UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA

Book store

Name: Damo Ferenc
Group: 302310

Table of Contents

Deliverable 1

Project Specification

The project is a website, where the users can buy and read books, articles or other documents. These have a price, and the users can buy them paying with credit card, or they are free, and every user can read them. Every user has its own library, where the bought or collected materials can be found, and can be selected for reading. The users must register to have access to the books, and when they buy one, they must pay for it.

Functional Requirements

The users should be able to register with username, password and e-mail address, and to log in.
The users should be able to search for books and to put them, in their basket.
The users should be able to see a preview of the selected book.
The users should have a personal library, where they can select books and read them.
The books should be shown page by page, and the users should be able to change the page they are seeing.
The users should be able to buy the books, which are in their basket.
The users should be able to pay for the books, with credit card.
The system should check the validity of the given credit card number and security code, and should check the transactions completeness, before doing any action with the basket and the users library.
After a successful transaction, the books should be seen in the user's library.
The users should be able to get the free books by simply adding them to their library.
The system administrator should be able to add new books and edit and delete existing ones.
If a book is in the user's library, it should remain there even if it is deleted from the marketplace.
The system should allow the administrator to log in with username and password.
The system should allow the administrator to access the personal data of the users.

Use Case Model 1

Use Cases Identification

Use-Case: Chat with other users

Level: user goal

Primary Actor: user

Main success scenario:

1. *The user enters the chat.*
2. *The user chooses a nickname*
3. *The user sends messages and reads other user's messages.*
4. *The user existst the chat.*

Extensions: no extentions

Use Case Model 2

Use Cases Identification

Use-Case: Buy book

Level: user goal

Primary Actor: User

Main success scenario:

1. *The user selects a book and puts it in the shopping basket.*

2. *The system puts the books in the users basket.*
3. *The user selects the basket and confirms the buying.*
4. *The system requests the bank card number and security code.*
5. *The user enters the card number and the security code.*
6. *The system validates the data, and puts the book in the users library*

Extensions:

2-3

A. The users puts more books in the basket

5

a. The card number and the security code is not correct or the transacrion fails.

b. Return to step 4.

Use Case Model 3

Use Cases Identification

Use-Case: Rate book

Level: user goal

Primary Actor: User

Main success scenario:

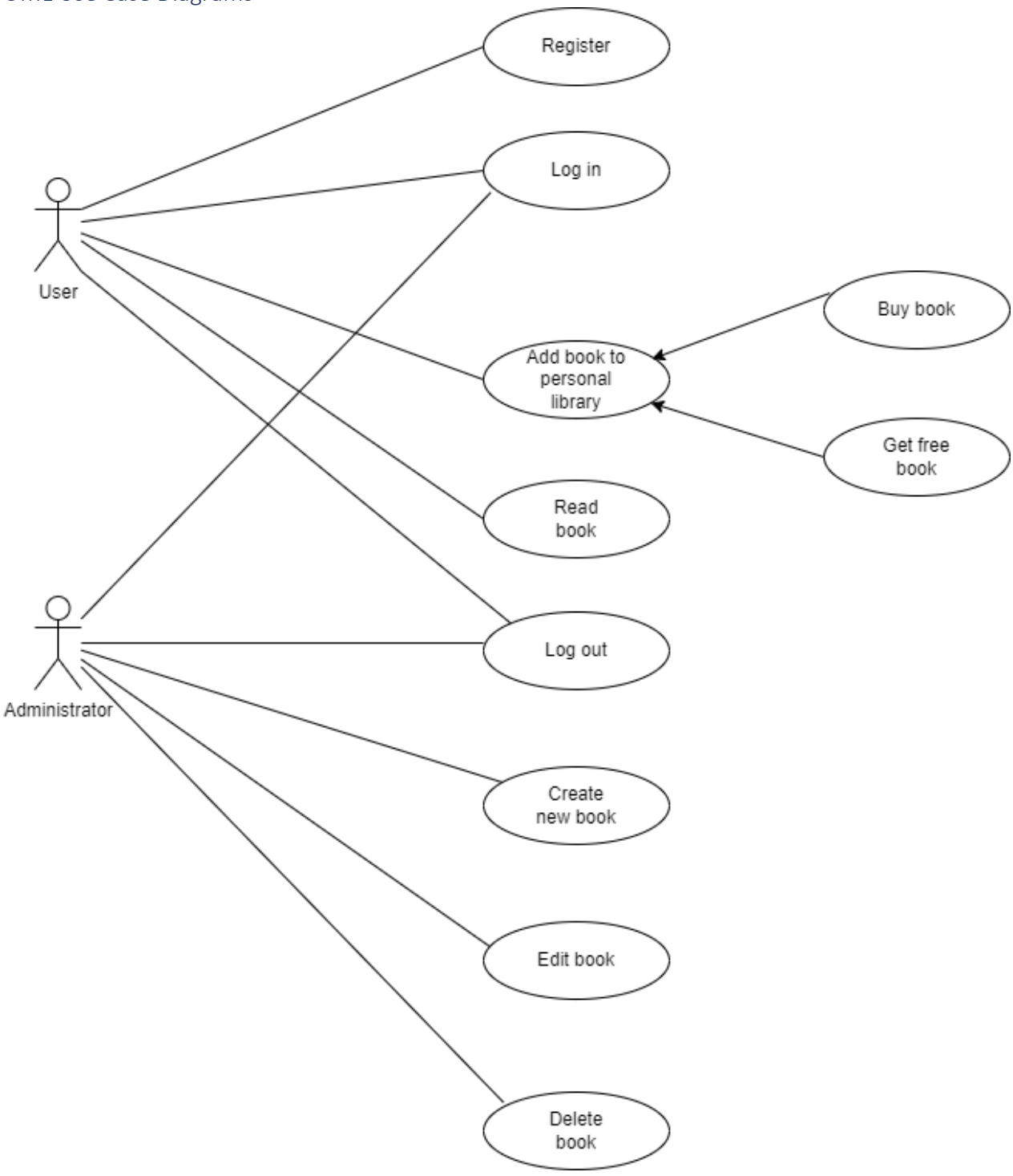
1. *The user selects the library.*
2. *The user selects the book.*
3. *The user rates the book.*

Extensions:

2

a. The user changes the page.

UML Use Case Diagrams



Supplementary Specification

Non-functional Requirements

The system should be performant, because it is necessary for the users to be able to switch fast between pages, to be able to read a book without waiting for the system.
Usability, because some users could be less used to web pages.
The system should be maintainable, because the number of the books and users can grow, and this should not have any effects on the system.
Security, because the system will get private data from the users, like credentials.

Design Constraints

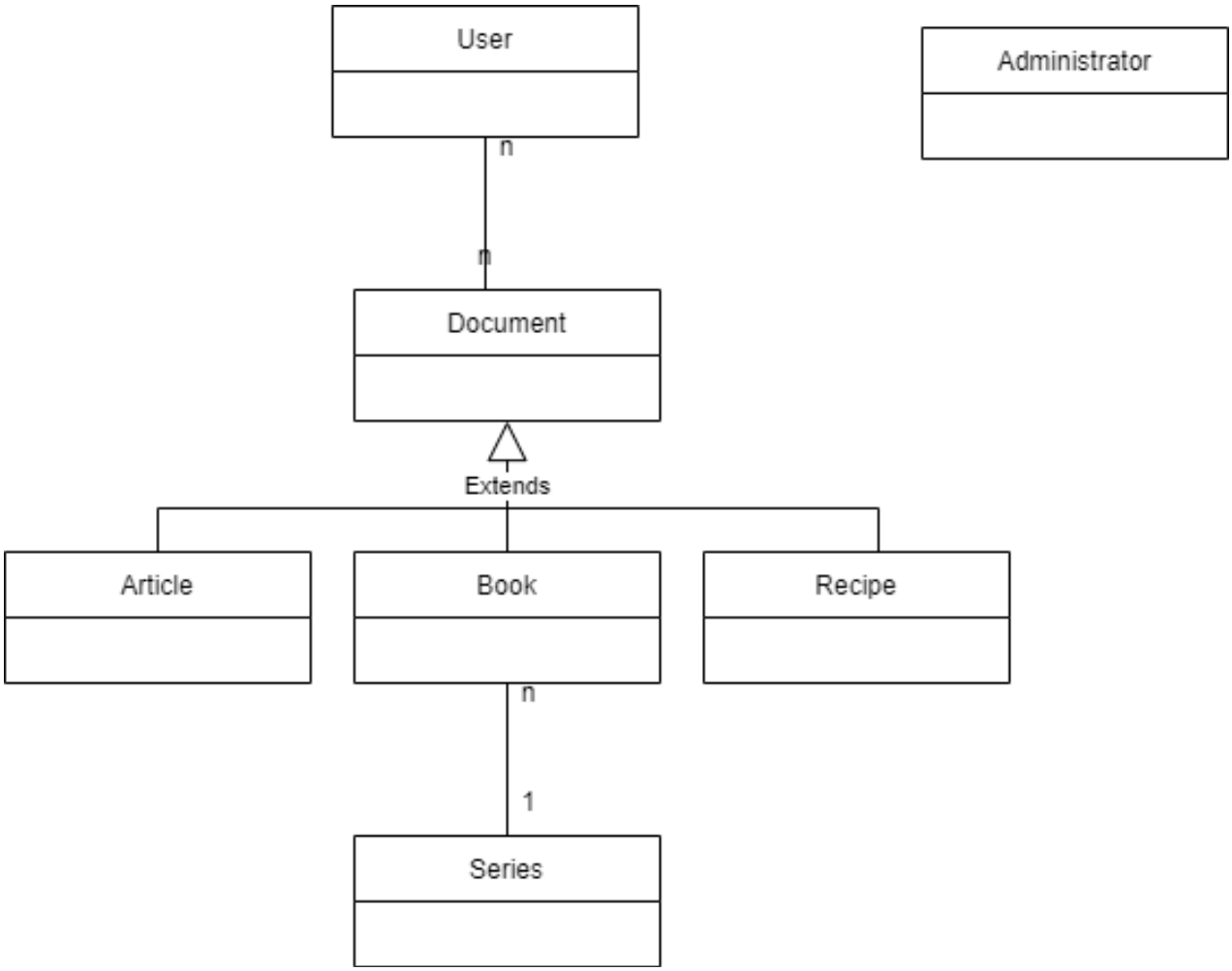
The language in which the system is implemented is java.
Should exist two different classes for the users and for the administrator.
The data must be stored in a database.

Glossary

User – The person, who is using the system, who reads and buys books.
Administrator – The person, who is the maintainer of the system, he is the only person who can edit the books or see the user's personal data.
Personal library – A storage where a user's books are, from where he can select the wanted one for reading.
Basket – The shopping cart, where the wanted books are stored.

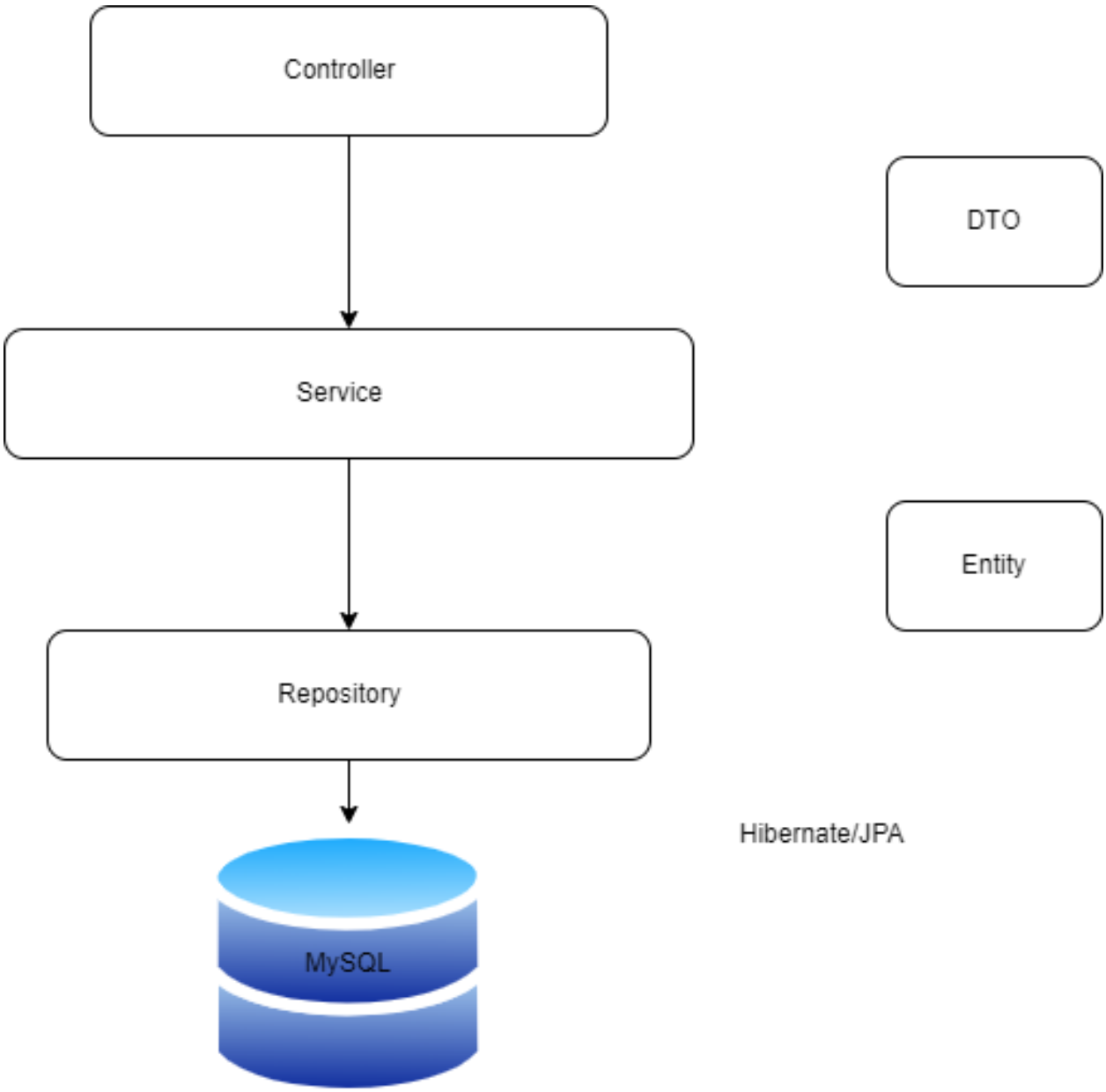
Deliverable 2

Domain Model

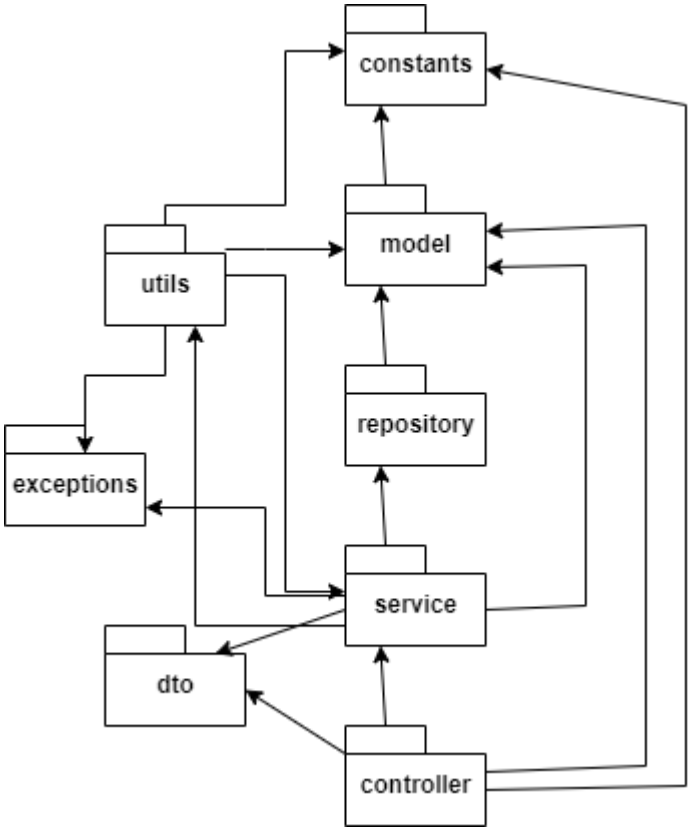


Architectural Design

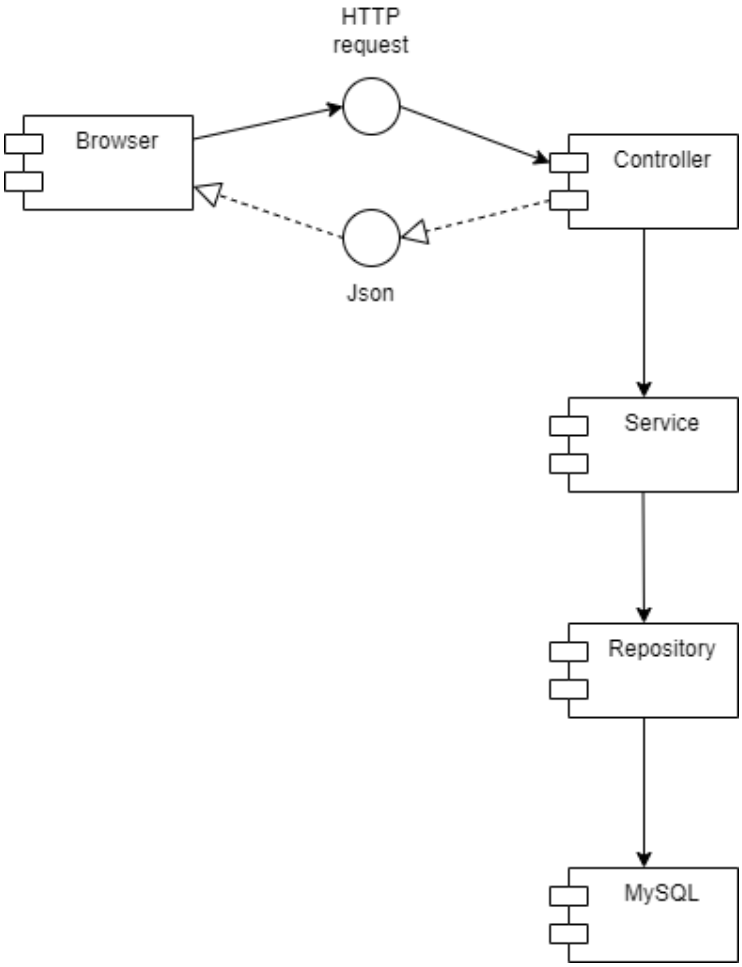
Conceptual Architecture

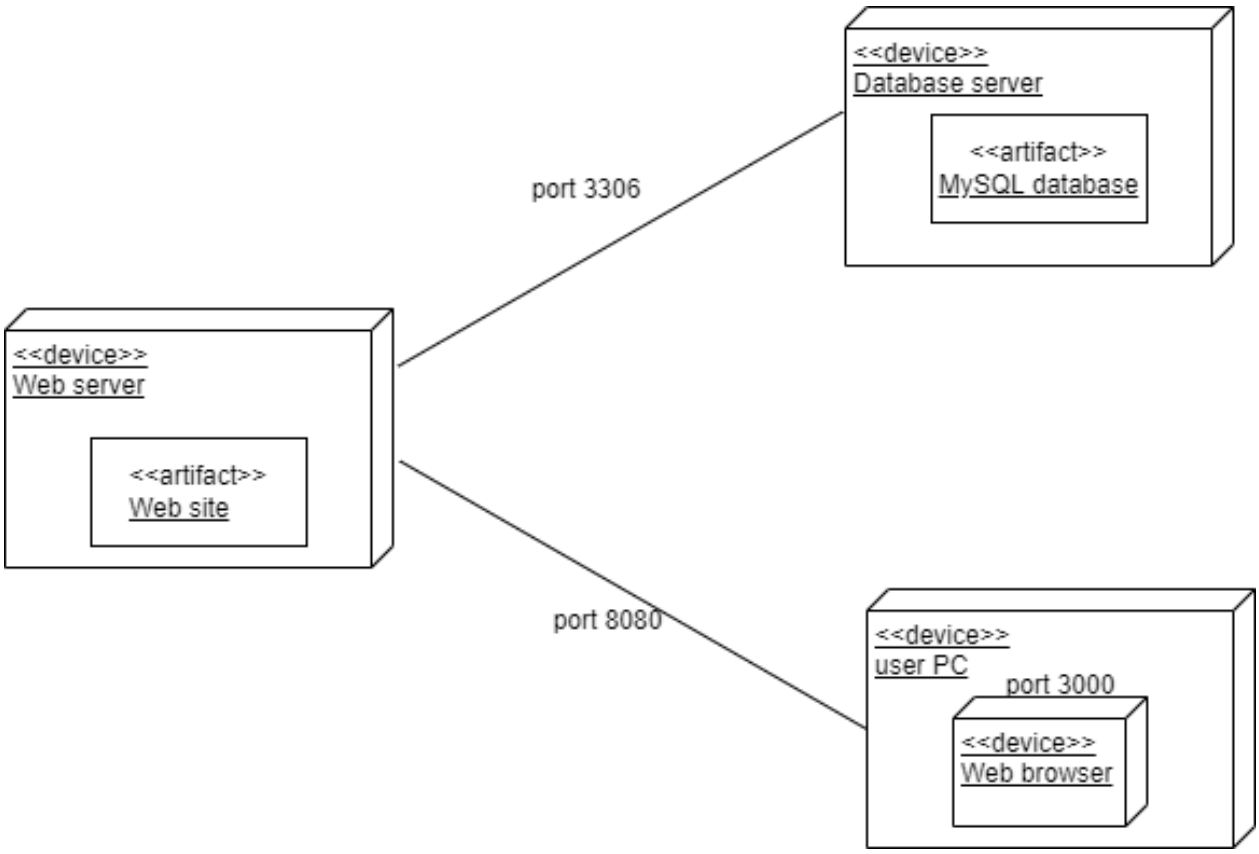


Package Design



Component and Deployment Diagram


HTTP
request
Browser
Controller
Json
Service
Repository
MySQL

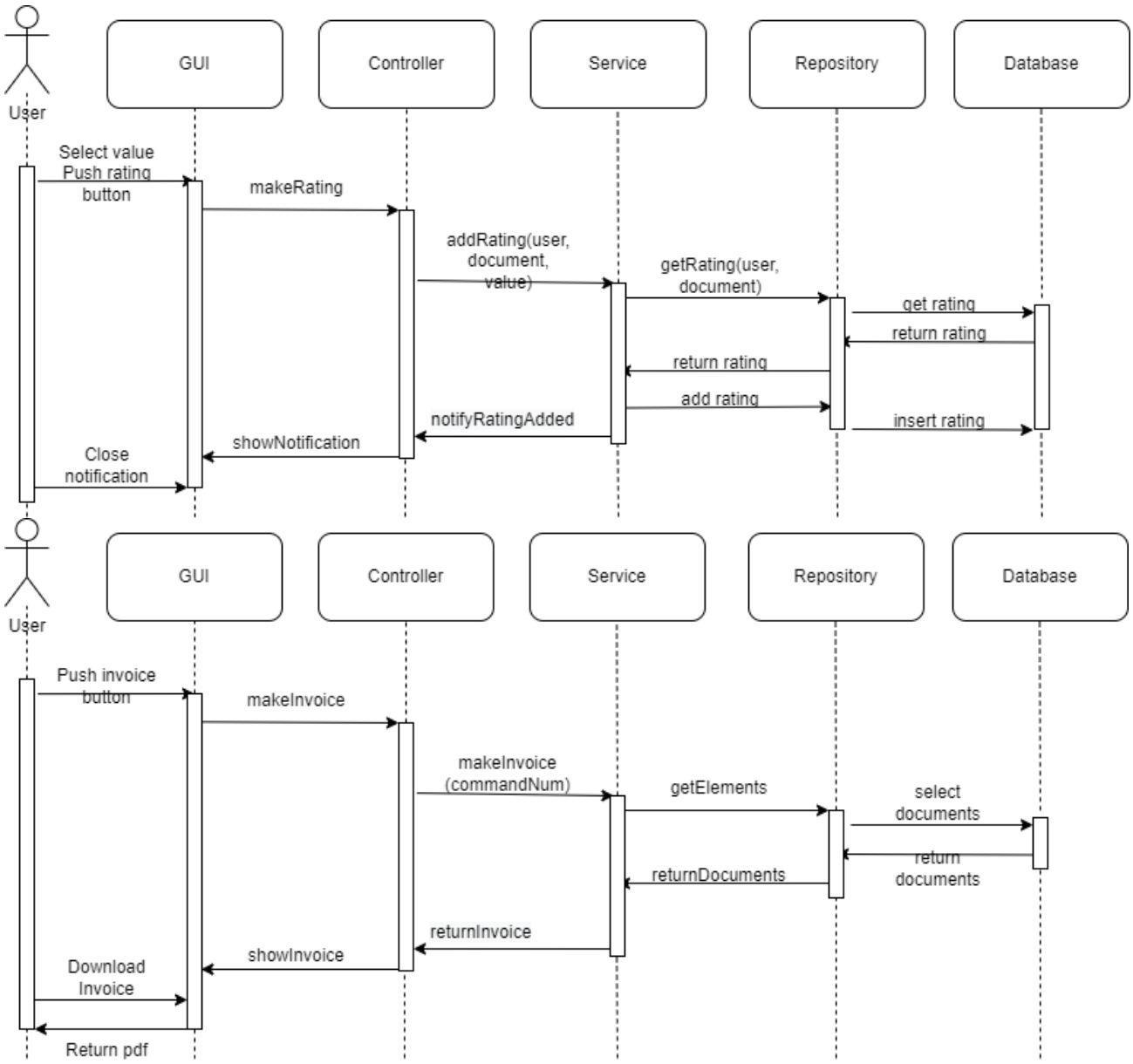


Deliverable 3

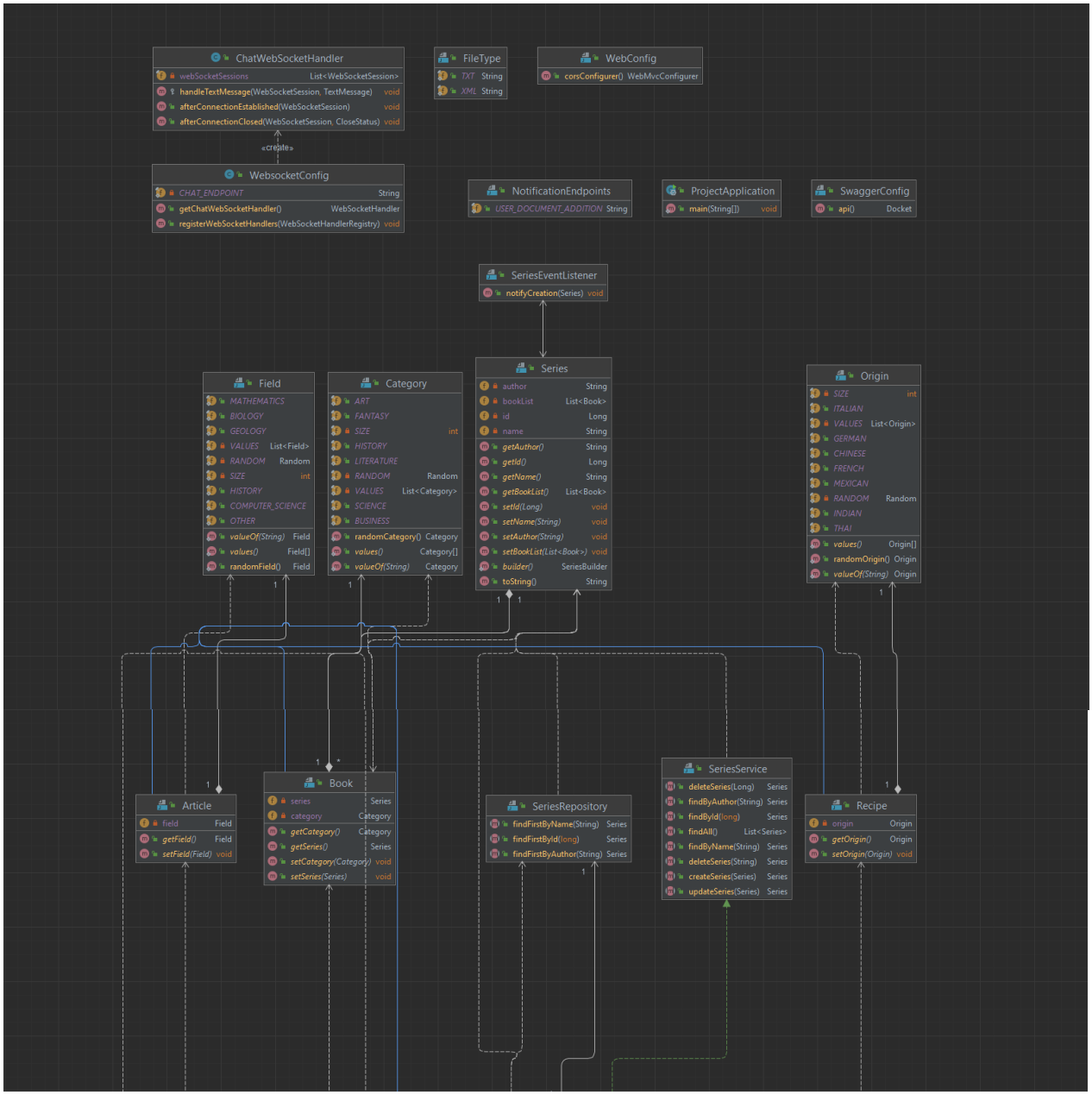
Design Model

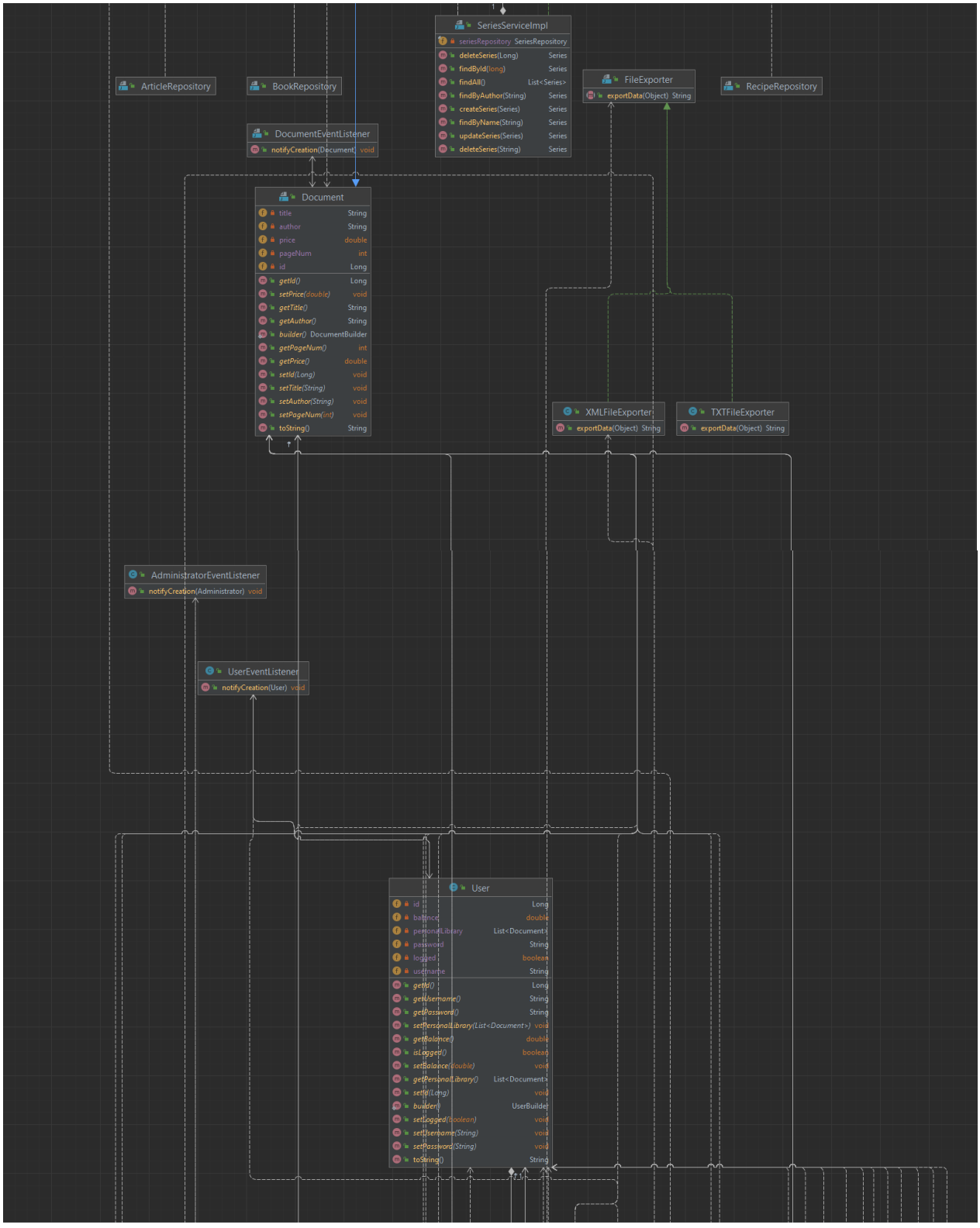
Dynamic Behavior

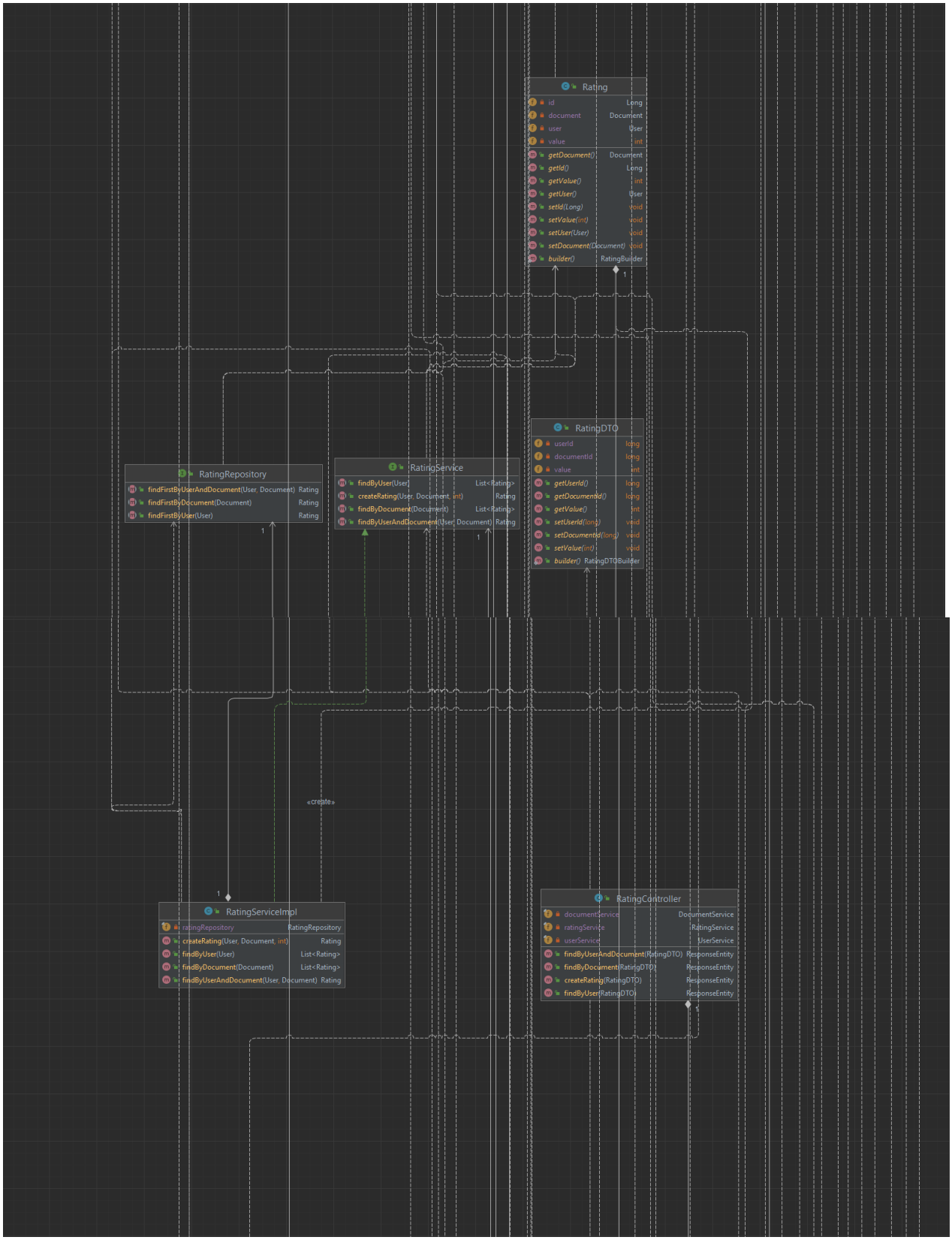
Sequence diagram for making a rating and for invoice generation.

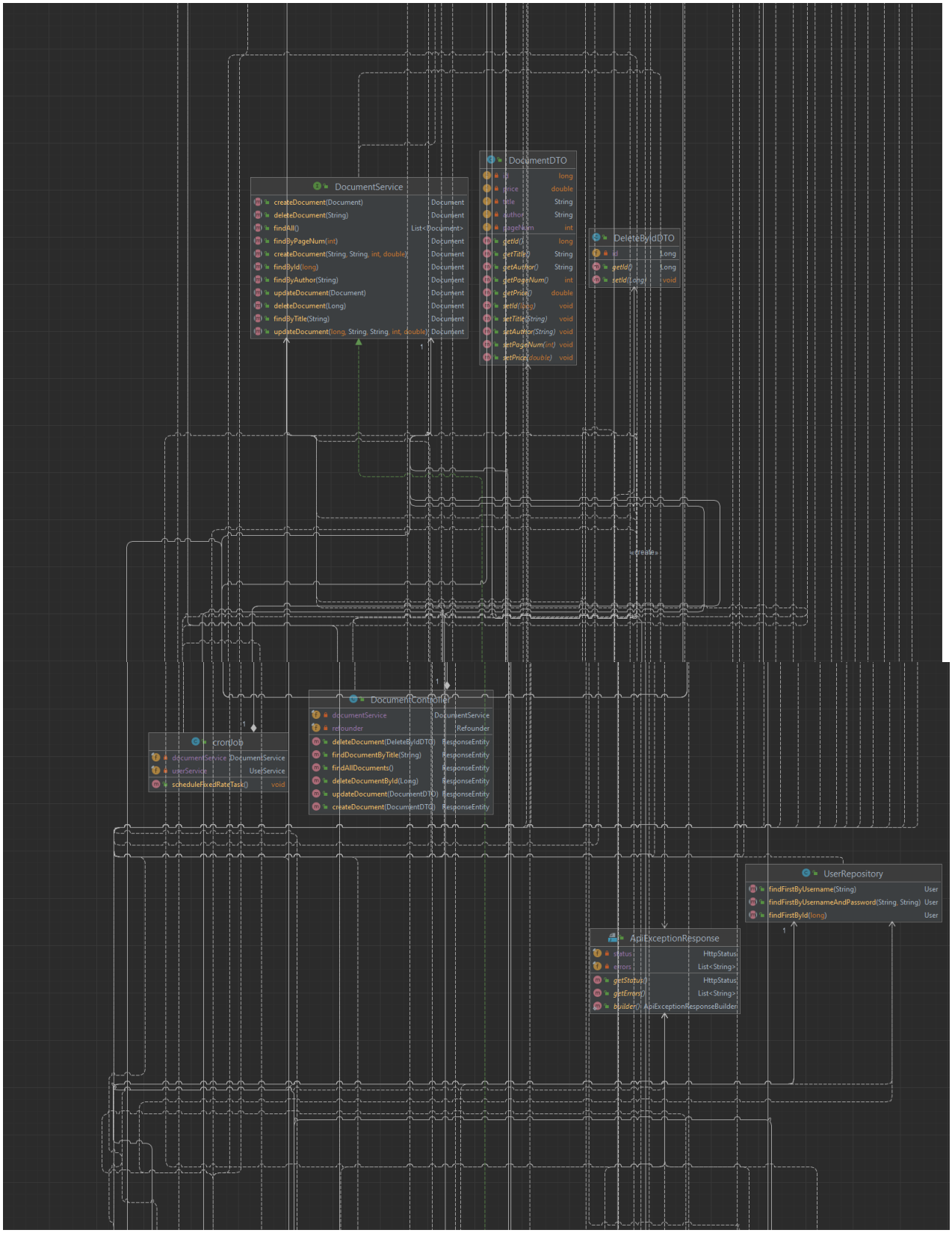


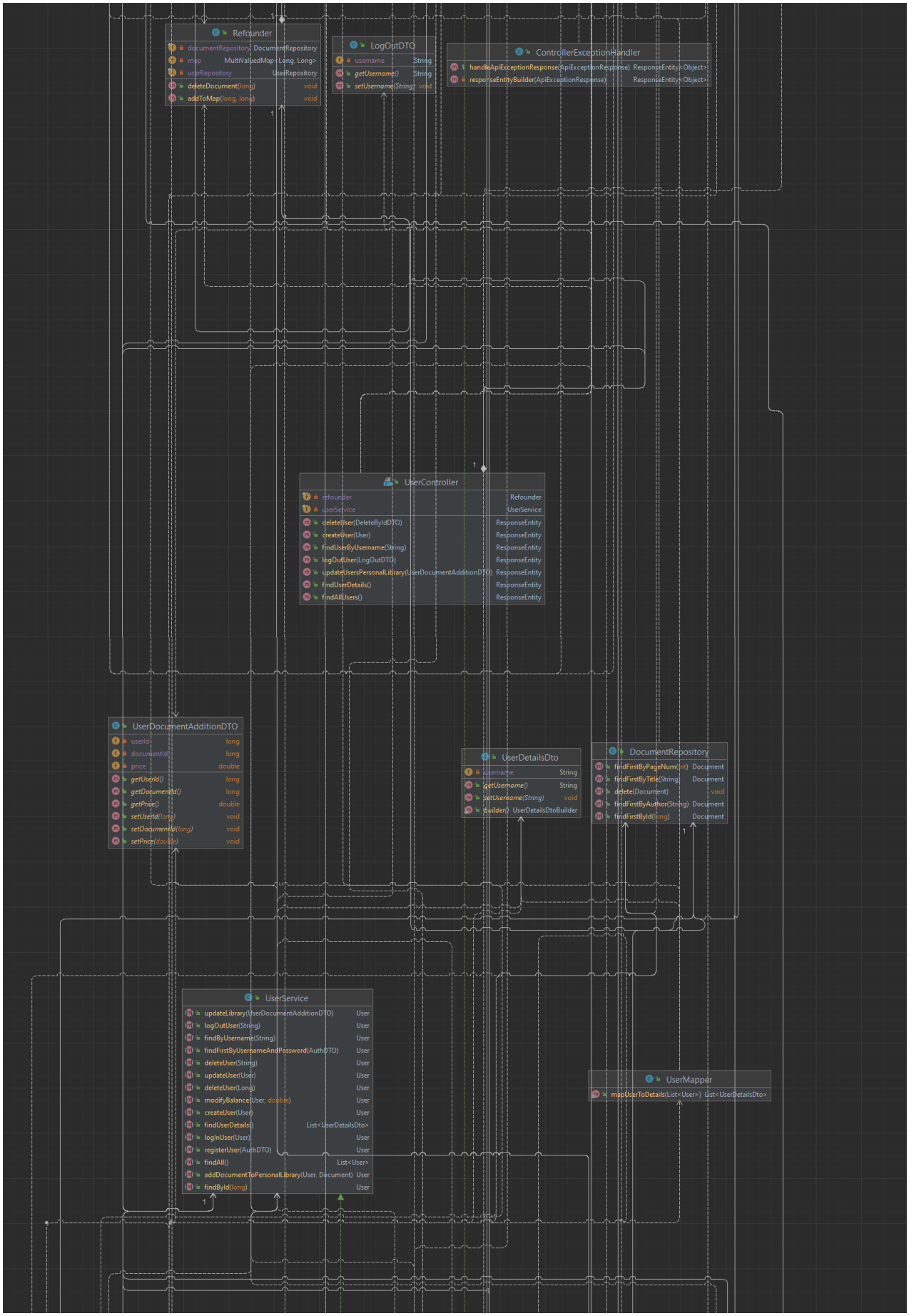
Class Diagram

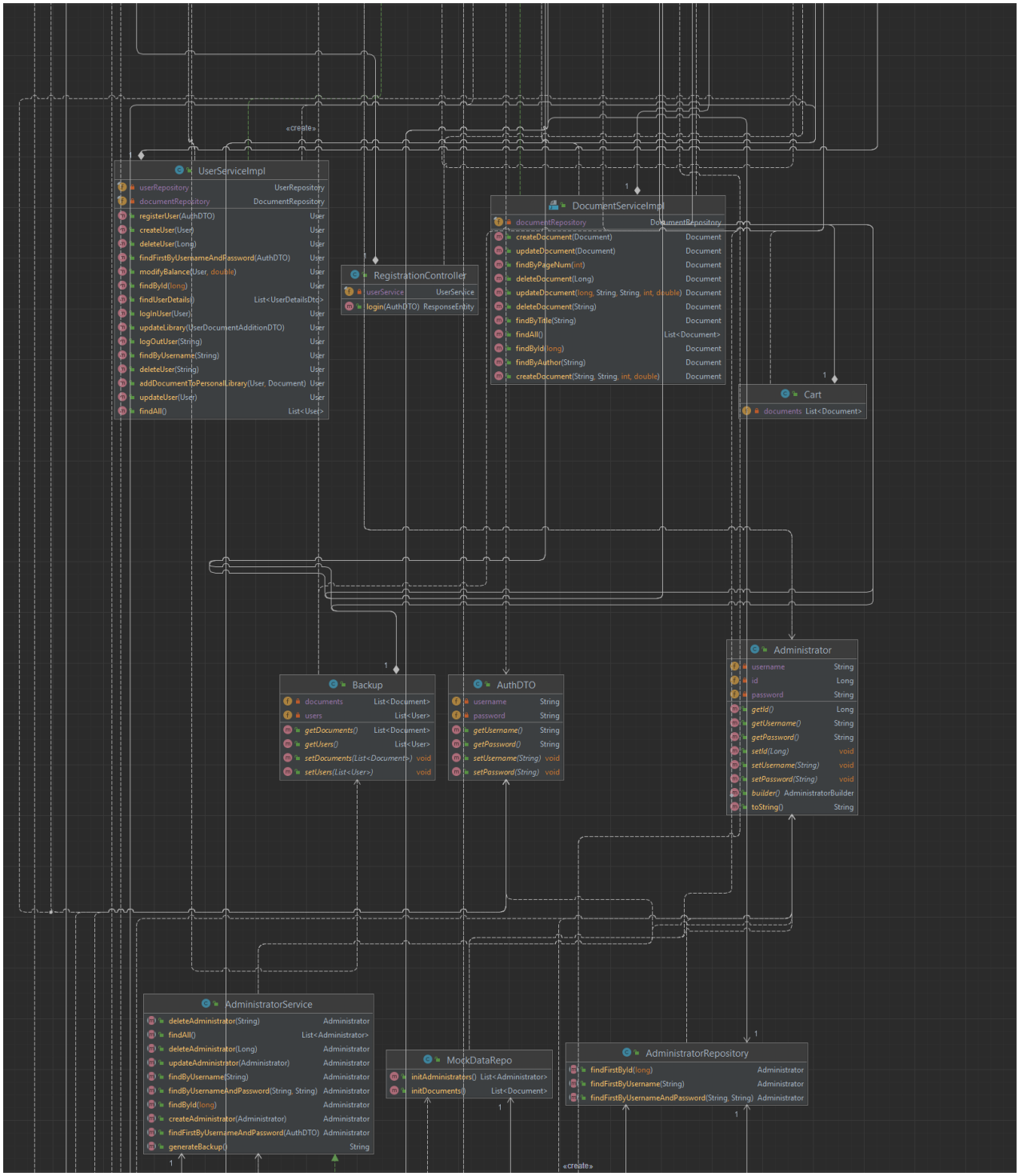

ChatWebSocketHandler
webSocketSessions List<WebSocketSession>
handleTextMessage(WebSocketSession, TextMessage) void
afterConnectionEstablished(WebSocketSession) void
afterConnectionClosed(WebSocketSession, CloseStatus) void
«create»
WebsocketConfig
CHAT_ENDPOINT String
getChatWebSocketHandler() WebSocketHandler
registerWebSocketHandlers(WebSocketHandlerRegistry) void
FileType
TXT String
XML String
WebConfig
corsConfigurer() WebMvcConfigurer
NotificationEndpoints
USER_DOCUMENT_ADDITION String
ProjectApplication
main(String[]) void
SwaggerConfig
api() Docket
SeriesEventListener
notifyCreation(Series) void
Field
MATHEMATICS
BIOLOGY
GEOLOGY
VALUES List<Field>
RANDOM Random
SIZE int
HISTORY
COMPUTER_SCIENCE
OTHER
valueOf(String) Field
values() Field[]
randomField() Field
Category
ART
FANTASY
SIZE int
HISTORY
LITERATURE
RANDOM Random
VALUES List<Category>
SCIENCE
BUSINESS
randomCategory() Category
values() Category[]
valueOf(String) Category
Series
author String
bookList List<Book>
id Long
name String
getAuthor() String
getId() Long
getName() String
getBookList() List<Book>
setId(Long) void
setName(String) void
setAuthor(String) void
setBookList(List<Book>) void
builder() SeriesBuilder
toString() String
Origin
SIZE int
ITALIAN
VALUES List<Origin>
GERMAN
CHINESE
FRENCH
MEXICAN
RANDOM Random
INDIAN
THAI
values() Origin[]
randomOrigin() Origin
valueOf(String) Origin
Article
field Field
getField() Field
setField(Field) void
Book
series Series
category Category
getCategory() Category
getSeries() Series
setCategory(Category) void
setSeries(Series) void
SeriesRepository
findFirstByName(String) Series
findFirstById(long) Series
findFirstByAuthor(String) Series
SeriesService
deleteSeries(Long) Series
findByAuthor(String) Series
findById(long) Series
findAll() List<Series>
findByName(String) Series
deleteSeries(String) Series
createSeries(Series) Series
updateSeries(Series) Series
Recipe
origin Origin
getOrigin() Origin
setOrigin(Origin) void

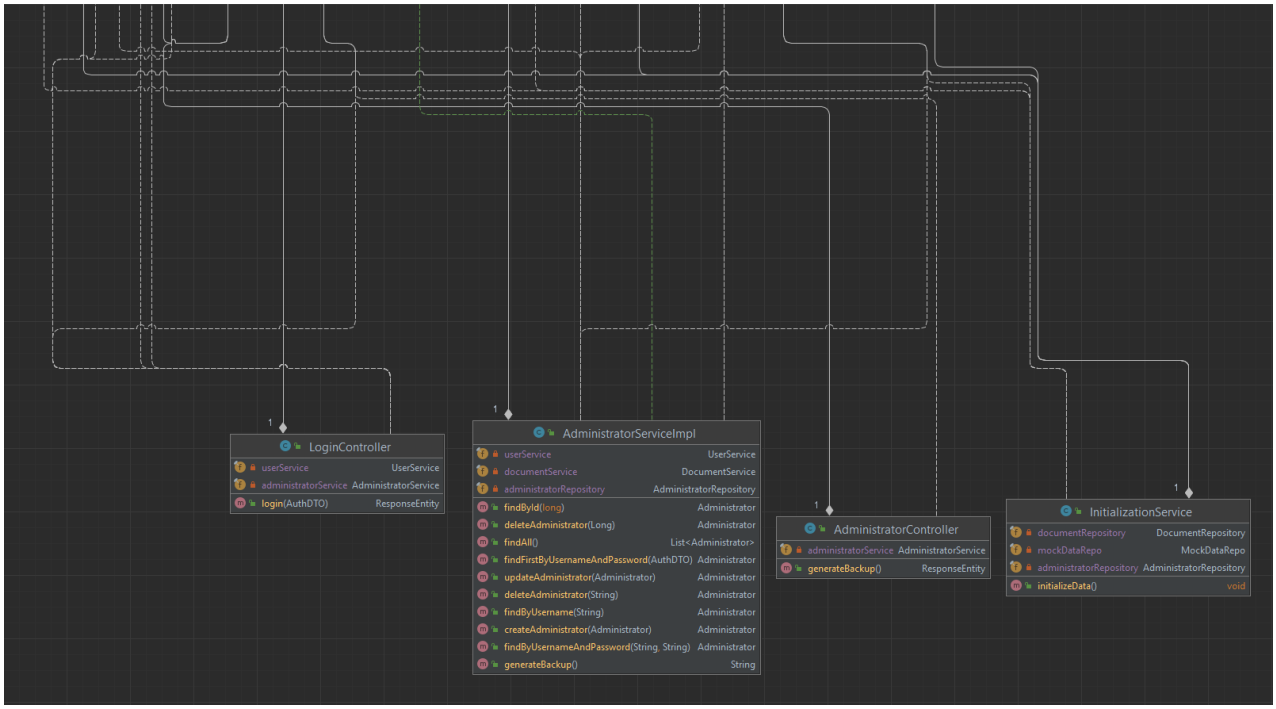

SeriesServiceImpl
seriesRepository SeriesRepository
deleteSeries(Long) Series
findById(long) Series
findAll() List<Series>
findByAuthor(String) Series
createSeries(Series) Series
findByName(String) Series
updateSeries(Series) Series
deleteSeries(String) Series
ArticleRepository
BookRepository
FileExporter
exportData(Object) String
RecipeRepository
DocumentEventListener
notifyCreation(Document) void
Document
title String
author String
price double
pageNum int
id Long
getId() Long
setPrice(double) void
getTitle() String
getAuthor() String
builder() DocumentBuilder
getPageNum() int
getPrice() double
setId(Long) void
setTitle(String) void
setAuthor(String) void
setPageNum(int) void
toString() String
XMLFileExporter
exportData(Object) String
TXTFileExporter
exportData(Object) String
AdministratorEventListener
notifyCreation(Administrator) void
UserEventListener
notifyCreation(User) void
User
id Long
balance double
personalLibrary List<Document>
password String
logged boolean
username String
getId() Long
getUsername() String
getPassword() String
setPersonalLibrary(List<Document>) void
getBalance() double
isLogged() boolean
setBalance(double) void
getPersonalLibrary() List<Document>
setId(Long) void
builder() UserBuilder
setLogged(boolean) void
setUsername(String) void
setPassword(String) void
toString() String


Rating
id Long
document Document
user User
value int
getDocument() Document
getId() Long
getValue() int
getUser() User
setId(Long) void
setValue(int) void
setUser(User) void
setDocument(Document) void
builder() RatingBuilder
RatingDTO
userId long
documentId long
value int
getUserId() long
getDocumentId() long
getValue() int
setUserId(long) void
setDocumentId(long) void
setValue(int) void
builder() RatingDTOBuilder
RatingRepository
findFirstByUserAndDocument(User, Document) Rating
findFirstByDocument(Document) Rating
findFirstByUser(User) Rating
RatingService
findByUser(User) List<Rating>
createRating(User, Document, int) Rating
findByDocument(Document) List<Rating>
findByUserAndDocument(User, Document) Rating
«create»
RatingServiceImpl
ratingRepository RatingRepository
createRating(User, Document, int) Rating
findByUser(User) List<Rating>
findByDocument(Document) List<Rating>
findByUserAndDocument(User, Document) Rating
RatingController
documentService DocumentService
ratingService RatingService
userService UserService
findByUserAndDocument(RatingDTO) ResponseEntity
findByDocument(RatingDTO) ResponseEntity
createRating(RatingDTO) ResponseEntity
findByUser(RatingDTO) ResponseEntity

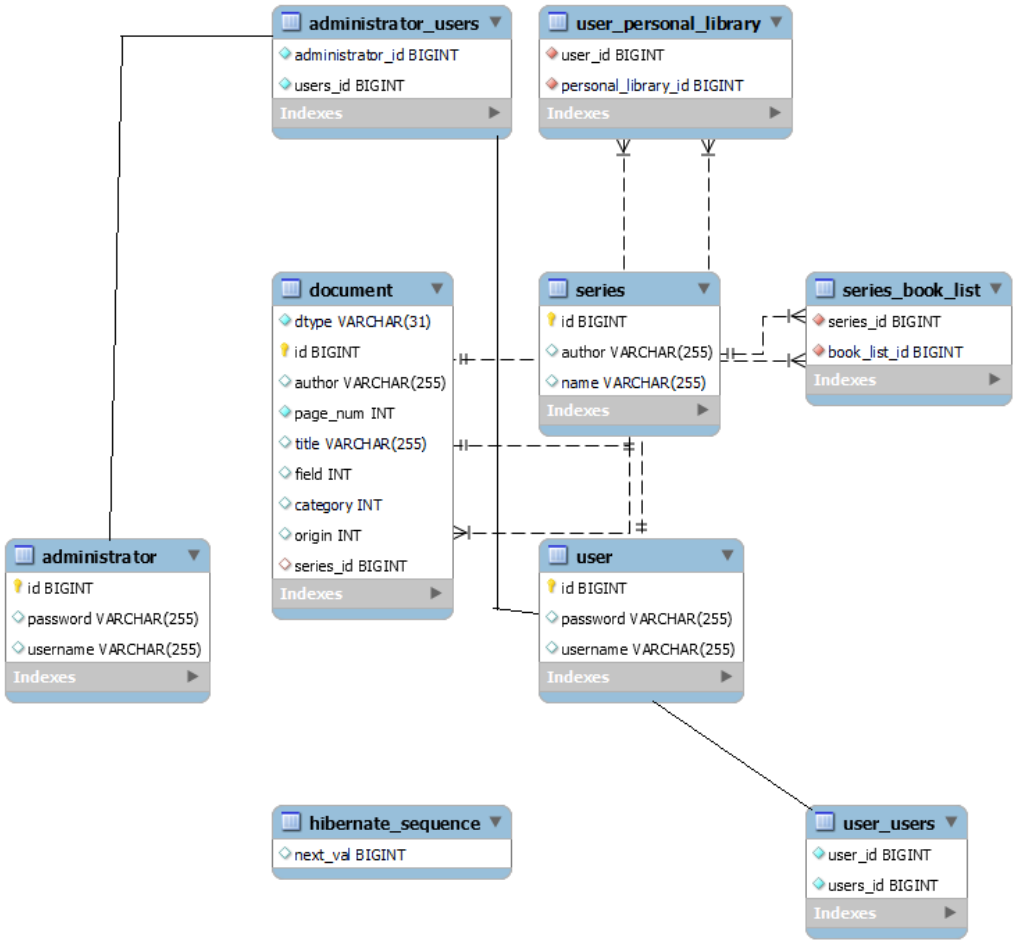

DocumentService
createDocument(Document) Document
deleteDocument(String) Document
findAll() List<Document>
findByPageNum(int) Document
createDocument(String, String, int, double) Document
findById(long) Document
findByAuthor(String) Document
updateDocument(Document) Document
deleteDocument(Long) Document
findByTitle(String) Document
updateDocument(long, String, String, int, double) Document
DocumentDTO
id long
price double
title String
author String
pageNum int
getId() long
getTitle() String
getAuthor() String
getPageNum() int
getPrice() double
setId(long) void
setTitle(String) void
setAuthor(String) void
setPageNum(int) void
setPrice(double) void
DeleteByIdDTO
id Long
getId() Long
setId(Long) void
«create»
DocumentController
documentService DocumentService
refounder Refounder
deleteDocument(DeleteByIdDTO) ResponseEntity
findDocumentByTitle(String) ResponseEntity
findAllDocuments() ResponseEntity
deleteDocumentById(Long) ResponseEntity
updateDocument(DocumentDTO) ResponseEntity
createDocument(DocumentDTO) ResponseEntity
cronJob
documentService DocumentService
userService UserService
scheduleFixedRateTask() void
UserRepository
findFirstByUsername(String) User
findFirstByUsernameAndPassword(String, String) User
findFirstById(long) User
ApiExceptionResponse
status HttpStatus
errors List<String>
getStatus() HttpStatus
getErrors() List<String>
builder() ApiExceptionResponseBuilder


Refounder
documentRepository: DocumentRepository
map: MultiValuedMap<Long, Long>
userRepository: UserRepository
deleteDocument(long): void
addToMap(long, long): void
LogOutDTO
username: String
getUsername(): String
setUsername(String): void
ControllerExceptionHandler
handleApiExceptionResponse(ApiExceptionResponse): ResponseEntity<Object>
responseEntityBuilder(ApiExceptionResponse): ResponseEntity<Object>
UserController
refounder: Refounder
userService: UserService
deleteUser(DeleteByIdDTO): ResponseEntity
createUser(User): ResponseEntity
findUserByUsername(String): ResponseEntity
logOutUser(LogOutDTO): ResponseEntity
updateUsersPersonalLibrary(UserDocumentAdditionDTO): ResponseEntity
findUserDetails(): ResponseEntity
findAllUsers(): ResponseEntity
UserDocumentAdditionDTO
userId: long
documentId: long
price: double
getUserId(): long
getDocumentId(): long
getPrice(): double
setUserId(long): void
setDocumentId(long): void
setPrice(double): void
UserDetailsDto
username: String
getUsername(): String
setUsername(String): void
builder(): UserDetailsDtoBuilder
DocumentRepository
findFirstByPageNum(int): Document
findFirstByTitle(String): Document
delete(Document): void
findFirstByAuthor(String): Document
findFirstById(long): Document
UserService
updateLibrary(UserDocumentAdditionDTO): User
logOutUser(String): User
findByUsername(String): User
findFirstByUsernameAndPassword(AuthDTO): User
deleteUser(String): User
updateUser(User): User
deleteUser(Long): User
modifyBalance(User, double): User
createUser(User): User
findUserDetails(): List<UserDetailsDto>
logInUser(User): User
registerUser(AuthDTO): User
findAll(): List<User>
addDocumentToPersonalLibrary(User, Document): User
findById(long): User
UserMapper
mapUserToDetails(List<User>): List<UserDetailsDto>


«create»
UserServiceImpl
userRepository UserRepository
documentRepository DocumentRepository
registerUser(AuthDTO) User
createUser(User) User
deleteUser(Long) User
findFirstByUsernameAndPassword(AuthDTO) User
modifyBalance(User, double) User
findById(long) User
findUserDetails() List<UserDetailsDto>
logInUser(User) User
updateLibrary(UserDocumentAdditionDTO) User
logOutUser(String) User
findByUsername(String) User
deleteUser(String) User
addDocumentToPersonalLibrary(User, Document) User
updateUser(User) User
findAll() List<User>
RegistrationController
userService UserService
login(AuthDTO) ResponseEntity
DocumentServiceImpl
documentRepository DocumentRepository
createDocument(Document) Document
updateDocument(Document) Document
findByPageNum(int) Document
deleteDocument(Long) Document
updateDocument(long, String, String, int, double) Document
deleteDocument(String) Document
findByTitle(String) Document
findAll() List<Document>
findById(long) Document
findByAuthor(String) Document
createDocument(String, String, int, double) Document
Cart
documents List<Document>
Backup
documents List<Document>
users List<User>
getDocuments() List<Document>
getUsers() List<User>
setDocuments(List<Document>) void
setUsers(List<User>) void
AuthDTO
username String
password String
getUsername() String
getPassword() String
setUsername(String) void
setPassword(String) void
Administrator
username String
id Long
password String
getId() Long
getUsername() String
getPassword() String
setId(Long) void
setUsername(String) void
setPassword(String) void
builder() AdministratorBuilder
toString() String
AdministratorService
deleteAdministrator(String) Administrator
findAll() List<Administrator>
deleteAdministrator(Long) Administrator
updateAdministrator(Administrator) Administrator
findByUsername(String) Administrator
findByUsernameAndPassword(String, String) Administrator
findById(long) Administrator
createAdministrator(Administrator) Administrator
findFirstByUsernameAndPassword(AuthDTO) Administrator
generateBackup() String
MockDataRepo
initAdministrators() List<Administrator>
initDocuments() List<Document>
AdministratorRepository
findFirstById(long) Administrator
findFirstByUsername(String) Administrator
findFirstByUsernameAndPassword(String, String) Administrator
«create»


LoginController
userService UserService
administratorService AdministratorService
login(AuthDTO) ResponseEntity
AdministratorServiceImpl
userService UserService
documentService DocumentService
administratorRepository AdministratorRepository
findById(long) Administrator
deleteAdministrator(Long) Administrator
findAll() List<Administrator>
findFirstByUsernameAndPassword(AuthDTO) Administrator
updateAdministrator(Administrator) Administrator
deleteAdministrator(String) Administrator
findByUsername(String) Administrator
createAdministrator(Administrator) Administrator
findByUsernameAndPassword(String, String) Administrator
generateBackup() String
AdministratorController
administratorService AdministratorService
generateBackup() ResponseEntity
InitializationService
documentRepository DocumentRepository
mockDataRepo MockDataRepo
administratorRepository AdministratorRepository
initializeData() void

Data Model


administrator_users
administrator_id BIGINT
users_id BIGINT
Indexes
user_personal_library
user_id BIGINT
personal_library_id BIGINT
Indexes
document
dtype VARCHAR(31)
id BIGINT
author VARCHAR(255)
page_num INT
title VARCHAR(255)
field INT
category INT
origin INT
series_id BIGINT
Indexes
series
id BIGINT
author VARCHAR(255)
name VARCHAR(255)
Indexes
series_book_list
series_id BIGINT
book_list_id BIGINT
Indexes
administrator
id BIGINT
password VARCHAR(255)
username VARCHAR(255)
Indexes
user
id BIGINT
password VARCHAR(255)
username VARCHAR(255)
Indexes
hibernate_sequence
next_val BIGINT
user_users
user_id BIGINT
users_id BIGINT
Indexes

System Testing

Some classes were tested with unit testing methodology: a first level of testing which is often performed by the developers themselves. It is the process of ensuring individual components of a piece of software at the code level are functional and work as they were designed to. Developers in a test-driven environment will typically write and run the tests prior to the software or feature being passed over to the test team. Unit testing will also make debugging easier because finding issues earlier means they take less time to fix than if they were discovered later in the testing process.
The CRUD methods of the service classes were tested using this method. The tests were made using Mockito, which is an open source testing framework for Java. With the mock annotation were created and injected mocked instances of repository classes without having to call Mockito.mock manually. In the setUp method with when().thenReturn() and doNothing().when() the repository methods were set to return the input or do nothing. For example if the findFirstByUsername() method of UserRepository class was called, its code was not run, instead the input was returned. This was necessary because the test verified just the service method, and not the repository too. So when a call to repository was made, the response was the input. If a repository method was void, the the doNothing was used. For example the delete method shouldn't return anything, so when it was called by the delete service method, its code was not execued, but it was checked if it was called with the verify method.
In the test methods, the output was compared to the correct value, and it was checked if it is null or not.
All the CRUD operations of the service classes were tested using this method.

Future Improvements

Future improvements could be some new functionalities, like on-line reading or book writing for users.
The system could be extended for music or videos too.
The security could be improved.

Conclusion

The system is a functional book buying website and server with database. The users can easily buy new documents, and the administrator can add new documents. The data is sent to the server and is stored. The users can use a forum-like chat to communicate with each other.

Bibliography

https://en.wikipedia.org/wiki/Non-functional_requirement
https://www.baeldung.com/
https://www.w3schools.com/
stackoverflow