Log file uploads, including user ID, file name, file size, timestamp, and the result of the upload (success or failure).
  + **User Role Changes**: Log any changes to user roles or permissions.
  + **System Events**: Log server or system-level events like startup, shutdown, or error messages.

1. **Log Format**:
   * Use a structured log format like **JSON** for easy parsing and querying in each log entry:
     + **Timestamp**: The time the action occurred.
     + **User ID**: The ID of the user performing the action.
     + **Action**: A description of the action (e.g., "Login attempt", "File upload").
     + **IP Address**: The IP address from which the request originated.
     + **Outcome**: The result of the action (e.g., "Success", "Failure").
     + **Details**: Any additional relevant information (e.g., file name, error message).

A screen shot of a computer code

Description automatically generated

**These practices will help to create robust and secure web environments and enhance user experience, with real-time alerts and secure log storage provides ongoing visibility into suspicious activities, enabling swift responses to potential threats!**