## AEON Bank Backend Engineer Take Home Assignment

## -by Chua Soon Ee

### Project Tech Stacks:

1. **Language and Framekwork**: Java 17, Spring Boot

2. **Database**: H2 (in memory for ease of run and test without the need to setup).

In staging and production, we can use other relational databases like MySQL, Postgres etc.

3**. Dependency Manager**: Apache Maven 3.9.8 (pom.xml)

4. **Cloud Infra & CI/CD tools**: Github, Github Actions, Docker, Docker Desktop 4.29.0 (local test), Dockerhub (image repo for local pull and run), AWS ECR & EKS (actual staging environment)

5. **Test Automation**: Junit5, Mockito (Coverage repositories and services layers unit tests)

6. **Operating Systems**: Linux & Windows

7. **Development Tools**: Spring Tool Suite with Lombok, Git, SourceTree, Postman

8. **Web Server**: Embedded Tomcat

### The assumptions that I followed when developing this library system:

1. Case insensitive is allowed for book's title, author and ISBN.

2. Since the requirements indicates that the ISBN number uniquely identifies a book in the following way:

- 2 books with the same title and same author but different ISBN numbers are considered as different books

- 2 books with the same ISBN numbers must have the same title and same author

Multiple copies of books with same ISBN number are allowed in the system

**Example, an actual book can have two ISBNs (10 and 13):**

Book Title: Your Invisible Power

Book Author: Genevieve Behrend

Book ISBN-10: 1603865136

Book ISBN-13: 978-1603865135

This library system is designed with two books with same title and author but different ISBN provided (regardless of ISBN-10 and ISBN-13), they are considered as different books.

3. Since there is no requirement saying must implement security or JWT token with different roles (admin and member), I assumed there is no need to implement for API calls. By right in real world example, we need to have two user roles. Admin and member must login to the system to perform book loan actions (borrow and return).

4. The provided ISBN with - hyphen and spaces will be removed to standardize the ISBN format in this library system.

5. This project can be deployed to Docker Desktop and AWS EKS. Tested working in actual environments with the correct settings describe in later sections below.

6. Swagger API doc is disabled for Production environment.

### Build and run from local Dev IDE:

1. Unzip the library-system.zip project

2. Open with Java IDE (Spring Tool Suite, IntelliJ, Eclipse etc)

3. Import Project to your IDE and *\*set VM arguments: -Dspring.profiles.active=dev*

3. Build and run the project.

4. Access API documentation using this url:

http://localhost:8080/library-system/swagger-ui/index.html

### Build and run on local Docker Desktop (Windows):

1. Start Docker Desktop

2. Maven build:

> mvn clean install

3. To build docker image:

> docker build -f Dockerfile-dev -t library-system .

4. To create and run a new container from the created docker image on local Docker Desktop:

> docker run -p 8080:8080 library-system

5. To list the running containers:

> docker ps

6. To SSH into the running container:

> docker exec -it <CONTAINER\_ID\_FROM\_STEP\_5> sh

7. Access API documentation using this url:

<http://localhost:8080/library-system/swagger-ui/index.html>

8. To stop the running container:

> docker stop <CONTAINER\_ID\_FROM\_STEP\_4>

### Access to H2 Database console:

1. URL: http://localhost:8080/library-system/h2-console

2. JDBC URL: jdbc:h2:mem:aeonbank

2. Username: sa

3. Password: password

### Docker Hub Account (for docker image repo):

1. Create access token

Docker Hub Account -> Personal access token -> create access token: <https://app.docker.com/settings/personal-access-tokens>

* Name: github-access-token
* Expiration Date: 30 days
* Access permission: Read-Only
* Copy the generated access token and set to your Github Repository Secrets for DOCKER\_PASSWORD.

1. Create your Docker Hub repositories (https://hub.docker.com/repositories):

* Create a repository -> repo name: staging-library-system
* Create a repository -> repo name: production-library-system

### Github Account (configuration for code repo and ci/cd):

1. Create a Github Account and create a repository named ***library-system***.
2. Login to Github, click Settings -> Secrets and variables -> Actions

A black text on a white background

AI-generated content may be incorrect.

1. Create all required secrets for sensitive data:

A screenshot of a computer

AI-generated content may be incorrect.

Github Account -> Settings -> Secrets and variables -> Actions:

https://github.com/<username>/library-system/settings/secrets/actions

**Environment secrets:**

1. DB\_USERNAME: sa

2. DB\_PASSWORD: password

**Repository secrets:**

1. DOCKER\_USERNAME

2. DOCKER\_PASSWORD = docker hub access token (generate one if not yet generated before)

3. DOCKER\_USERNAME (your Docker hub username)

4. DOCKER\_REPO\_STAGING (value similar as Docker Hub repo name created above e.g. staging-library-system)

5. AWS\_ACCESS\_KEY\_ID (a temporary access key id to be used)

6. AWS\_SECRET\_ACCESS\_KEY (a temporary secret access key to be used)

7. ECR\_REPO (AWS ECR repository without tag)

8. ECR\_REPO\_STAGING (AWS ECR repository with tag)

9. EKS\_CLUSTER\_STAGING (AWS EKS Cluster name for staging namespace)

### Push Code to Github repository

1. Open command prompt and change directory to library-system project directory.
2. Run:
   1. git init
   2. git add –all
   3. git commit -m "added initial codebase"
   4. git branch -M main
   5. git remote add origin https://github.com/<YOURUSERNAME>/library-system.git
   6. git push -u origin main
3. Create staging branch for staging deployment.

### Github Actions

After code push to Github, Github action workflow files will be triggered and run the jobs/steps based on branch (staging and main) to:

*\*main = production*

CI/CD Workflow files under library-system/.github/workflows:

1. **Docker Hub CI/CD & Local Deploy** (ci-cd-docker-hub.yml)
   1. After job completed successfully, you can docker pull the image from your Docker Hub repository and run locally

> docker pull chuasoonee/staging-library-system:latest

> docker run -d -p 8080:8080 chuasoonee/staging-library-system:latest

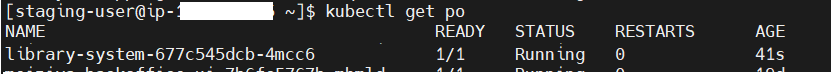
* 1. Access <http://localhost:8080/library-system/swagger-ui/index.html>

1. **AWS ECR & EKS CI/CD** (ci-cd-aws-ecr-eks.yml)
   1. Must have AWS EKS cluster created
   2. Create namespace staging and prod
   3. End to end build, test, push docker image to AWS ECR, deploy to AWS EKS.

> kubectl get svc

> kubectl get po

Verify the pod is running



## How to use the APIs:

1. **Access to Swagger API Docs**:

<http://localhost:8080/library-system/swagger-ui/index.html>

1. **Register borrowers:**

Expand POST /borrowers

Click "Try it out" button

Replace Request body with the following and click Execute button to register a borrower:

{

"name": "Adrian Tee",

"email": "adrian.tee@aeonbank.com"

}

Add more borrowers:

{

"name": "Alex Pee",

"email": "alex.pee@merz.com"

}

1. **Register books:**

Expand POST /books

Click "Try it out" button

\*\*NOTE: ISBN must be a valid one. Invalid ISBN is not allowed to be registered. Validation implemented.

\*\*ISBN will be normalized by replacing - (hyphen) and \\s (any whitespace character: space, tab, etc.)

Replace Request body with the following and click Execute button to register a book:

{

"isbn": "9780132350884",

"title": "Clean Code",

"author": "Robert Cecil Martin"

}

Click multiple times with the same request body allow adding the same book with different IDs.

Add more books:

{

"isbn": "978-0007179732",

"title": "The Monk Who Sold His Ferrari",

"author": "Robin Sharma"

}

1. **Get a list of all books in the library:**

EXPAND GET /books/all

Click "Try it out" button

Click Execute button to list all books.

1. **Get a paginated list of books in the library:**

EXPAND GET /books

Click "Try it out" button

Replace pageable with the following to sort the list by title with descending order:

\*\*you can filter by isbn, title, author and loanStatus.

\*\*you can sort by isbn, title and author.

{

"page": 0,

"size": 10,

"sort": "title,desc"

}

Click Execute button to list all books.

1. **Borrow a book:**

EXPAND POST /loans

Click "Try it out" button

Make sure to set valid bookId and borrowerId. Invalid bookId or borrowerId will throw exception.

Book already on loan cannot be borrowed by other at the same time.

A borrower can borrow multiple books.

Replace Request body with the following and click Execute button to borrow a book:

{

"bookId": 1,

"borrowerId": 1

}

Borrow more books:

{

"bookId": 2,

"borrowerId": 2

}

{

"bookId": 3,

"borrowerId": 1

}

Click Execute button to to borrow book.

1. **Return a borrowed book on behalf of a borrower:**

EXPAND PATCH /loans/{loanId}/return  
Click "Try it out" button

Insert a valid loanId.

Click Execute button to return a loaned book.

1. **Get a list of loans records**

EXPAND GET /loans

Click "Try it out" button

Replace pageable with the following:

{

"page": 0,

"size": 10

}

Click Execute button to list all loan records.

\*\*You can filter by:

loanStatus (unselected will list all loan records)

borrowerName (partial value)

borrowerEmail (exact value)

### 12factors compliances checklist:

1. **Codebase**: git repository - github

2. **Dependencies**: Maven as dependency management (pom.xml)

3. **Config**: Avoid .properties/.yaml for sensitive data like password, secret key. etc. Credentials are injected via environment variables on Github Secrets and variables (Environment secrets and Repository secrets), and Kubernetes (AWS EKS) secrets.

4. **Backing Services**: H2 for demo, MySQL or any other relational database for staging & production, abstracted via Spring Data JPA.

5. **Build, Release and Run**: Docker to build -> Test -> release -> run separation.

6. **Processes**: Runs as stateless services with RESTful APIs, adhering to microservices principles.

7. **Port Binding**: Exposes HTTP APIs via Spring Boot, mapped dynamically via environment variables (EXPOSE 8080)

8. **Concurrency**: Can scale horizontally by deploying multiple instances (Docker + Kubernetes ready).

9. **Disposability**: Fast startup and graceful shutdown via Spring Boot’s lifecycle.

10. \***Dev/Prod Parity**: Using H2 in dev for ease of testing purpose. In actual staging & production can use MySQL or any other relational database.

11. \***Logs**: Currently logs to console only—consider using structured logging (ELK, CloudWatch) for better observability.

12. Admin Processes: Supports migrations and database seeding via SQL scripts.

*\*means partially complied.*