Assignment 1 Documentation

Iulia-Olivia Cipleu, Group 3

March 2024

1 Aim of Project

The scope of this project is to establish a database consisting of a minimum of two tables that are in a One-To-Many relationship. The project will also involve creating CRUD operations for these tables and testing them using Postman, which is a popular API client that makes it easy for developers to create, share, test, and document APIs.
For this project, I have chosen to develop a Library Book Return Management System using Java Spring and MySQL.

2 Database Description

The database is composed of four tables: Book, Author, Borrow, and Borrower. Each book is associated with a single author, but an author may have written multiple books. The Borrow table represents the act of borrowing a book and records the details of one book and one borrower at a time. A borrower can check out several books, and over time, a book may be borrowed by numerous individuals.

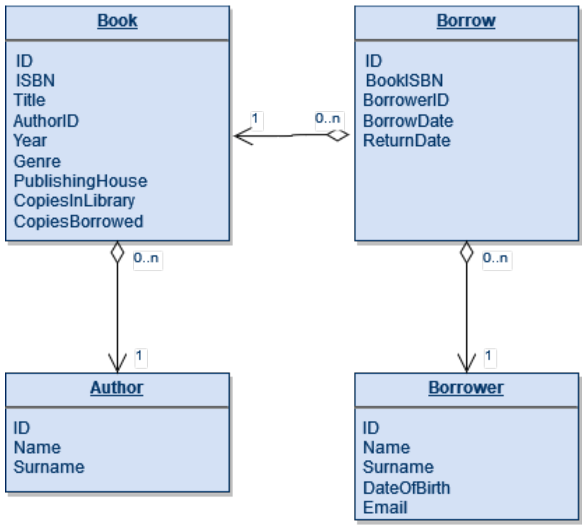

Figure 1: Database Schema Part 1

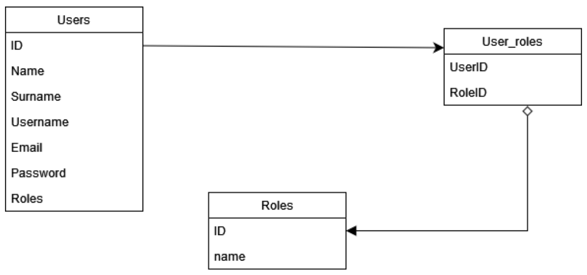

Figure 2: Database Schema Part 2

3 Packages

3.1 Model

This package encapsulates the database tables. It implements the classes Book, Borrow, Borrower, and Author, each with their specific fields, all the information possible. Also, there is a Enum Genre, used by Book, but not existing as a table in database.

3.2 DTO

DTO, or Data Transfer Object, is a design pattern used in software development. It is an object that carries data between processes. In the context of the project, DTO classes are simplified versions of the model classes. They contain only the necessary fields that are required for specific operations, which can help improve performance by reducing the amount of data that needs to be transferred, especially in network-related operations. It is a way to tailor the data to the exact needs of the client. This can be particularly useful in scenarios where the model classes have many fields, but only a subset of them are used in most operations.

3.3 Repository

This package contains only interfaces, one for each class in model, which has the CRUD operations declared.

3.4 Mapper

This package houses classes that include converter methods, which facilitate the transformation from entity (as defined in the Model) to DTO and vice versa. These methods play a crucial role in converting data from JSON format to a format suitable for the database.

3.5 Service

This is the package which actually performs the CRUD operations: Create, Update, Read, Delete, using the classes from Repository and Mapper. Here lies the actual logic of the application.

3.6 Controller

As Service, contain the core logic, but this time it also provides the URLs and creates the bridge between the database and JSON. Each method specifies if there are needed path variables and body. Broadly speaking, the JSON request in transformed in SQL syntax, and the result is again converted in JSON.

3.7 Validators

This packages defines different validators for the input, like: email format, dates to not be in the future, retun date after borrow date, positive numbers etc.

3.8 Exceptions

This class was created to facilitate the understanding of some error, like those throw by the validator, or other cases (trying to delete a non-existing item).

4 UML Class Diagram

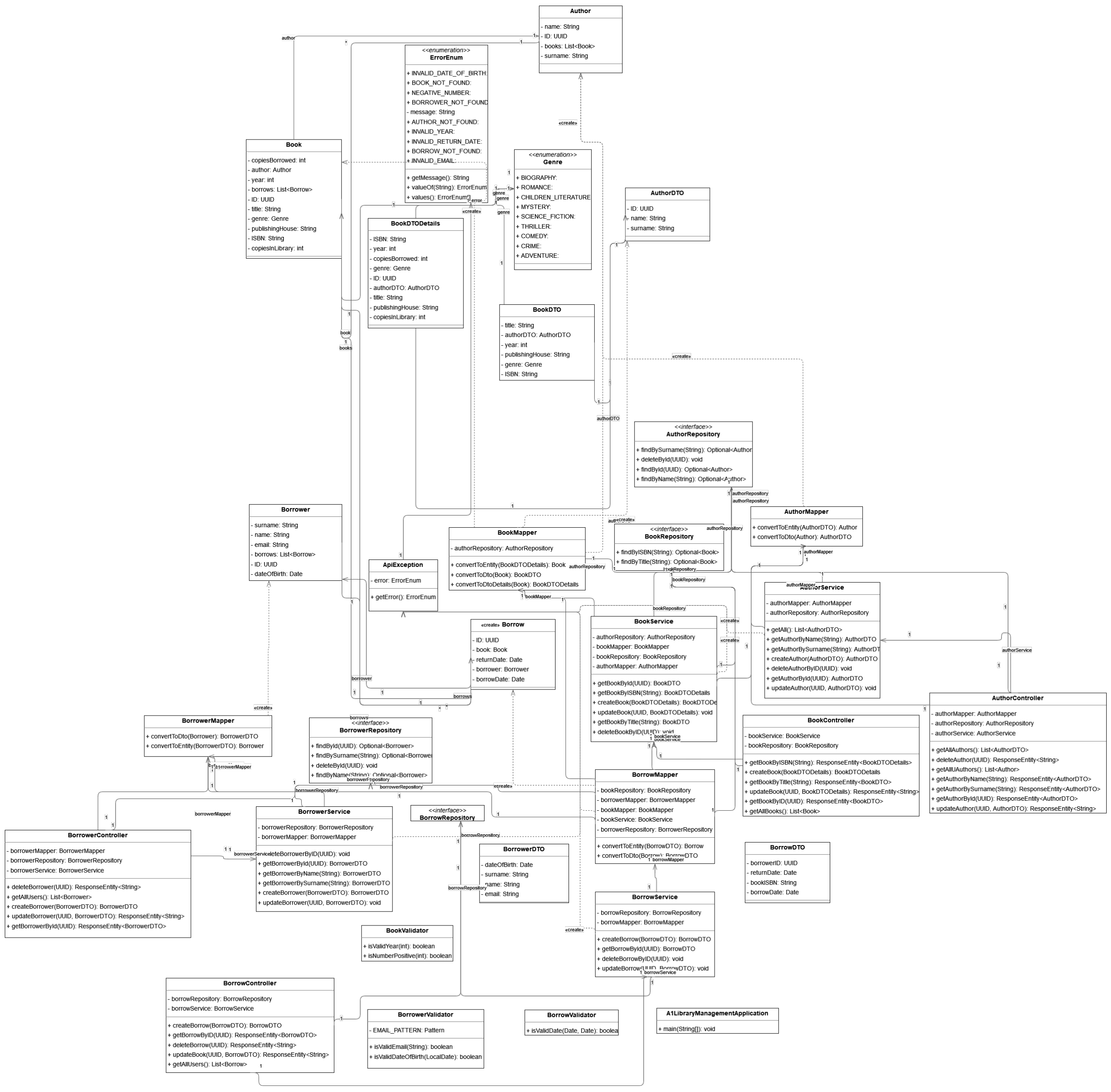
Author
- name: String
- ID: UUID
- books: List<Book>
- surname: String
<<enumeration>> ErrorEnum
+ INVALID_DATE_OF_BIRTH:
+ BOOK_NOT_FOUND:
+ NEGATIVE_NUMBER:
+ BORROWER_NOT_FOUND:
- message: String
+ AUTHOR_NOT_FOUND:
+ INVALID_YEAR:
+ INVALID_RETURN_DATE:
+ BORROW_NOT_FOUND:
+ INVALID_EMAIL:
+ getMessage(): String
+ valueOf(String): ErrorEnum
+ values(): ErrorEnum[]
Book
- copiesBorrowed: int
- author: Author
- year: int
- borrows: List<Borrow>
- ID: UUID
- title: String
- genre: Genre
- publishingHouse: String
- ISBN: String
- copiesInLibrary: int
<<enumeration>> Genre
+ BIOGRAPHY:
+ ROMANCE:
+ CHILDREN_LITERATURE:
+ MYSTERY:
+ SCIENCE_FICTION:
+ THRILLER:
+ COMEDY:
+ CRIME:
+ ADVENTURE:
AuthorDTO
- ID: UUID
- name: String
- surname: String
BookDTODetails
- ISBN: String
- year: int
- copiesBorrowed: int
- genre: Genre
- ID: UUID
- authorDTO: AuthorDTO
- title: String
- publishingHouse: String
- copiesInLibrary: int
BookDTO
- title: String
- authorDTO: AuthorDTO
- year: int
- publishingHouse: String
- genre: Genre
- ISBN: String
<<interface>> AuthorRepository
+ findBySurname(String): Optional<Author>
+ deleteById(UUID): void
+ findById(UUID): Optional<Author>
+ findByName(String): Optional<Author>
Borrower
- surname: String
- name: String
- email: String
- borrows: List<Borrow>
- ID: UUID
- dateOfBirth: Date
BookMapper
- authorRepository: AuthorRepository
+ convertToEntity(BookDTODetails): Book
+ convertToDto(Book): BookDTO
+ convertToDtoDetails(Book): BookDTODetails
<<interface>> BookRepository
+ findByISBN(String): Optional<Book>
+ findByTitle(String): Optional<Book>
AuthorMapper
+ convertToEntity(AuthorDTO): Author
+ convertToDto(Author): AuthorDTO
ApiException
- error: ErrorEnum
+ getError(): ErrorEnum
AuthorService
- authorMapper: AuthorMapper
- authorRepository: AuthorRepository
+ getAll(): List<AuthorDTO>
+ getAuthorByName(String): AuthorDTO
+ getAuthorBySurname(String): AuthorDTO
+ createAuthor(AuthorDTO): AuthorDTO
+ deleteAuthorByID(UUID): void
+ getAuthorById(UUID): AuthorDTO
+ updateAuthor(UUID, AuthorDTO): void
«create» Borrow
- ID: UUID
- book: Book
- returnDate: Date
- borrower: Borrower
- borrowDate: Date
BookService
- authorRepository: AuthorRepository
- bookMapper: BookMapper
- bookRepository: BookRepository
- authorMapper: AuthorMapper
+ getBookById(UUID): BookDTO
+ getBookByISBN(String): BookDTODetails
+ createBook(UUID, BookDTODetails): BookDTODetails
+ updateBook(UUID, BookDTODetails): void
+ getBookByTitle(String): BookDTO
+ deleteBookById(UUID): void
AuthorController
- authorMapper: AuthorMapper
- authorRepository: AuthorRepository
- authorService: AuthorService
+ getAllAuthors(): List<AuthorDTO>
+ deleteAuthor(UUID): ResponseEntity<String>
+ getAllUsers(): List<Author>
+ getAuthorByName(String): ResponseEntity<AuthorDTO>
+ getAuthorBySurname(String): ResponseEntity<AuthorDTO>
+ getAuthorById(UUID): ResponseEntity<AuthorDTO>
+ updateAuthor(UUID, AuthorDTO): ResponseEntity<String>
BookController
- bookService: BookService
- bookRepository: BookRepository
+ getBookByISBN(String): ResponseEntity<BookDTODetails>
+ createBook(BookDTODetails): BookDTODetails
+ getBookByTitle(String): ResponseEntity<BookDTO>
+ updateBook(UUID, BookDTODetails): ResponseEntity<String>
+ getBookByID(UUID): ResponseEntity<BookDTO>
+ getAllBooks(): List<Book>
BorrowerMapper
+ convertToDto(Borrower): BorrowerDTO
+ convertToEntity(BorrowerDTO): Borrower
<<interface>> BorrowerRepository
+ findById(UUID): Optional<Borrower>
+ findBySurname(String): Optional<Borrower>
+ deleteById(UUID): void
+ findByName(String): Optional<Borrower>
BorrowMapper
- bookRepository: BookRepository
- borrowerMapper: BorrowerMapper
- bookMapper: BookMapper
- bookService: BookService
- borrowerRepository: BorrowerRepository
+ convertToEntity(BorrowDTO): Borrow
+ convertToDto(Borrow): BorrowDTO
BorrowerController
- borrowerMapper: BorrowerMapper
- borrowerRepository: BorrowerRepository
- borrowerService: BorrowerService
+ deleteBorrower(UUID): ResponseEntity<String>
+ getAllUsers(): List<Borrower>
+ createBorrower(BorrowerDTO): BorrowerDTO
+ updateBorrower(UUID, BorrowerDTO): ResponseEntity<String>
+ getBorrowerById(UUID): ResponseEntity<BorrowerDTO>
BorrowerService
- borrowerRepository: BorrowerRepository
- borrowerMapper: BorrowerMapper
+ deleteBorrowerByID(UUID): void
+ getBorrowerByID(UUID): BorrowerDTO
+ getBorrowerByName(String): BorrowerDTO
+ getBorrowerBySurname(String): BorrowerDTO
+ createBorrower(BorrowerDTO): BorrowerDTO
+ updateBorrower(UUID, BorrowerDTO): void
<<interface>> BorrowRepository
BorrowerDTO
- dateOfBirth: Date
- surname: String
- name: String
- email: String
BorrowDTO
- borrowerID: UUID
- returnDate: Date
- bookISBN: String
- borrowDate: Date
BorrowService
- borrowRepository: BorrowRepository
- borrowMapper: BorrowMapper
+ createBorrow(BorrowDTO): BorrowDTO
+ getBorrowById(UUID): BorrowDTO
+ deleteBorrowById(UUID): void
+ updateBorrow(UUID, BorrowDTO): void
BookValidator
+ isValidYear(int): boolean
+ isNumberPositive(int): boolean
BorrowController
- borrowRepository: BorrowRepository
- borrowService: BorrowService
+ createBorrow(BorrowDTO): BorrowDTO
+ getBorrowByID(UUID): ResponseEntity<BorrowDTO>
+ deleteBorrow(UUID): ResponseEntity<String>
+ updateBook(UUID, BorrowDTO): ResponseEntity<String>
+ getAllUsers(): List<Borrow>
BorrowerValidator
- EMAIL_PATTERN: Pattern
+ isValidEmail(String): boolean
+ isValidDateOfBirth(LocalDate): boolean
BorrowValidator
+ isValidDate(Date, Date): boolean
A1LibraryManagementApplication
+ main(String[]): void

Figure 3: UML Diagram

The diagram can be found in a separate PNG.

5 Use Case Diagram

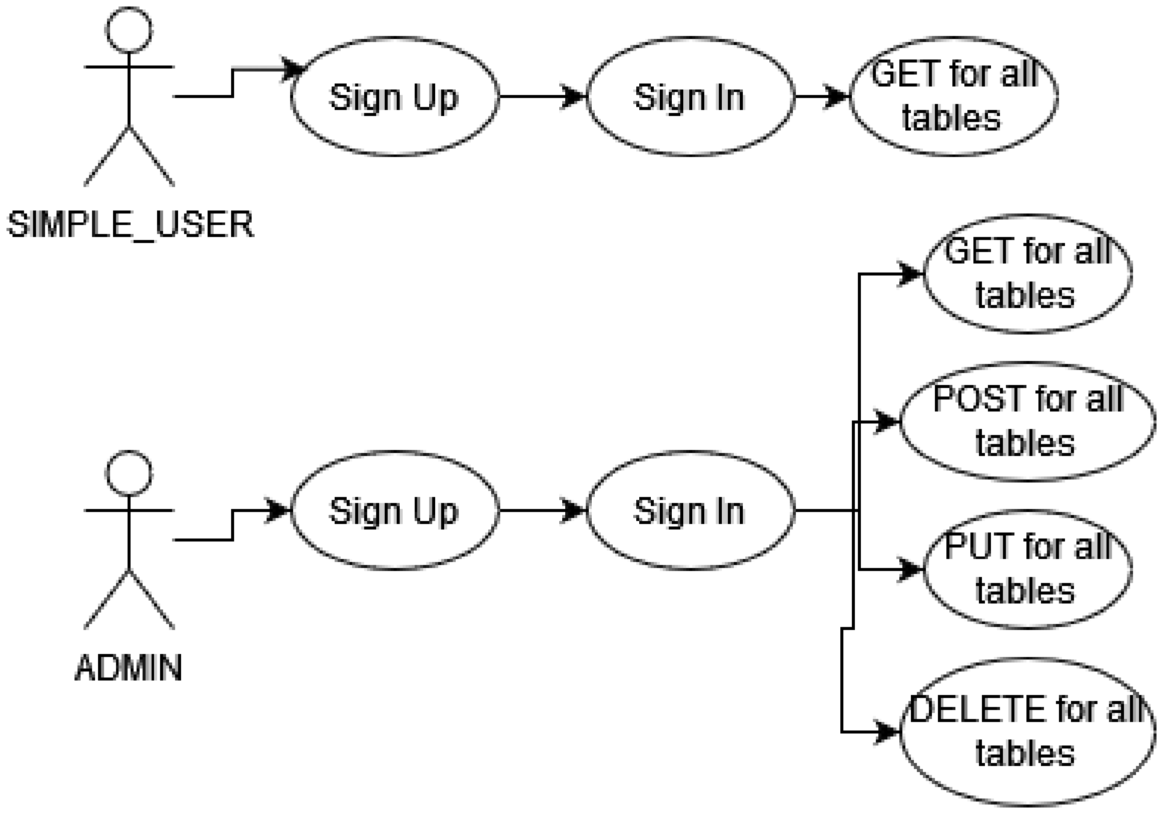

Figure 4: Use Case Diagram