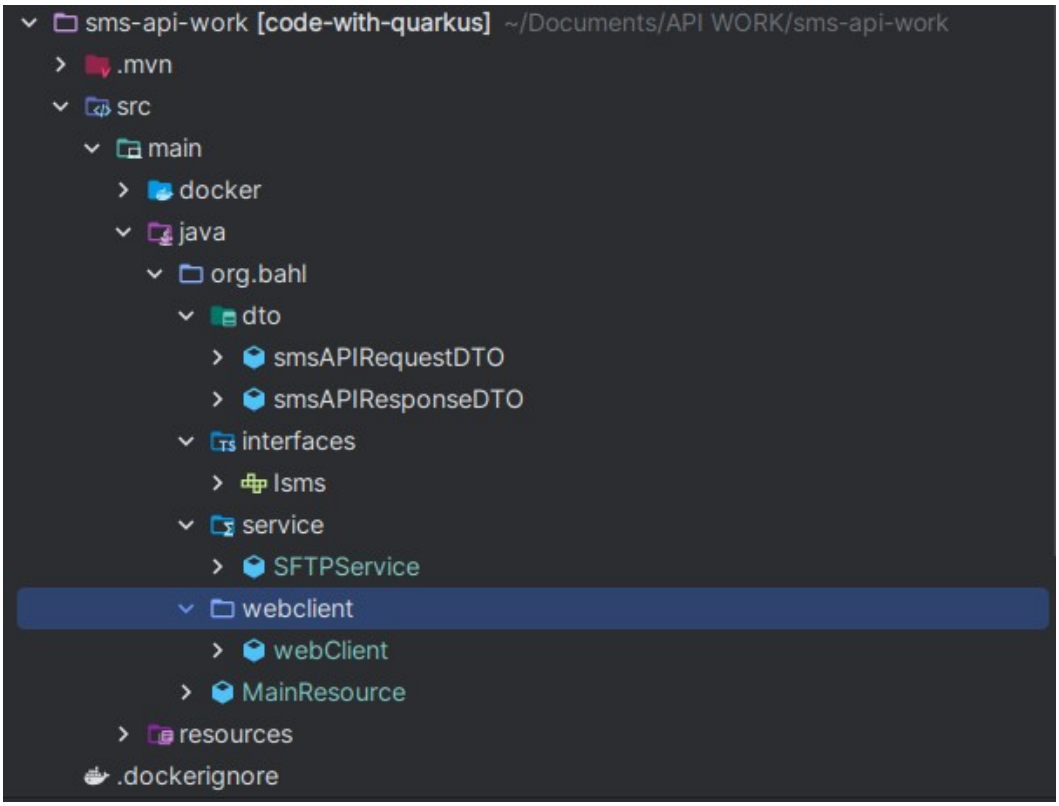
Library Management System

To design and develop a RESTful **Library Management System** using Java and Quarkus. This project simulates real-world features such as **book lending**, **member management**, and **availability tracking**, following **clean architecture principles** — using **DTOs(Data Transfer Objects)**, **controllers**, **interfaces**, and **service implementations**. No Database is involved consume public API for this purpose and just simulate it in-memory java collections, hashmaps, Maps and Lists.

Project Goals:

- Build a clean, scalable Quarkus application.
- Implement separation of concerns: Controller → Service → Interface → DTO.
- Expose complete CRUD operations using REST APIs.
- Use OpenAPI/Swagger for documentation.
- Simulate business logic like book checkout, return, availability.
- Add caching, Add logging (Quarkus logger), Add request validation using `@Valid`, `Export lendings to CSV` and scheduled tasks.

Project Structure:



src/main/java/com/library

```
├── controller      → REST endpoints (resources)
├── service         → Interfaces
├── service/impl    → Implementations
├── dto             → All DTOs
├── model           → Internal models (if needed)
├── exception       → Custom exceptions
└── utils           → Common helpers
```

CRUD Operation	*Endpoint*	*Description*
Get	/books	Get all books
POST	/books	Add new book
PUT	*/books/{id}*	Update book info
DELETE	*/books*{id}	Delete a book
GET	/members	Get all members
POST	/members	Add new member
POST	*/lending*/returns/{id}	Mark book as return
GET	*/lending*/history	Show all lending books data with member(history)
POST	/lending	Lend a book to member

Business Logic Rules:

- A book can only be lent if it's marked as **available**.
- When a book is lent, mark it as **unavailable**.
- On return, mark the book as **available again**.
- Handle edge cases like lending a non-existent book or returning an already returned one.

Entities & DTOs:

Use DTOs for all input/output. Do not expose internal models directly.

```
public class BookDTO {
   public String id;
   public String title;
   public String author;
   public boolean available;
}
```

```
public class MemberDTO {
   public String id;
   public String name;
   public String email;
}
```

```
public class LendingDTO {

   public String lendingId;
   public String bookId;
   public String memberId;
   public String lendingDate;
   public String returnDate; //
nullable
}
```

Swagger Configuration:

Ensure Swagger UI is available at `/q/swagger-ui` using Quarkus OpenAPI extension:

quarkus-extension add 'quarkus-smallrye-openapi'

Modules & Responsibilities

1. Book Module

- Add new books
- View all books
- Update book info
- Delete books
- Track availability

2. Member Module

- Register new members
- View all registered members

3. Lending Module

- Lend a book to a member (only if book is available)
- Return a book (and mark it available again)
- View lending history (optional)

Development Tasks

Task 1: Initialize Project

- Use https://code.quarkus.io to generate a Quarkus project.
- Add the following extensions:
    - RESTEasy Reactive
    - Swagger/OpenAPI
    - Hibernate Validator (for input validation)
    - Jackson or JSON-B (for JSON handling)

Task 2: Create DTOs

- HINT: DTOs are just plain Java classes with fields and (optionally) constructors or getters/setters.

Task 3: Build Controller Layer

- Add endpoints using `@Path`, `@POST`, `@GET`, `@PUT`, etc.
- HINT: Return `Response` objects with proper status codes like `Response.ok()`, `Response.status(404)`, etc.

Task 4: Build Service Layer

- Create interfaces (e.g., `BookService`) and implementations (`BookServiceImpl`)
- Implement logic to store and retrieve data from Java `List<BookDTO>` or `Map<String, BookDTO>`
- HINT: Use UUID for generating `id` values: `UUID.randomUUID().toString()`

Task 5: Exception Handling

- Create custom exceptions like `BookNotFoundException`, `AlreadyLentException`, etc.
- Use `@Provider` to create a global exception mapper if needed.

Task 6: Add Swagger

- Add Quarkus OpenAPI extension
- Document each endpoint using `@Operation`, `@APIResponse`, etc.
- HINT: Access Swagger at: `http://localhost:8080/q/swagger-ui`

Task 7: Add Book Lending Logic

- On lending, mark book as `available = false`
- On return, mark book as `available = true`
- Validate if book exists and is available before lending
- Store lending records in memory

Deliverable:

- Fully functional Quarkus application
- In-memory simulation of book/member/lending records
- Clean separation of layers: DTOs, interfaces, controllers
- API documentation at `/q/swagger-ui`