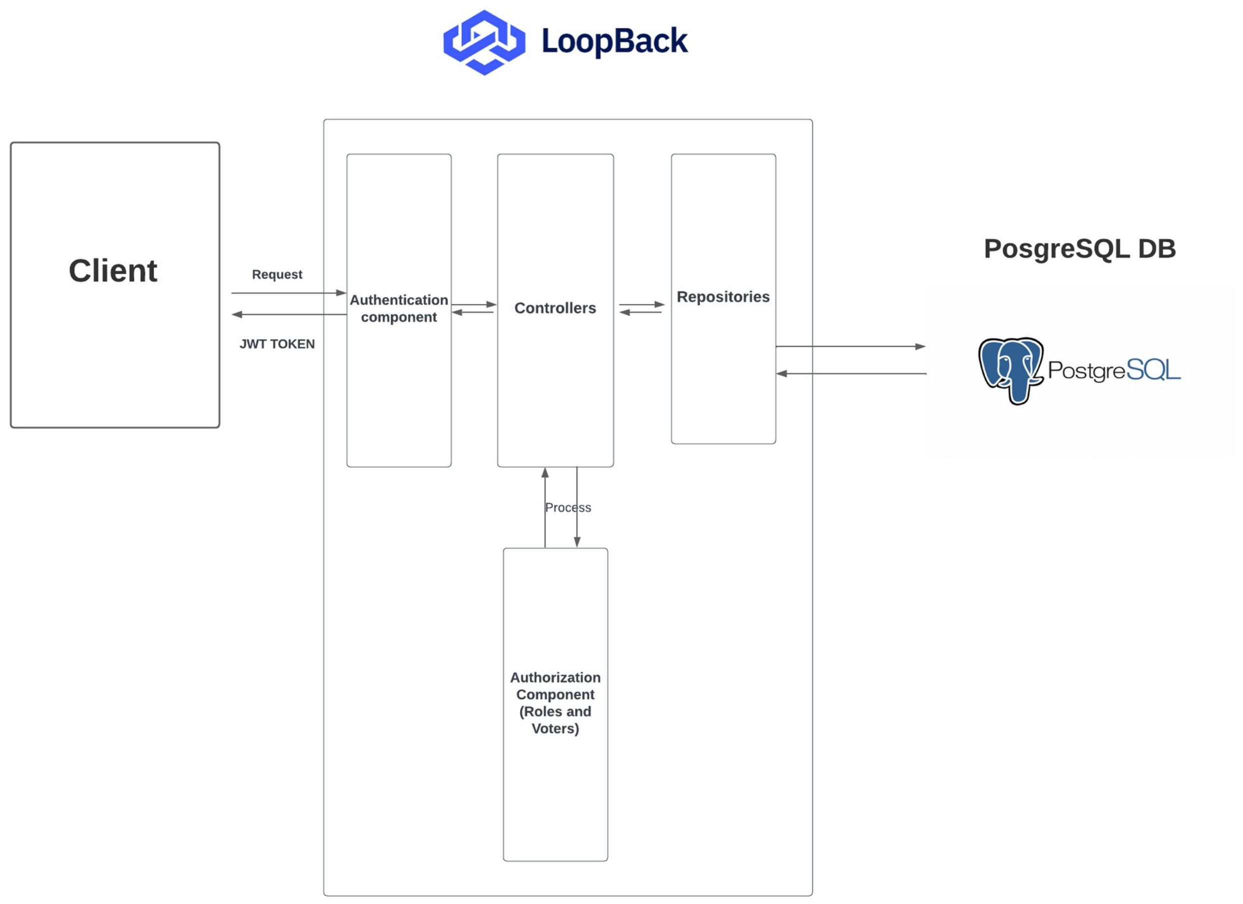
Documentation

I built an API for a library application based on the Loopback4 framework. The API has endpoints to allow users to sign up, browse and search books, create personal reading lists, and add existing books as entries to their lists. The following is a diagram of the app architecture.



This entails many security risks that I must tackle. The following is a list of the most impeding security risks, and the measures I have taken in mitigating them.

Threat model and Implemented Security Measures

Backend

1. **Authorization and Access Control:**
    - **Threat**: Unauthorized users gaining access to sensitive actions or other users' data.
    - **Vulnerability**: Insufficient privilege separation, insufficient data protection.
    - **Mitigation**:
        - I am using JWT tokens for authentication and authorization.
        - I implemented "Voters" such as "OWN_LIST_ONLY" that check users' id using the JWT token to permit users to edit their own lists only and prevent them from accessing other users' data.
        - I also implemented a role-based privilege system to assign specific roles (e.g. regular user, admin) with corresponding permissions.
          Regular users have limited access to their own data (e.g, creating/ modifying/deleting their own lists), while admins have broader privileges.
        - I protected certain routes like creating and deleting books, allowing only admins to use them.

2. **Injection Attacks and Data Validation and Integrity** (SQL Injection, XSS):
    - **Threat**: SQL injection attacks can lead to unauthorized access or manipulation of the database. XSS attacks can compromise user sessions or steal sensitive information.
    - **Vulnerability**: Lack of input validation and sanitization.
    - **Mitigation**:
        - I implemented input validation using a package called Joi to create validation schemas for all user inputs, such as book and list names, descriptions and user and entry creation.
        - I implemented functions that ensure that only valid data is accepted so that only **existing** books can be added to **existing** lists by users who own them.
        - Additionally, I implemented checks to avoid creating duplicate entries, or multiple users with same email address or username.
        - For the ***/search*** endpoint, a title param is used in a query to search for books with similar title. Using Loopback4 functions, I'm using parametrized queries that would prevent SQL injection attack attempts through the "title" searched.
        - On the frontend, I would use packages such as DOMPurify to sanitize input as well.

3. Sensitive Data Exposure:

- **Threat**: Exposure of sensitive data such as passwords or personal information.
- **Vulnerability**: Weak encryption, improper storage of sensitive data.
- **Mitigation**:
    - I am encrypting passwords using *bcrypt*, a strong encryption algorithm
    - I do not store unencrypted passwords as normal text in the database
    - I would avoid exposing sensitive information in error messages or logs while error handling

4. Denial of Service (DoS) Attacks:

- **Threat**: Disruption of service availability due to overwhelming server resources.
- **Vulnerability**: Lack of rate limiting, vulnerability to brute force attacks.
- **Mitigation**:
    - I implemented rate limiting using a package called ***loopback-rate-limiting***
    - Further measures for the future would include : CAPTCHA, and other techniques to mitigate the risk of DoS attacks. Implementing account lockout mechanisms.

5. Third-party Dependencies:

- **Threat**: Security vulnerabilities in third-party libraries or dependencies.
- **Vulnerability**: Failure to keep dependencies up to date.

- **Mitigation**: I should regularly review and update third-party libraries and dependencies to address security vulnerabilities and only use reputable and well-maintained libraries from trusted sources.

6. Error Handling:

- **Threat**: Exposing sensitive information through error messages.
- **Vulnerability**: Inadequate error handling mechanisms.
- **Mitigation**:
    - I implemented proper error handling mechanisms to provide informative error messages to users.
    - I would avoid exposing stack traces or detailed error messages in production environments to prevent disclosure of sensitive information

7. Testing : I have also made sure to test the controllers and validation schemas thoroughly.

8. User Authentication and Authorization:

- **Threat**: Credential stuffing attacks, brute force attacks.
- **Vulnerability**: Weak password policies, lack of account lockout mechanisms.
- **Mitigation**:
  - I'm using a validation package called Joi to enforce strong password policies as a bare minimum
    To mitigate this further in the future I could enforce multi-factor authentication, and account lockout after multiple failed login attempts.

Important but not yet implemented :

9. Protection against Data Loss:

- **Threat**: Data loss due to accidental deletion or system failures.
- **Vulnerability**: Lack of backup and recovery mechanisms.
- **Mitigation**:
  - Implement backup and recovery mechanisms to protect against data loss.
  - Regularly backup the database and ensure secure storage of backups to prevent unauthorized access.

Frontend

My work for this project is limited to the backend. I'm aware of the security risks on the frontend side such as :

1. Cross-Site Request Forgery (CSRF):

- **Threat**: Unauthorized actions performed on behalf of authenticated users.
- **Vulnerability**: Lack of CSRF protection.
- **Mitigation**:
  - Implement CSRF tokens and verify requests to prevent attackers from tricking authenticated users into executing unintended actions.

2. Secure Communication:

- **Threat**: Unauthorized interception of data transmitted between client and server.
- **Vulnerability**: Transmission of data over unencrypted channels (HTTP).
- **Mitigation**:
  - Implement HTTPS to encrypt data transmitted between the client and server.
  - Avoid sending sensitive information, such as user credentials or session tokens, in plaintext over the network.