Project 7

Book management system

Congke Zhao & Rui Zhang

Statement:

We were affected by COVID-19 in the process of doing the project, which caused many delays in the project's production. Our project ended up not being complicated, very different from project 5 and project 6, but still finished well, with a front-end interface and a back-end database while handling data exchange between the front-end and back-end. At the beginning of project 5 design, we had a complete plan, but in the implementation phase, we took some detours, which we only noticed after project 6. We overestimated our capabilities, especially in database connectivity; we encountered problems we couldn't solve. Also, there were problems with the front-end interface, so we modified our thinking. This time, the code was completely different from project 6. We gave up designing the page directly through java. We used HTML to create a web application and used the h2 database. h2 database is an open-source embedded database engine that is written in java language and is platform-independent. Also, the H2 database provides a very convenient web console for manipulating and managing the database content. The final design just allows the librarian to manage the books and removes the user login part because we think the program is only for the librarian, and there is no login interface. We initially planned to use Navicat as the running tool for MySQL and tomcat as the server proxy. However, since there was a conflict between the configuration of Rui's IOS computer environment and congke's Windows environment, and there was an unsolvable problem with the configuration of Rui's IOS computer environment, we abandoned this plan and used the h2 database directly.

Compared to projects 5 and 6, we ditched almost all of our code and tried to use maven for management while introducing the spring boot framework, which helped improve our efficiency due to time constraints. Due to the huge changes, there is also a big change in the design patterns. Some design patterns come from the bottom of the framework and are very easy to use after understanding them. And try to use the design at first did not think of the pattern.

UML:

Design pattern

MVC design pattern for the whole app.

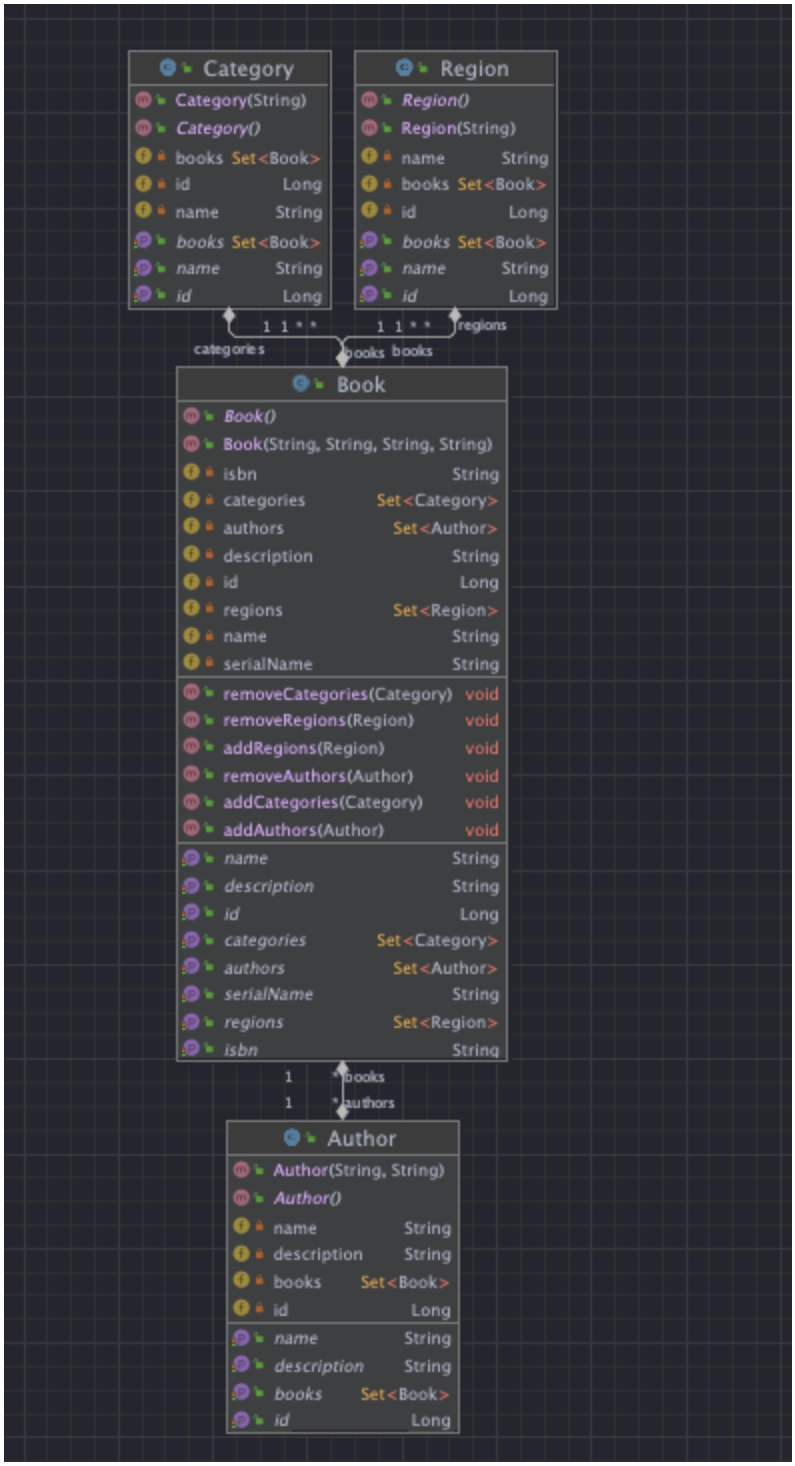
- Controller in MVC, based on HTTP request to call/ invoke model to handle CRUD on my database. Models include service, entity, and repository. Models resend the messages back to the Controller after the Controller gets the data from the backend and passes it to the front end, which is HTML will demonstrate the data.

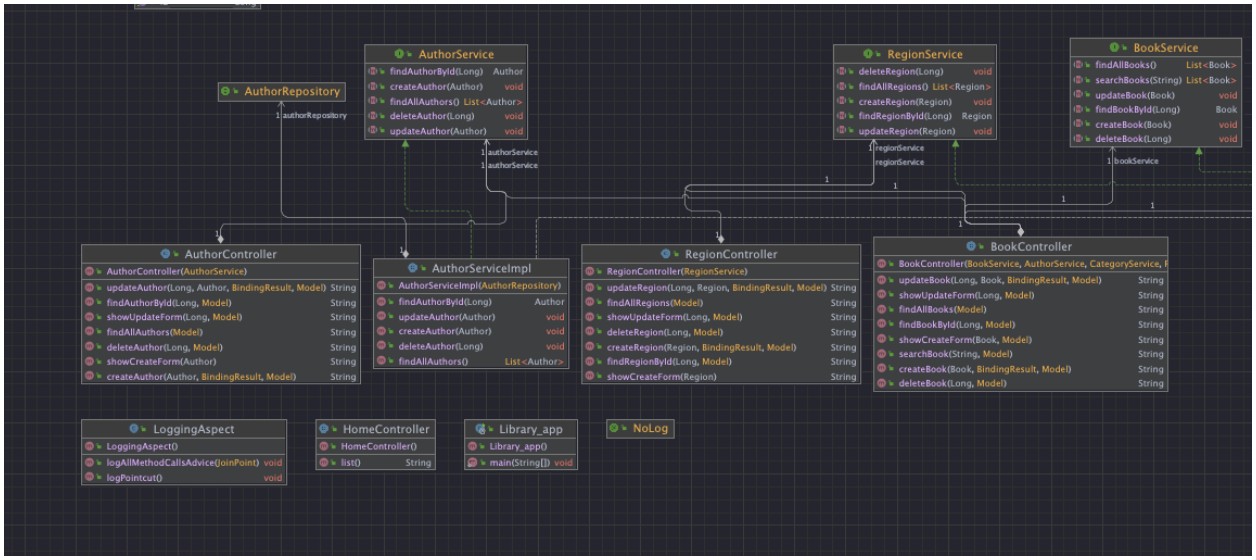
The whole project was based on the spring boot basic spring framework. There are factory and singleton patterns inside this framework.

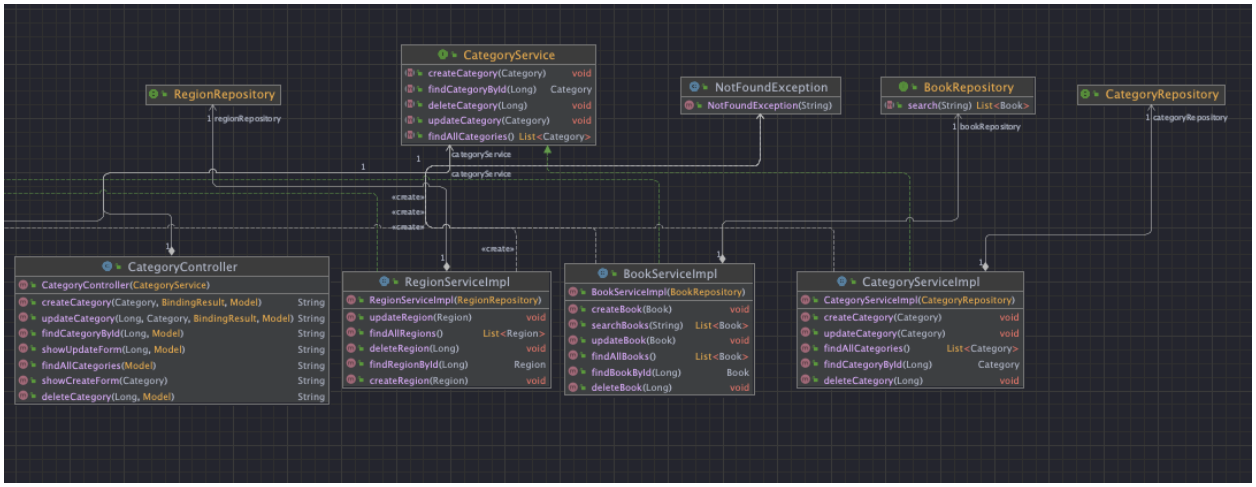
- For singleton, all the Services, controllers, and repositories use those annotations, and Spring framework will generate only one instance for them.
- Factory pattern, Inside the spring framework, there is a factory class to generate instances that I request as controllers; the factory pattern will generate those instances of classes.

Using the façade design pattern as SLF4j, this slf4j façade replaces the default logging framework, which is logback with log4j2. Insert other sets of APIs for future use.

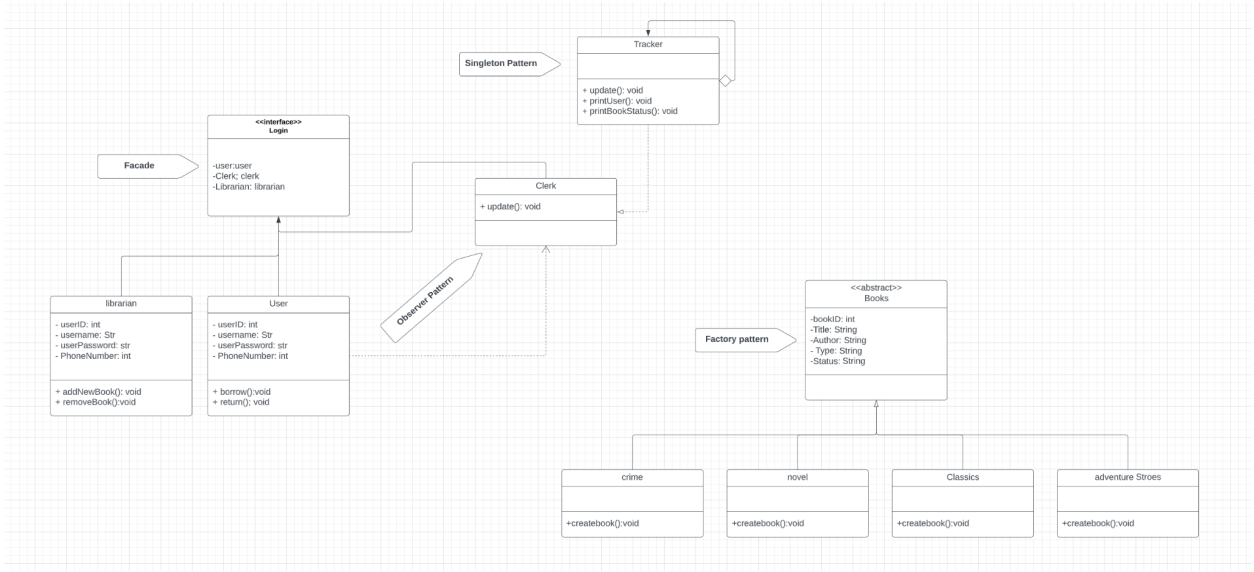
Proxy pattern, Spring framework's dynamic proxy tech, so I can add a log before every method runs. (Logging aspect)


Category
Category(String)
Category()
books Set<Book>
id Long
name String
books Set<Book>
name String
id Long
Region
Region()
Region(String)
name String
books Set<Book>
id Long
books Set<Book>
name String
id Long
1 1 * *
categories
books
1 1 * *
regions
books
Book
Book()
Book(String, String, String, String)
isbn String
categories Set<Category>
authors Set<Author>
description String
id Long
regions Set<Region>
name String
serialName String
removeCategories(Category) void
removeRegions(Region) void
addRegions(Region) void
removeAuthors(Author) void
addCategories(Category) void
addAuthors(Author) void
name String
description String
id Long
categories Set<Category>
authors Set<Author>
serialName String
regions Set<Region>
isbn String
1
*
books
1
*
authors
Author
Author(String, String)
Author()
name String
description String
books Set<Book>
id Long
name String
description String
books Set<Book>
id Long


AuthorService
findAuthorById(Long) Author
createAuthor(Author) void
findAllAuthors() List<Author>
deleteAuthor(Long) void
updateAuthor(Author) void
AuthorRepository
RegionService
deleteRegion(Long) void
findAllRegions() List<Region>
createRegion(Region) void
findRegionById(Long) Region
updateRegion(Region) void
BookService
findAllBooks() List<Book>
searchBooks(String) List<Book>
updateBook(Book) void
findBookById(Long) Book
createBook(Book) void
deleteBook(Long) void
AuthorController
AuthorController(AuthorService)
updateAuthor(Long, Author, BindingResult, Model) String
findAuthorById(Long, Model) String
showUpdateForm(Long, Model) String
findAllAuthors(Model) String
deleteAuthor(Long, Model) String
showCreateForm(Author) String
createAuthor(Author, BindingResult, Model) String
AuthorServiceImpl
AuthorServiceImpl(AuthorRepository)
findAuthorById(Long) Author
updateAuthor(Author) void
createAuthor(Author) void
deleteAuthor(Long) void
findAllAuthors() List<Author>
RegionController
RegionController(RegionService)
updateRegion(Long, Region, BindingResult, Model) String
findAllRegions(Model) String
showUpdateForm(Long, Model) String
deleteRegion(Long, Model) String
createRegion(Region, BindingResult, Model) String
findRegionById(Long, Model) String
showCreateForm(Region) String
BookController
BookController(BookService, AuthorService, CategoryService,
updateBook(Long, Book, BindingResult, Model) String
showUpdateForm(Long, Model) String
findAllBooks(Model) String
findBookById(Long, Model) String
showCreateForm(Book, Model) String
searchBook(String, Model) String
createBook(Book, BindingResult, Model) String
deleteBook(Long, Model) String
LoggingAspect
LoggingAspect()
logAllMethodCallsAdvice(JoinPoint) void
logPointcut() void
HomeController
HomeController()
list() String
Library_app
Library_app()
main(String[]) void
NoLog
authorRepository
authorService
regionService
bookService


CategoryService
createCategory(Category) void
findCategoryById(Long) Category
deleteCategory(Long) void
updateCategory(Category) void
findAllCategories() List<Category>
RegionRepository
NotFoundException
NotFoundException(String)
BookRepository
search(String) List<Book>
CategoryRepository
regionRepository
bookRepository
categoryRepository
categoryService
«create»
CategoryController
CategoryController(CategoryService)
createCategory(Category, BindingResult, Model) String
updateCategory(Long, Category, BindingResult, Model) String
findCategoryById(Long, Model) String
showUpdateForm(Long, Model) String
findAllCategories(Model) String
showCreateForm(Category) String
deleteCategory(Long, Model) String
RegionServiceImpl
RegionServiceImpl(RegionRepository)
updateRegion(Region) void
findAllRegions() List<Region>
deleteRegion(Long) void
findRegionById(Long) Region
createRegion(Region) void
BookServiceImpl
BookServiceImpl(BookRepository)
createBook(Book) void
searchBooks(String) List<Book>
updateBook(Book) void
findAllBooks() List<Book>
findBookById(Long) Book
deleteBook(Long) void
CategoryServiceImpl
CategoryServiceImpl(CategoryRepository)
createCategory(Category) void
updateCategory(Category) void
findAllCategories() List<Category>
findCategoryById(Long) Category
deleteCategory(Long) void

Project 5 UML:



The most crucial change is the use of patterns. In Project 5, we intended to use the Java processing front end directly but found that using HTML was more flexible and allowed the code to be encapsulated better. Therefore, we should use HTML as the front end and the simpler bootstrap for editing. At the same time, the framework is introduced, which we should have considered before, but because the framework can help us save a lot of time, such as processing HTTP requests, and it is more convenient and direct in-database processing. We have the problem that we cannot connect to the database, and we have tried many different methods to solve it, which is the reason why we finally changed to H2database. There are also many changes in the design pattern. Due to the solid H2 database, we found that many design patterns were not easy to be compatible during the learning process, mainly due to the lack of ability and deep understanding.

Most of the **Third parties code** that we use is the configuration problem of maven. To better use the maven and spring boot framework, there are many dependencies that need to be added by ourselves, so we copied a lot of configuration files.
The database connection is learned from:

https://stackoverflow.com/questions/38660104/how-to-connect-intellij-withlocal-mysql
https://www.javatpoint.com/example-to-connect-to-the-mysql-database
https://blog.csdn.net/kongsanjin/article/details/96425826
Java swing (interface):
https://www.geeksforgeeks.org/java-swing-jpanel-with-examples/
https://docs.oracle.com/javase/7/docs/api/javax/swing/JFrame.html
database:
https://www.w3schools.com/MySQL/default.asp
https://www.youtube.com/watch?v=JR1c8p2apJM
Spring Data JPA:
https://www.digitalocean.com/community/tutorials/spring-data-jpa
Spring boot H2 database:
https://www.javatpoint.com/spring-boot-h2-database
https://www.youtube.com/watch?v=U3FH7YA__hI
https://github.com/aliasad-khowaja/spark-lms
https://stackoverflow.com/questions/18373383/jpa-onetoone-difference-between-cascade-merge-and-persist
https://www.geeksforgeeks.org/spring-boot-h2-database/
https://www.baeldung.com/spring-boot-h2-database
BootstrapCDN:
https://www.bootstrapcdn.com/fontawesome/
Java Spring & HTML
https://stackoverflow.com/questions/52006600/java-spring-simple-html-link
SLF4J:
https://www.baeldung.com/slf4j-with-log4j2-logback

Record Demo:
https://vimeo.com/780964877

In the process of making the project, we give priority to writing code. After completing part of the code, we analyze and reorganize the code framework and redesign it. This makes the code more cluttered and increases the workload. And

we didn't list the relationship of each class or interface in advance, which made the production process very complicated. Of course, the biggest problem is that they are not proficient in code learning and cannot use code flexibly.

Our code has been simplified a lot. We spent a lot of time in the database connection because we used a database model we had never been exposed to. Although we have learned the connection method of MySQL, we do not know why the connection has been bugged this time. So we changed the database schema. The h2 database was our first contact, but it did succeed.

Another key problem is the use of design patterns. In the process of creating our code, Later, we used the database as the main method but found it was very difficult to change the structure, so we could use the corresponding design pattern as in the initial design. This part still needs to be learned. However, we have tried new methods, such as framework, among which some patterns also meet our design requirements.

The last key problem is that the difference in the system leads to the difference in the environment configuration. The difference in many file configurations leads to the fact that we can only run on one computer, which has a considerable impact on our cooperation. That's why we waste a lot of time on environment configuration, such as database and web frameworks.