Bookie

Software Design Specification

09.05.2021

Talha Bayburtlu 150118066
Zeynep Alıcı 150119517
Ahmet Akıl 150118038

Prepared for
CSE3044 Software Engineering Term Project

Table of Contents

1. Introduction 2
   1.1. Purpose 2
   1.2. Statement of scope 2
   1.3. Software context 2
   1.4. Major constraints 2
   1.5. Definitions 2
   1.6. Acronyms and Abbreviations 2
   1.7. References 2
2. Design Consideration 2
   2.1. Design Assumptions and Dependencies 2
   2.2. General Constraints 3
   2.3. System Environment 3
   2.4. Development Methods 3
3. Architectural and component-level design 3
   3.1. System Structure 4
      3.1.1. Architecture diagram 4
   3.2. Description for Component n 4
      3.2.1. Processing narrative (PSPEC) for component n 4
      3.2.2. Component n interface description. 4
      3.2.3. Component n processing detail 4
   3.3. Dynamic Behavior for Component n 4
      3.3.1. Interaction Diagrams 4
4. Restrictions, limitations, and constraints 5
5. Conclusion 5

1. Introduction

This section provides an overview of the entire requirement document. This document describes all data, functional and behavioral requirements for software.

1.1. Purpose

This application provides its users to exchange books by exploring other Bookie's virtual libraries in their city and getting communication with them. Users are able to display their books by creating their virtual libraries and filling it with books which allows users to be discovered by other users and have a chance to meet and chat with the people that are having similar tendencies.

1.2. Statement of scope

First of all, users must be able to register to the application with their personal information like name, surname, phone, email, the city that they are living etc. Then after a verify process user will be able to use other functionalities in our application by just login-in process. Those are kind of authentication services which are quite essential for our application in order to separate users from one to another and give the application a secure layer.

Then, the users must be able to create their virtual libraries by searching their books on the application with the book's name and then the user must pick the book in the result list. By searching and picking the books, the virtual library is going to be formed. This is a functionality which is quite essential because it is a fundamental functionality of the application.

Users should be able to explore other users' virtual libraries and can view it as personal or can view the book's and relevant information with it. There will be a comment section for each of the book's that are in a virtual library. This enables the communication for users to interact with each other by commenting on a book that is posted. This feature is essential because it is a fundamental functionality of the application. But we planned a section for users to get contact in by calling, sending SMS and sending WhatsApp messages which is a future requirement and not in first deployment of the application. Also, this feature may be changed with a live chat section inside in our application but for a feature like that the developers must learn, and practice required technologies (for example Apache Kafka) which may not be suitable with limited time.

1.3. Software context

We want users to interact with themselves by exploring other users' virtual libraries, but we do not want to make the communication information of users public because this may result in a legal problem. Because of that we thought a comment section for a posted book is a best result for the communication process.

Also, the books that are owned by them can create a conflict like when the book is not the book that is expected after an exchange. We are using Google Books API for resolving that. The book that is posted can't create a conflict because the relevant information of the book like image, the name, the publisher etc. are constant.

1.4. Major constraints

Application's book search functionality is dependent on Google Books API. We are planning to store the books that are placed inside virtual libraries in an SQL Database. But in the search phase the addition of books to the library is applied with Google Books API. Also, the book shown in a post with relevant information includes an image which we don't store and comes from Google Books.

We don't have a test server to access and make our tests in that environment. This results the database, the backend and the mobile application should be run for each of the developers which may require time to interact with it.

Also, we are planning to deploy the application on Google Play Store but not on Apple Store because Apple Store requires an Apple membership and the membership costs 100 dollars per year which we can't afford.

1.5. Definitions

- Application: A mobile application, most commonly referred to as an app, is a type of application software designed to run on a mobile device, such as a smartphone or tablet computer. Mobile applications frequently serve to provide users with similar services to those accessed on PCs.
- Database: A structured set of data held in a computer, especially one that is accessible in various ways.
- Google Books: Google Books is a service from Google Inc. that searches the full text of books and magazines that Google has scanned, converted to text using optical character recognition (OCR), and stored in its digital database.

1.6. Acronyms and Abbreviations

- API: Application Programming Interface
- SQL: Structured Query Language
- BLOC: Business Logic Component
- MVVM: Model View ViewModel Design Pattern

1.7. References

- SOFTWARE DESIGN SPECIFICATION (rivier.edu) (SDS example that examined)
- CS487 Software Engineering Design Specification Template (iit.edu) (SDS example that examined)
- How it works - Apple Developer Program (Apple Store cost issue)

- Google Books - Wikipedia (Google Books detailed information)

2. Design Consideration

2.1. Design Assumptions and Dependencies

The software should be used in a modern smartphone and should be used by a person that knows how to use a smartphone and familiar with standard UI elements such as forms and buttons

2.2. General Constraints

The application needs to have an internet connection to communicate with the spring backend to authenticate and fetch the required information, the performance will be fine in any smartphone that was manufactured in the recent years

2.3. System Environment

- An IDE that can be used to developer Flutter Application (ex: IntelliJ or Android Studio)
- Flutter & Dart SDK
- Apache Tomcat Server
- Maven (not necessarily need to be in system environment because it comes with code itself, but it is better to have it in the system)
- Java JDK
- PostgreSQL Database and pgAdmin

2.4. Development Methods

Frontend: While thinking about a design approach we thought about one of the most well-known flutter state management solutions called bloc (https://pub.dev/packages/flutter_bloc) which provides an event driven programming solution where the user interface fires events and the business logic component (bloc) yields the appropriate state for that given event.

But after that we decided to use the stacked (https://pub.dev/packages/stacked) MVVM like flutter architecture which provides a way to separate the services (the repository), the view models (business logic) and the presentation layer (user interfaces)

Backend: While designing the Backend, we decided to use Java, which is a language suitable for object-oriented programming techniques. We also decided to use the spring framework is an application framework and inversion of control container for the Java platform. As a database management system, we decided to use PostgreSQL. We also decided to use Spring JPA repository to increase the applicability of database transactions. We used spring security configuration interfaces for the management and reliability of the user's authentication operations.

3. Architectural and component-level design

Our program is divided into five main layers:

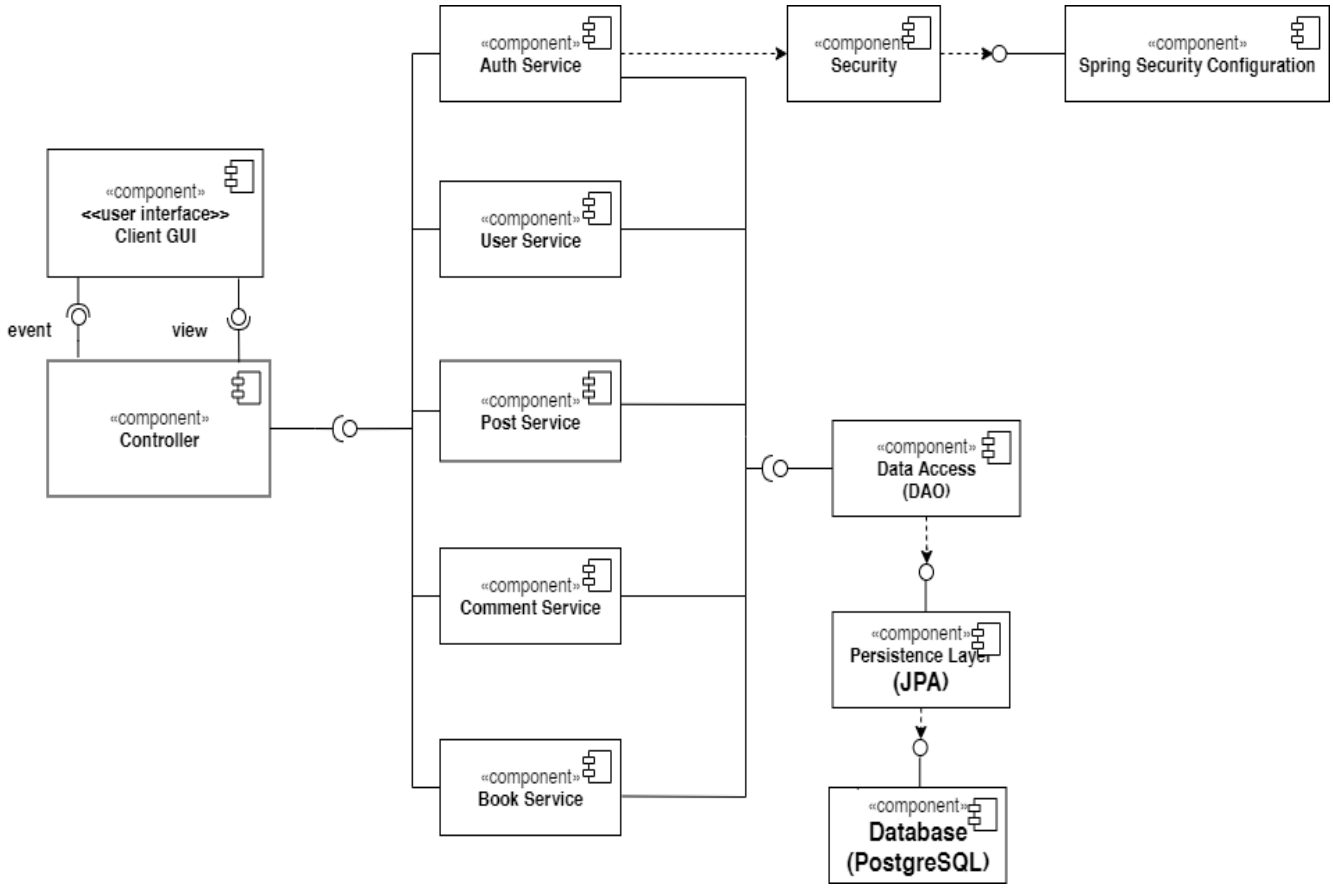
- Client Layer: In this layer, GUI is represented, and the users can explore mechanics. The mechanics will send requests to the backend system and receive expected responses. Which will be handled in the ViewModels of the ClientLayer.
- Controller Layer: This layer consists of endpoints that are defined as API (Application Programming Interface). The request that is made from the client will end up here. The basic data transmutations can be done before sending and after receiving relevant information to/from the service layer.
- Service Layer: This layer contains the business logic of the backend system. Based on the data being transferred from the controller layer some actions are taken. It is possible to reach other services from one service. Also, generally there will be a call placed in services to data access object (DAO) layer which returns asked information from the database. Returned information is mutated based on the business logic and when it is ready to respond to the client, it will be sent back to the controller layer.
- Data Access Object (DAO) Layer: This layer communicates between the database and service. Necessary information can get with the help of Hibernate which predefines some basic implementations of basic database operations and makes implementation of complex database operations.
- Database (PostgreSQL): The database layer is responsible for holding information to work on as records on tables that are defined with columns and suitable variable types. A connection can be open from DAO to database for information transactions. After that the connection closes and the information continue the journey.

3.1. System Structure

As mentioned in the architectural and component-level design section, we defined the client layer as a component. There is more than one controller in the controller layer but because their work is not different from one to another, we decided to call all controllers as a one component. There are more than one services in the services layer, but their work differs from each other and we count them as separate components. These services are, authentication service, user service, post service, comment service and book service. Authentication service uses a security component to authenticate users. Security component is controlled by the spring security configuration component which creates a configuration for the security side of the backend system.

Those services are connected to the DAO layer and because all DAO layers do similar things, we specify them as a component. The persistence component maps the POJO's to database records. And the database layer consists of organized data to work on.

3.1.1. Architecture diagram



3.2 Description for Components

Component: Client GUI

3.2 Description

The Client GUI is the component that allows the users to interact with the system

3.2.1 Processing narrative

It is the component that contains all of the User Interfaces and frontend application state management which communicates with the Controller component

3.2.2 Interface Description

The application should follow the MVVM design principles where data models and view models handle the required business logics of the application where the view layer will simply be used to show the data on the actual User interface.

3.2.3 Processing Detail

3.2.3.1 Design Class hierarchy

There is a separate view class for each screen of the application and each view class also has a viewModel attached to it to handle the business logic. Also, the Services classes of the GUI will communicate with the Controller

3.2.3.2 Restrictions/limitations

Views contain only the data that needs to be displayed on the screen.

3.2.3.3 Performance issues

The MVVM state management approach is performant since it separates the views and the business logics.

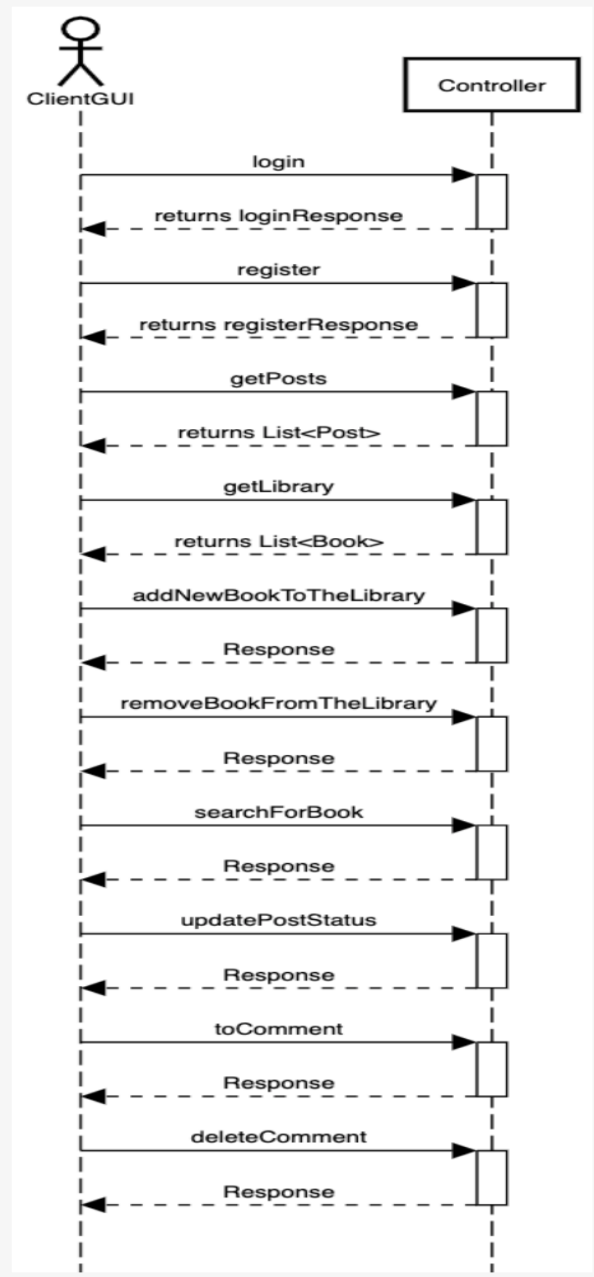
3.2.3.4 Design constraints

The data layer and the presentation layer should be separated as much as possible.

3.3 Dynamic Behavior For DataAccess(DAO)

Client layer handles all of the user interactions by sending a request to the Controller and waiting until the request is resolved, and displaying the proper user interface with respect to the request.

3.3.1 Interaction Diagrams



Component: DataAccess (DAO)

3.2 Description

The data access layer is the component in which the methods that enable database operations to take place in interaction with the persistence layer are implemented.

3.2.1 Processing narrative (PSPEC)

It is the component that contains the methods that provide operations such as getting data from the database and updating the data in the database. It is related to the properties of the objects in the project and the operations that the user or the administrator wants to perform are determined by the method signatures in the dao layer.

3.2.2 Interface Description

In general, unique parameters such as id, email, which are required to perform transactions on the database, are given as inputs. The inputs may vary depending on the action the user wants to do. Parameters such as the title of the book the user wants to search, the id of the comment made by the user, the id of the post are possible inputs of this component.

The result of the database operation is returned as an object, and the output of this component is an object corresponding to a table in the database. Or it is a response object created according to need.

3.2.3 Processing Detail

3.2.3.1 Design Class hierarchy

There is a separate DAO class for each table in the database. These classes are independent of each other and do not use each other's methods and attributes.

Dao Classes: Book Repository, User Repository, Post Repository, Comment Repository, Address Repository

3.2.3.2 Restrictions/limitations

Dao classes contain only the required operations used at the business layer.

3.2.3.3 Performance issues

The functions from the JPA repository are used. So, the performance issue is taken care of with a secondary library used at the back side.

3.2.3.4 Design constraints

Dao classes should be independent from each other and separate for each table. JPA repository should be used.

3.2.3.5 Processing detail for each operation

Operations of Address Repository:

- getByCity: It returns the address object by city name.

Operations of Book Repository:

- getById: It returns a book object (an entry at the book table) by id.
- getAllByIdIn: It gets an id list and returns the list of book objects.

Operations of Comment Repository:

- getAllByPost: It gets a post object and returns it’s all comments.
- getById: It returns a comment object by comment id.

Operations of Post Repository:

- getByBookIdAndUserId: It returns the post of a specific user book.
- getById: It returns a post object by post id.
- deleteByBookIdAndUserId: It deletes a record from the database by book id and user id.

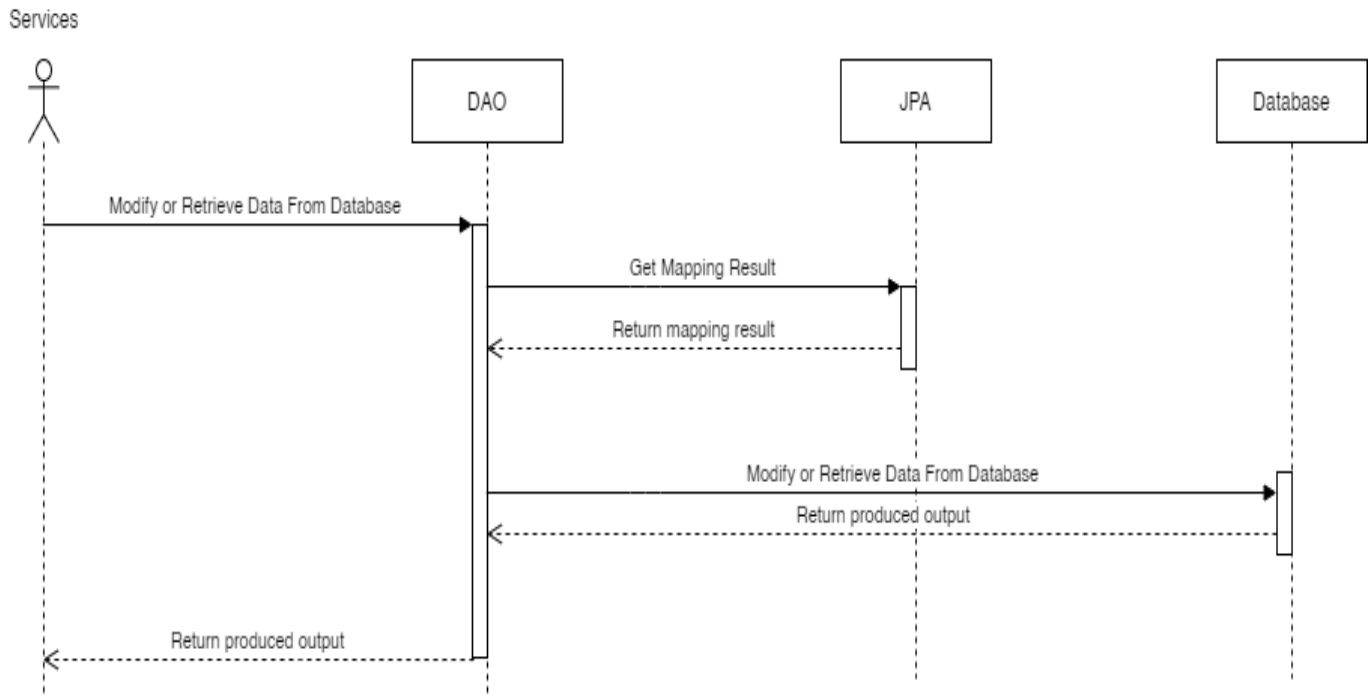
Operations of User Repository

- getByEmail: It gets specific users by its unique email.
- getById: It gets users by its unique id.
- getByAddress_City: It gets the list of users that are living in the city given as parameters.
-

3.3 Dynamic Behavior For DataAccess (DAO)

Data Access Objects are used for communication between services and the database. Services calls relevant methods from DAO's for modifying records in tables or retrieving records in tables from the database. DAO uses JPA (Persistence layer) for mapping entities in the backend system to table format in database. A connection is opened between database and DAO and after the transaction is made, the connection is closed.

3.3.1 Interaction Diagrams



Component: User Service

3.2 Description

The user service is where the operations associated with the user table in the database and related business rules are implemented. Operations and rules related to the user are determined in this service.

3.2.1 Processing narrative (PSPEC)

The user service component includes listing users according to their properties such as the city they live in and their email. At the same time, it is the class that the rules are implemented in, for the user to see his own library and write comments. This is a component where the HTTP operations, inputs and outputs and limits are determined on the respective controller. The method signatures in the user service determine which objects will be given in response to the user and what kind of input will be received from the user.

3.2.2 Interface Description

Inputs to user service are the same as those in the dao tier. Only business rules are set here. Input requirements have been previously determined in the data access layer, and if we list the inputs in this component; It consists of the user's email, id, city where they live, and the book id they want to add to their library. Outputs are the user itself, the list of users matching the given input, and the user's book list.

3.2.3 Processing Detail

3.2.3.1 Design Class hierarchy

This component is a single business class. User service injects other services (Book Service, Comment Service) and is used. Although it is an independent component by itself, other service components and user repository are used and dependent on these components.

3.2.3.2 Restrictions/limitations

The user must be logged in to the system in order to perform the operations belonging to the user and must have a record in the user table.

3.2.3.3 Performance issues

Since services use repository, they are dependent on spring because they use Spring JPA Repository

3.2.3.4 Design constraints

It is independent of other service components. It uses all its own attributes and functions and does not use an attribute belonging to another service class. Only it is dependent on its own repository. (User Repository)

3.2.3.5 Processing detail for each operation

findById: The user is found by its id.
findByEmail: The user is found by its email.

getUsersByCity: All users that living the city (is given as input) are retrieved from database.

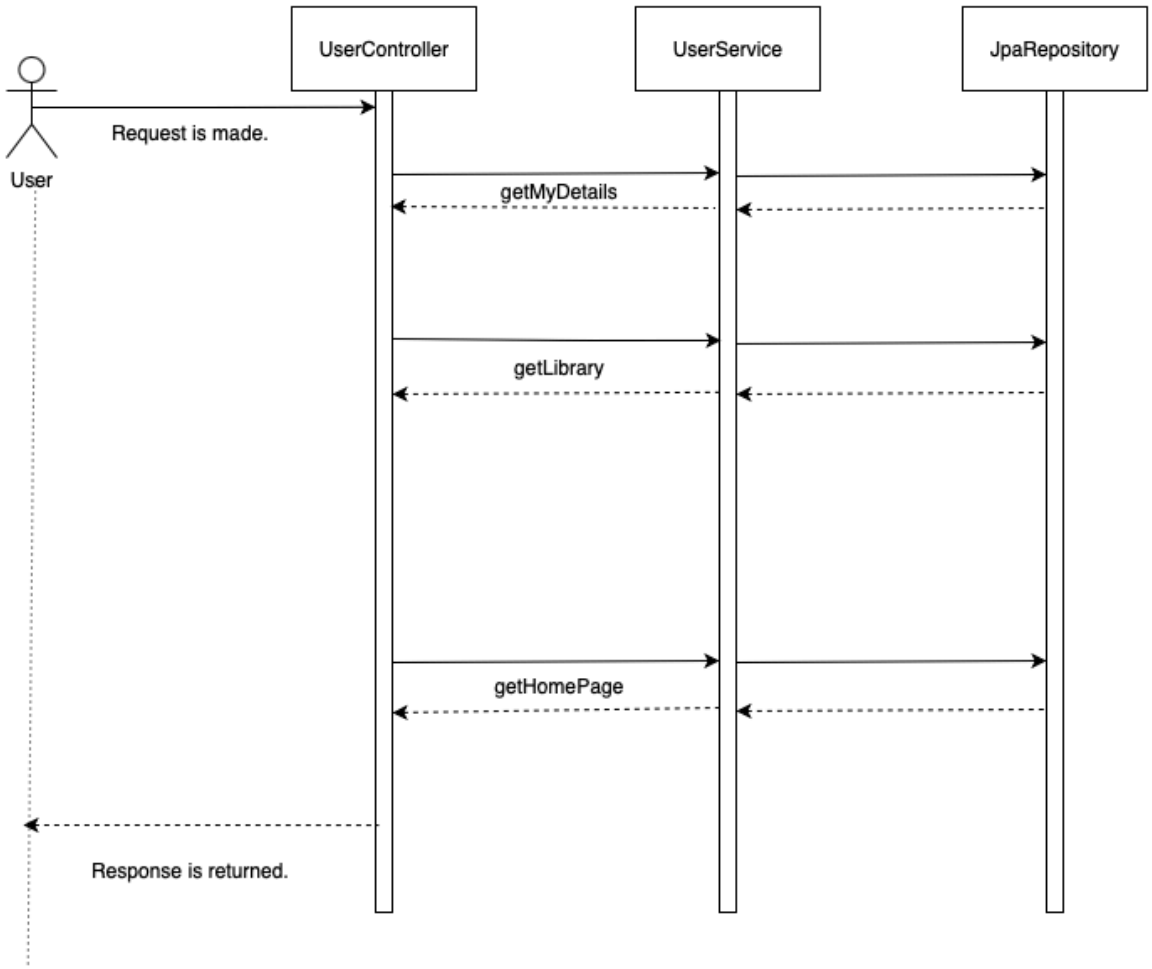
getLibraryByUserId: Using the post service, all the books belonging to the relevant user are listed.

getHomepagePostsByUserId: The limit of objects that will appear on the user's homepage is determined by HomepagePostResponse. The HomepagePostResponse list that will be on this user's homepage is returned by user id.

3.3 Dynamic Behaviour

UserService consists of the management of the parts of the operations that the user wants to do in the database that affect the user and are affected by the user. After the user makes a request to the UserController, here the UserService methods are called and the business rules are applied. Later, if business rules are passed, these calls are forwarded to the reposritory. The necessary information is taken from the database or recorded in the database.

3.3.1 Interaction Diagrams



Component: Post Service

3.2 Description

The basic building block of the Bookie application is the post entity that contains the book shared by the user, its description, and the status of this book. The component in which the rules of the transactions to be performed with this entity in the database are determined is the post service.

3.2.1 Processing narrative (PSPEC)

HTTP get, post, delete operations from post controller are passed to this component. The business rules specified here are implemented separately for each operation. The operations to be performed regarding the post table in the database must meet the rules in this component.

After the rules here are fulfilled and the necessary restrictions are passed, database transactions are forwarded to the post repository.

3.2.2 Interface Description

Post id, user id, book id, and post description are the inputs for this component. Its output is an object belonging to the post from the entity layer or it is the PostStatus dto.

3.2.3 Processing Detail

3.2.3.1 Design Class hierarchy

3.2.3.2 Restrictions/limitations

A user can share the same book only once. Post status must be available for other users to request this book.

3.2.3.3 Performance issues

Since services use repository, they are dependent on spring because they use Spring JPA Repository.

3.2.3.4 Design constraints

It is independent of other service components. It uses all its own attributes and functions and does not use a attribute belonging to another service class. Only it is dependent on its own repository.(Post Repository)

3.2.3.5 Processing detail for each operation

Add: The id of the user and the id of the book are taken and added a post record to the database.
Delete: The id of the user and the id of the book are taken and a record deleteted from the database.
getBooksByUserId: With the user's id, all posts belonging to this user are taken. Then, the book list of these posts is obtained from these posts and returned.

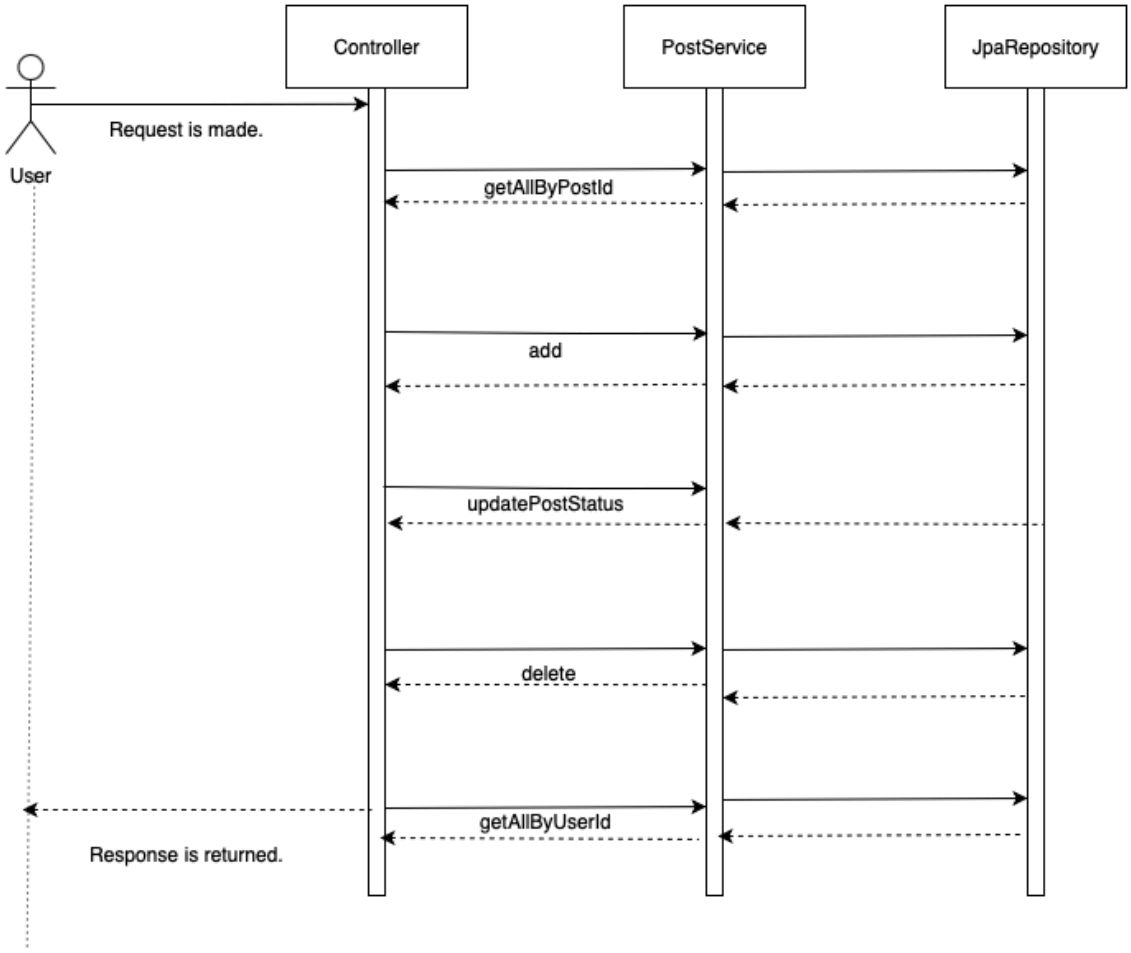
getAllPostsByUserId: All posts belonging to this user are taken by user id.

updatePostStatus: The id of the post, the owner of the post and post status is given as parameters. Then post status updated with this status parameter.

3.3 Dynamic Behaviour

PostService consists of the management of the parts of the operations that the user wants to do in the database that affect the post table and are affected by the post table. After the user makes a request to the Controller, here the PostService methods are called and the business rules are applied. Later, if business rules are passed, these calls are forwarded to the repository. The necessary information is taken from the database or recorded in the database.

3.3.1 Interaction Diagrams



Component: Comment Service

3.2 Description

It is the component that contains the rules for the user to comment on the books of other users in the system. The limits of the transactions that can take place regarding the comment in the database are determined here.

3.2.1 Processing narrative (PSPEC)

The text of the comment you want to write and the information of which post you want to comment is given to the controller. The CommentService methods are called and the repostiyory layer can be accessed after the necessary rules are provided here. In addition, the

methods for commenting in our application, such as listing the comments of a post, deleting a comment, are included in this component.

3.2.2 Interface Description

Post id, comment id, User id, user object, post object are possible inputs of this component. It is also an entry CommentDTO, which contains the information needed to write a comment. Its output is the comment object and the comment object list.

3.2.3 Processing Detail

3.2.3.1 Design Class hierarchy

It is a single class and uses the book repository, which related to the book entity, although it is independent from other components. It does not use the attributes of another component.

3.2.3.2 Restrictions/limitations

Any comments must not exceed the character limit in the database.

3.2.3.3 Performance issues

Since services use repository, they are dependent on spring because they use Spring JPA Repository.

3.2.3.4 Design constraints

It is independent of other service components. It uses all its own attributes and functions and does not use an attribute belonging to another service class. Only it is dependent on its own repository. (Comment Repository)

3.2.3.5 Processing detail for each operation

getAllByPost: All comments belonging to the post given as parameter in the method are retrieved by calling the method in the repository and the list of these comments is returned.

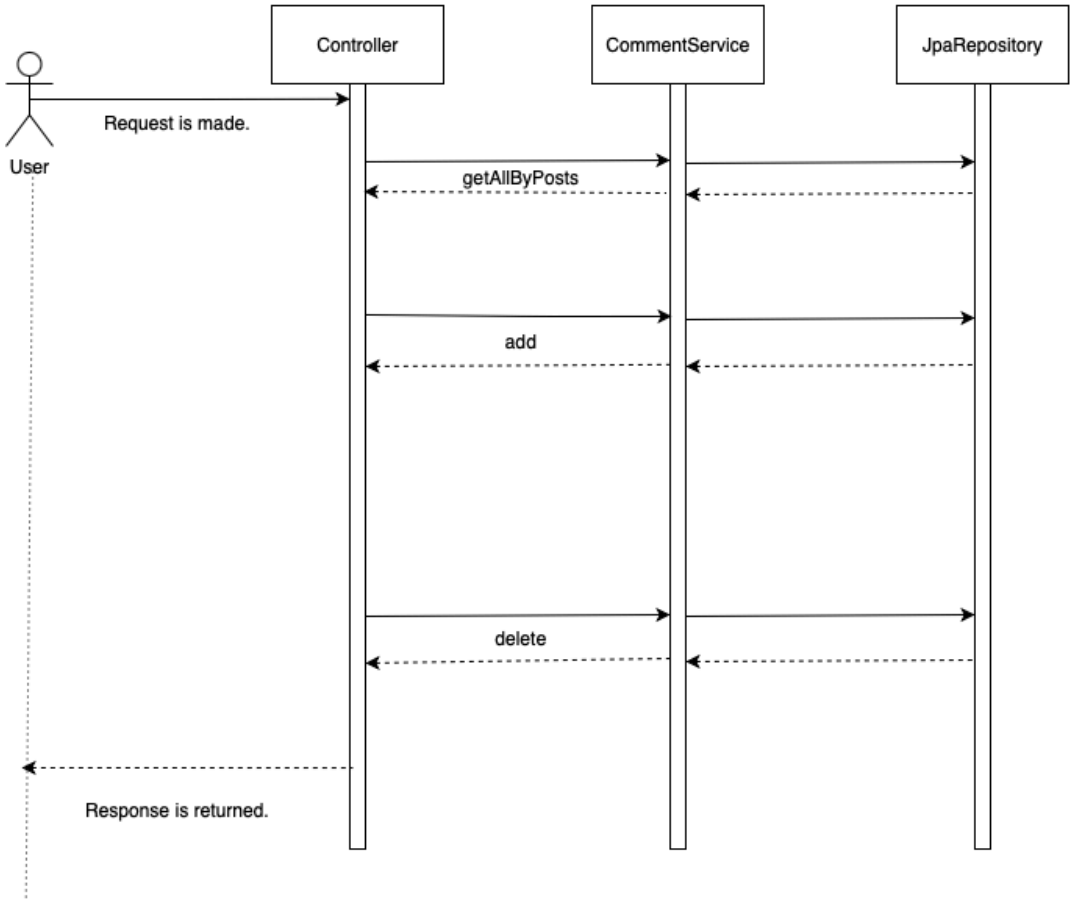
add: The user sharing the post, the post and the description of the post are taken as parameters and this comment is saved in the database.

delete: The id of the user and the comment that wants to take action is taken as a parameter. If this comment belongs to this user, the comment will be deleted from the database.

3.3 Dynamic Behaviour

CommentService consists of the management of the parts of the operations that the user wants to do in the database that affect the comment table and are affected by the comment table. After the user makes a request to the Controller, here the CommentService methods are called and the business rules are applied. Later, if business rules are passed, these calls are forwarded to the reposritory. The necessary information is taken from the comment table of database or recorded in the comment table of database.

3.3.1 Interaction Diagrams



Component: Book Service

3.2 Description

The system uses the Google Books api for books that the user interacts with. Book Service includes the rules of getting the books to be accessed from this API and adding them to Bookie's own database, and the rules for searching the books from the API and its own database.

3.2.1 Processing narrative (PSPEC)

If the book that the user wants to add is available in our database, the information is taken from here. However, if it is not available, a request is sent to google books API and the book's information is added to our database. The book is searched according to the title of the book. The user is expected to give the name of the book they want to access. Error management related to these, access to the details of the book are performed with the methods here.

3.2.2 Interface Description

System's inputs are the book id and the title of the book. Its output is book object and the list of book objects.

3.2.3 Processing Detail

3.2.3.1 Design Class hierarchy

It is a single class and uses the book repository, which belongs to the book entity, although it is independent from other components.

3.2.3.2 Restrictions/limitations

The book you want to process must be in the book table in the database or google books API.

3.2.3.3 Performance issues

Since services use repository, they are dependent on spring because they use Spring JPA Repository.

3.2.3.4 Design constraints

It is independent of other service components. It uses all its own attributes and functions and does not use an attribute belonging to another service class. Only it is dependent on its own repository. (Book Repository)

3.2.3.5 Processing detail for each operation

searchForBooks: It takes the title of the book as input and sends a request to the Google Books api, searches for books with this title and returns this book list.

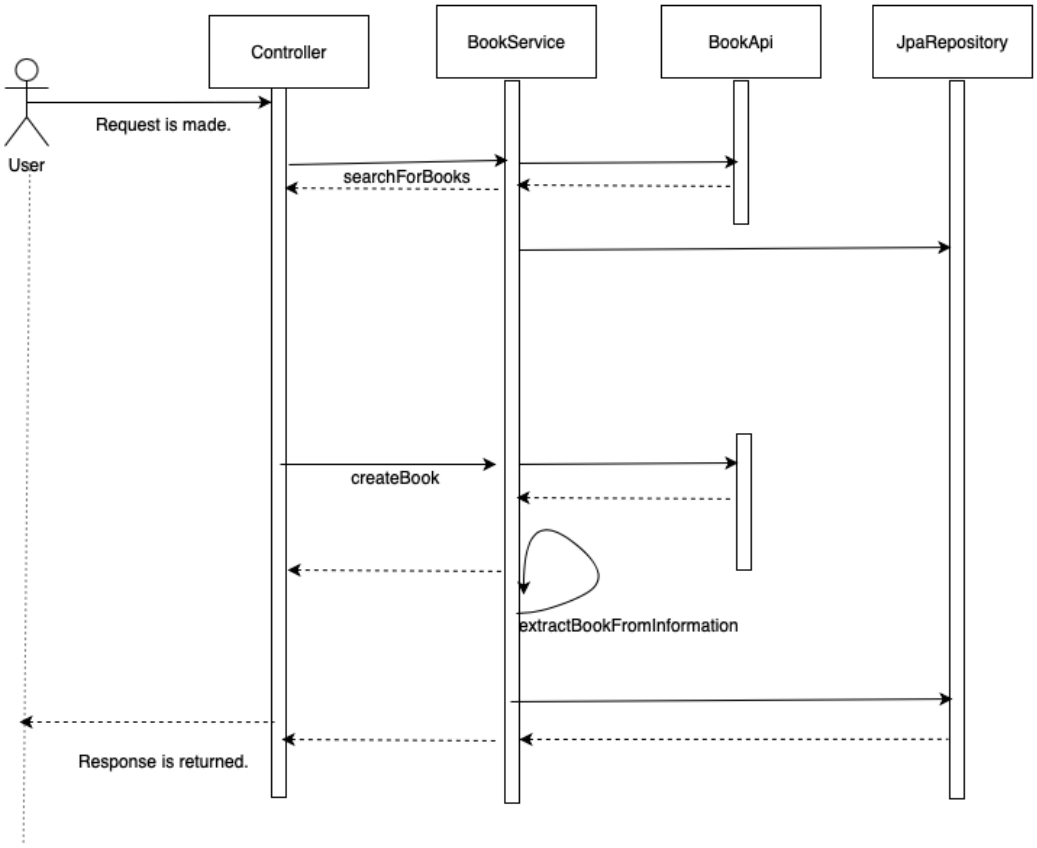
createBook: It takes the id of the book to be saved in the database and sends a request to api and reaches the response for the book. This answer is send to the exractBookFromInformation method of the same class.

extractBookFromInformation: The information that the application needs is taken from the response. And the attributes of the book object is set.

3.3 Dynamic Behaviour

BookService consists of the management of the parts of the operations that the user wants to do in the database that affect the book table and are affected by the book table. After the user makes a request to the Controller, here the BookService methods are called and the business rules are applied. Later, if business rules are passed, these calls are forwarded to the repository. The necessary information is taken from the book table of database or recorded in the book table of database. If this information is not present in the book table, a request is sent to google book api.

3.3.1 Interaction Diagrams



Component: Controller

3.2 Description

The controllers are used to communicate with the frontend/mobile side of the application. Frontend or mobile side of the application sends related information to the backend via controllers for running the business logic with them. Likewise, the controllers send refined data to the frontend/mobile side of the application.

3.2.1 Processing narrative (PSPEC)

Controllers work with endpoints that are unique. For each endpoint, it is possible to receive and send data to work on or to deliver what it is need. It is essential to use because the data comes from frontend/mobile should not reach the database directly for security reasons and the data should not send from any other layer which prevents the interfering of layers' work. Received data is transferred to the service components which contains the business logic of the backend system. And the data that comes out from services are sent to controllers in order to complete last actions like data manipulation, data transferring between objects etc.

3.2.2 Interface Description

An endpoint should be a defined path that is unique. The endpoints must follow HTTP request methods like GET, POST, PATCH, PUT, DELETE etc. For the endpoints that are aimed to modify data uses request bodies which contain the input to process. An endpoint may be specified with path variables like the variables that can be used to determine which data to work on. Also query parameters are generally used to choose options like how to manage data. The endpoints are formed and called API (Application Programming Interface) as mentioned in the introduction section.

By using data transfer objects (DTOs) the data can be converted to the entities that are managed by services and vice versa. In the vice versa case they become resource objects to return processed data.
Controllers store service objects to transfer data to the inner side of the backend which is the service layer that manages business logic.

3.2.3 Processing Detail

3.2.3.1 Design Class hierarchy

There is a separate controller class for each of the services that exist but as their goals define the same thing, they do it is defined as a component. A controller has endpoints which enable HTTP requests to come into the backend and because of that it is the leaf layer of the backend system. Converted data is sent to the related

service which modifies the data to the business logic that is applied. Also, the processed data is sent back to the same controller and then it is sent as response back to the caller of the endpoint.

3.2.3.2 Restrictions/Limitations

The endpoints of the controllers should be unique for all endpoints in API. And the services that are used to process data and use business logic must be stored as a variable and hence should not be used for them to use a singleton pattern.

3.2.3.3 Performance issues

Because of the conversion of data to which business logic accepts may decrease the performance of the application. Also, by using a layered application and not accessing the database directly decreases the performance but it is necessary to do for security issues.

3.2.3.4 Design constraints

The biggest design constraint is that a controller must connect to a layer that contains business logic which is a service layer. It is not safe to connect the controller to a database directly because it gives huge sovereignty to the caller of the endpoint which is a security issue to handle.

3.2.3.5 Processing detail for each operation

Operations of AuthController:

- login: Redirects related login information with email and password to auth service. Returns JWT if authentication is ensured.
- register: Redirects related register information like full name, email, phone number etc. to auth service. Returns registered user information.

Operations of BookController:

- searchForBooks: Redirects related book title book service to find requested books in Google Books. Returns list of most accurate books with their information.

Operations of PostController:

- updatePostStatus: Updates the status of the post that is created by a user. The status message can be freely determined by the endpoint caller. Returns an updated version of the post that is posted.

Operations of UserController:

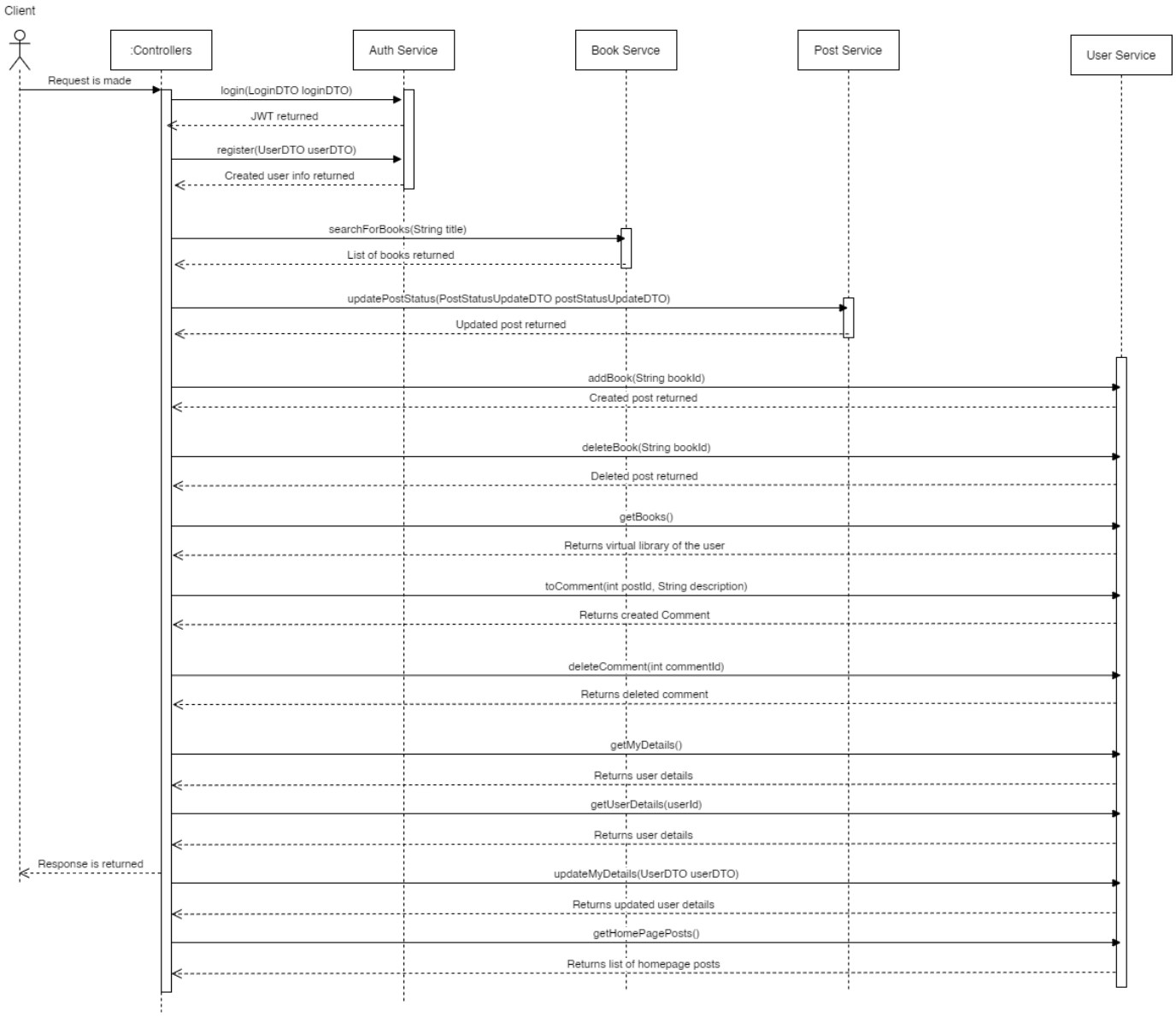
- addBook: Redirects related book id information to its service for adding it to the user’s virtual library. If the book does not exist in our database then firstly it is created with another service call and then it is added to the user's virtual library. Returns the post that is created.

- deleteBook: Redirects related book id information to its service for deleting it from the user's virtual library. Returns the post that is deleted.
- getBooks: Redirects related call to its service for returning currently authenticated user’s virtual library.
- toComment: Redirects related call to its service for commenting to a post for currently authenticated user. Returns the comment that is made.
- deleteComment: Redirects related call to its service for deleting a comment from a post that is owned by the currently authenticated user.
- getMyDetails: Redirects related call to its service to return currently authenticated user’s full details.
- getUserDetails: Redirects related call to its service to return another user’s non-restricted details.
- updateMyDetails: Redirects the call to the service for updating currently authenticated user’s details. Returns updated details of user.
- getHomepagePosts: Redirects the call to the service for getting homepage posts for exploring purposes. Returns a list of the user's virtual libraries.

3.3 Dynamic Behaviour For Controllers

Controllers are generally used for redirecting endpoint calls to layers which contain business logic. Calls are made from the frontend/mobile side of the application which contains the client gui and client business logic. The calls are redirected to the services for processing data and the processed data is returned to the client via controllers with the same endpoint that the call is made.

3.3.1 Interaction Diagrams



Component: Security

3.2 Description

The security of the application is provided with the usage of JSON Web Tokens (JWT) and hashed passwords with bcrypt encoder. Any http request that has been made to the backend system other than registration and login functionalities are secured and requires an authorization header which contains the JWT.

3.2.1 Processing narrative (PSPEC)

Based on the secret key defined in our application a token is generated with the dependent factors like the email of the user, the timestamp that is tried to login at that time and the signature algorithm used for SHA256 hash function. The created token is returned to the user for later usages of other endpoints. When a call is made to the secured endpoint the filter internal process begins. The authorization header is taken and reviewed if it contains the JWT. If it contains it, the email of the user is extracted

from it. After that if the related user exists in our database, the program checks if it's details like the email and password are matching with the information that is extracted from JWT. If it is, authentication is applied and the call to the endpoint starts from the related controller.

3.2.2 Interface Description

The JwtUtil class is responsible for the creation and the validation of the token itself. The generation process happens with the dependent factors like the email of the user, the timestamp that is tried to login at that time and the signature algorithm used for SHA256 hash function. The validation process happens by matching the email of the user within the database and in the JWT.

JwtRequestFilter class collaborates with JwtUtil class for validating the JWT. If the token is deemed to be appropriate for access to user-based levels, JwtRequestFilter class simply sets authentication related configuration for Spring Security.

UserDetailsServiceImpl class is simply responsible for loading users with email information that is in JWT from the database and giving it authority roles for determining which levels should that user access for endpoints. Currently there is only ROLE_USER exists because no admin endpoint is determined. If any admin endpoint will be required there will be a role for admin as well.

There exist SecurityConfiguration class, but it will be mentioned in its own component part.

3.2.3 Processing Detail

3.2.3.1 Design Class hierarchy

For the login process, the JwtUtil class of the component works for generating the JWT. For any other http request that is made to secure endpoints first arrives JwtRequestFilter class which does a filter operation and validates the token. In validating operating UserDetailsServiceImpl class is called for retrieving the user from database and it's authorities.

3.2.3.2 Restrictions/Limitations

The token must be sent via authorization header with Bearer prefix in order the filter & validate operation work as expected. Any other type of sent will not be accepted and therefore fails the security part of the backend system.

3.2.3.3 Performance issues

The performance is going to be affected in a negative manner because for each of calls that is made to secure endpoints should be validated with internal filters and this may cause some amount of time to resolve.

3.2.3.4 Design constraints

The tokens must have an expiration date in order to prevent unlimited access with just a token to our systems. So some checks will be placed to check if the JWT is

expired or not even if it is a valid token. If the JWT is expired, users should use login functionality to get a new JWT.

3.2.3.5 Processing detail for each operation

Operations of JwtUtil:

- extractUsername: Extracts the email in the claim that is inside in JWT.
- extractAllClaims: Extract all claims for JWT.
- generateToken: Takes user details and based on the details it creates the token. Returns the token that is generated.
- createToken: Takes user email and based on the email, the timestamp that the request is made, secret key and the signature algorithm defined as SH256 hashing it creates a unique token. Returns the token that is created.
- validateToken: Validates token by extracting from JWT and comparing it with the user details. Returns boolean like true or false.

Operations of JwtUtil:

- doFilterInternal: Finds the mail of the authorized user and tries to validate the token with the user details that comes from UserDetailsServiceImpl class's loadUserByUsername by calling validateToken method.

Operations of JwtUtil:

- loadUserByUsername: Loads user details with receiving email of the user by accessing the database. Also adds granted authorities to access the endpoint that is determined with levels.

3.3 Dynamic Behaviour For Security

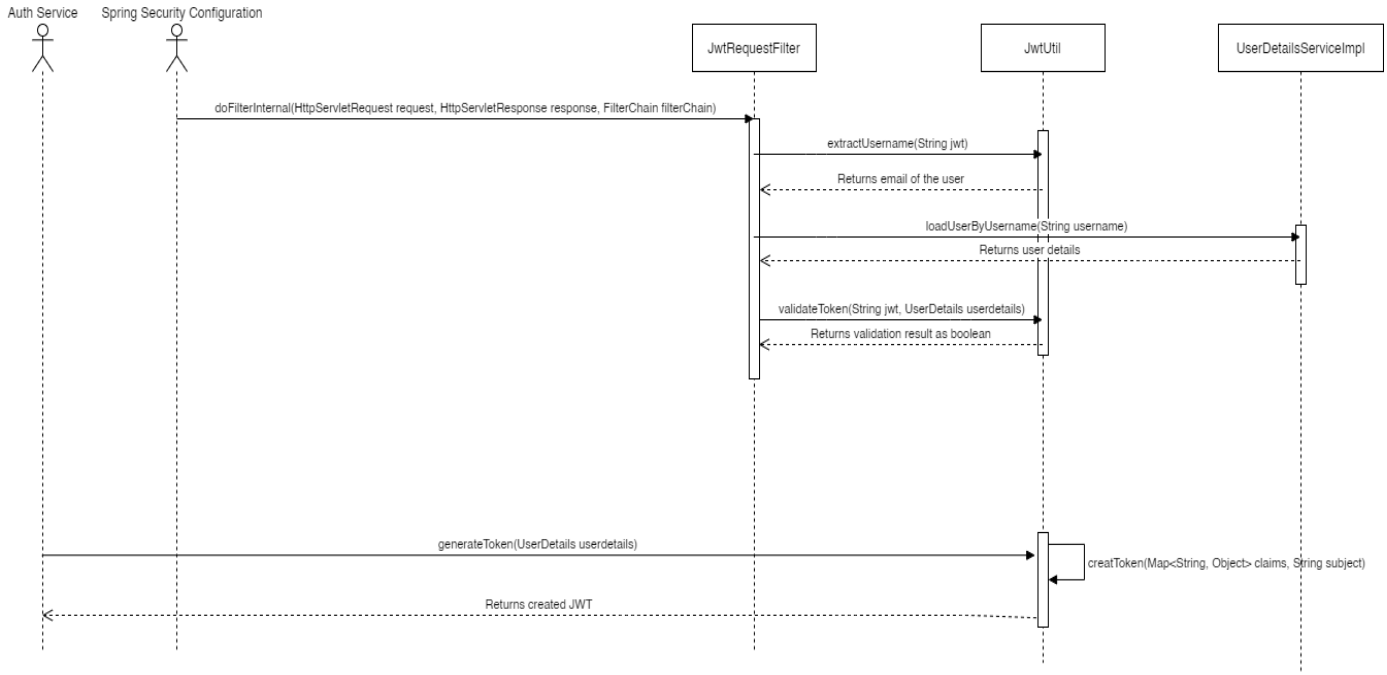
As mentioned in the interface description section of this component, JwtUtil class for the creation and the validation of the token itself. After a login process the token is created and returned via this class's methods.

JwtRequestFilter class collaborates with JwtUtil class for validating the JWT. If the token is deemed to be appropriate for access to user based levels, JwtRequestFilter class simply sets authentication related configuration for Spring Security.

UserDetailsServiceImpl is responsible for loading users with email information that is in JWT from the database and giving it authority roles for determining which levels should that user access for endpoints.

Any of the call comes from SecurityConfiguration component which has configuration settings for this component.

3.3.1 Interaction Diagrams



Component: Spring Security Configuration

3.2 Description

Spring security configuration component handles the security component of the system. It is connected to the classes explained in the security section.

3.2.1 Processing narrative (PSPEC)

It is responsible for authentication manager builder's user detail service's creation, the endpoints which are exposed to all users, which are exposed to authenticated users and which requires admin authentication defined here. Also, JwtRequestFilter class is enabled through the configuration method.

Also, CORS configuration is handled, though it simply allows everything for origins methods and headers for the production phase of the system. This will be reconfigured before when our system goes live.

3.2.2 Interface Description

There is only one class in this component called SecurityConfiguration. It extends WebSecurityConfigurerAdapter of the Spring's predefined Interface for security configuration.

There are two methods named configure; first one configures user details service by creating and setting field by taking authentication manager builder, second one takes HttpSecurity object and defines how the levels of the endpoints should exposed to

users and add filter as mentioned JwtRequestFilter class's instance for calling the relevant methods from completing security process.

One method configures cors configuration by returning a CorsConfiguration object. PasswordEncoder method simply returns a BcryptPasswordEncoder instance for encoding purposes. Last method creates an authentication manager bean for managing the authentication process.

3.2.3 Processing Detail

3.2.3.1 Design Class hierarchy

A call made to an endpoint firstly arrives security configuration and the call is redirected to JwtRequestFilter class. Security configuration class extends WebSecurityConfigurerAdapter class which is a template for security configuration class provided by Spring Security.

3.2.3.2 Restrictions/Limitations

Every endpoint should be stated in configuration method because the endpoint that is not listed will be unreachable. If there is no exceptiıon that endpoint is defined, wildcards can be used to not list all endpoints but if there are exceptions whether they are small or big the endpoints should be stated one by one not using wildcard , because wildcards states on or more than endpoints which may create a leveling problem.

3.2.3.3 Performance issues

For each endpoint, there will be a check for currently authenticated user's authorization roles that fits the level requirements of that endpoint. This creates an extra step for the call made and therefore may decrease the performance.

3.2.3.4 Design constraints

Multiple configuration methods must be created because each configuration takes different types of objects to work on. Therefore method overloading is used to make configurations of the security system.

3.2.3.5 Processing detail for each operation

Operations of SecurityConfiguration:

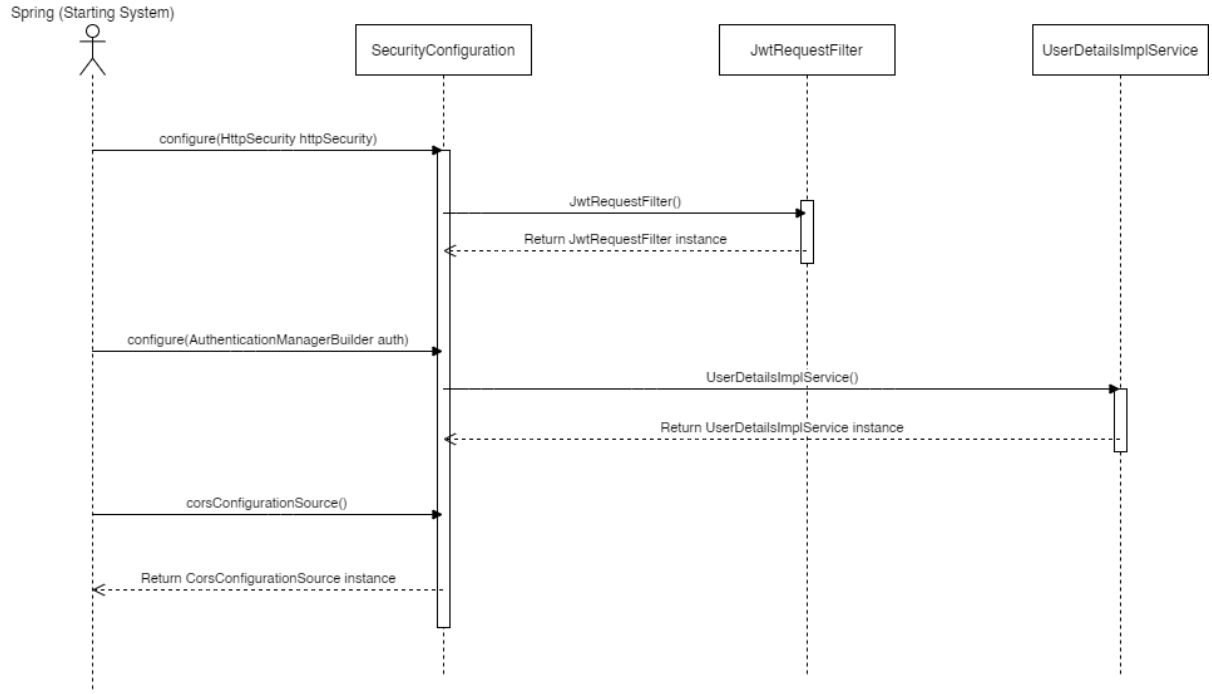
- configure: Takes AuthenticationManagerBuilder object and sets user details service for feature use.
- configure: Takes HttpSecurity object and configures CORS and CSRF configurations, determines the levels of the endpoints based on the roles of the user and adds filter named JwtRequestFilter for doing internal filters and validating the token that comes from the client.
- corsConfigurationSource: Configures the CORS policy options. Though it simply allows everything for origins methods and headers for the production phase of the system. This will be reconfigured before when our system goes live. Returns the configuration of the cors object
- passwordEncoder: Returns a spring bean that is named BcryptPasswordEncoder for encoding password purposes.

- authenticationManagerBean: Returns an authentication manager bean.

3.3 Dynamic Behaviour For Spring Security Configuration

There is only one class named SecurityConfiguration which extends WebSecurityConfigurerAdapter interface that has predefined method skeletons for the important configuration methods. The configuration class is annotated with @Configuration annotation which simply tells Spring to read and start the system based on the options that are defined.

3.3.1 Interaction Diagrams



Component: Persistence Layer (JPA)

3.2 Description

Persistence layer transforms plain old java objects (POJOs) to the records that are inside tables. The objects and the table records are not in the same format, so this transform is needed because of object-relational mapping (ORM).

3.2.1 Processing narrative (PSPEC)

JPA uses entity manager or persistence manager to handle the mapping between POJO's and database records. For the POJO's that takes place in a system called entities. Those entities include only the fields that are going to be mapped in the table. And also, there can be another mapping between the tables that can take place and shown with fields. There is no business logic or whatsoever takes place in entities. The annotations that are placed above the fields specify the columns in the database tables and the relation between the tables.
Entity manager looks at the annotations and maps them to related tables and columns when a transfer happens between the system and the database.

3.2.2 Interface Description

The columns that are specified with notations should have information like column name, required, nullable etc. and the notation for them is @Column annotation. Also, ID's can be specified with @Id annotation, the generation of the ID's can be specified with @GenerationType. The relations between the tables can be reflected to POJO's by annotating them with @ManyToMany, @ManyToOne/@OneToMany and @OneToOne annotations. As the inputs specified, the output comes from entity manager which is a transformed version of the POJO's into the table records.

3.2.3 Processing Detail

3.2.3.1 Design Class hierarchy

Entity manager class has methods to read annotation information from entity classes and map it to the database tables as records. Entity manager is called from data access objects which aims to access databases through a connection.

3.2.3.2 Restrictions/Limitations

JPA opens a connection to the database and makes transactions. The connection cannot be opened for a long time because the database expects operations to handle. The connection must be closed in order to persist the data changes. This gives a need to open/close connection phases when data is wanted to persist in the database.

3.2.3.3 Performance issues

The mapping phase of the POJO's into table records takes time. Also opening connection and closing phases will take some time too. Therefore, performance may decrease but these are necessary steps for proceeding.

3.2.3.4 Design constraints

JPA only understands what entity to map which table by annotations. Any other configuration than that will result in unexpected mapping errors which can damage the database.

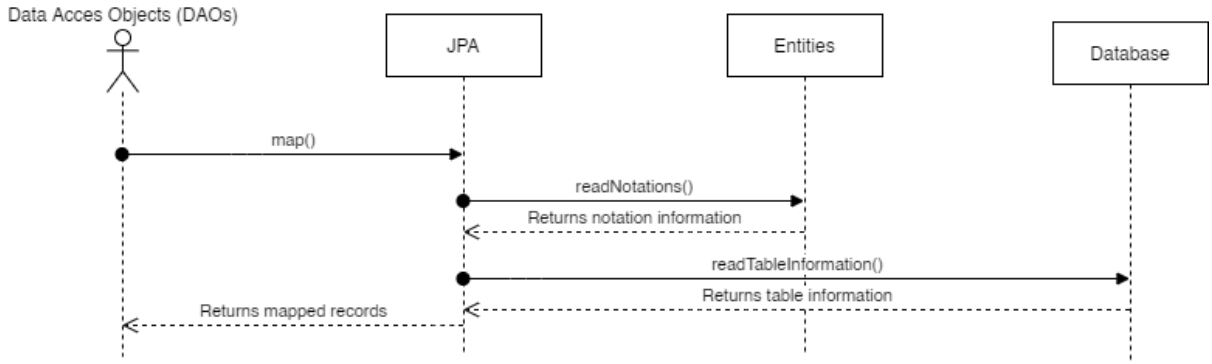
3.2.3.5 Processing detail for each operation

- Defining Mapping Details: This should be done via annotations to the entities (POJOs) that are defined in the system. The annotations are explained in the interface description section.
- Data Persistence Operation: A connection to the database is opened and entity manager maps entities to table records. After creating table records are sent to the database and the connection is closed.

3.3 Dynamic Behaviour For JPA

Entity classes are defined in the system to manage information through the system. JPA is called through data access objects and it reads mapping information from entity classes and persists the data in the database with a connection.

3.3.1 Interaction Diagrams



Component: Database (PostgreSQL)

3.2 Description

The database stores all the information that is processed. It stores within tables and there are columns inside tables to classify the information to its needs. The data types may differ from information needs to need but an information must be unique and can be identified with primary keys. The tables are connected to each other with foreign keys.

3.2.1 Processing narrative (PSPEC)

Information must be stored with records inside tables. The basic operations are like insert, select, update and delete modifies the records based on the requests that come from the backend system. Those basic operations can be done via queries. Basic queries can get together and create more complex queries which the backend system asks for.

3.2.2 Interface Description

The inputs are the queries that come from the backend system via a connection. Those queries are executed and if they result in an output it will reflect it via that connection. And if it does not result in an output then it may modify the records inside databases as requested. The connection opened between the data access object of the backend system and the PostgreSQL database must be closed after the transactions are done.

3.2.3 Processing Detail

3.2.3.1 Design Class hierarchy

Tables are responsible for holding the records and with the help of the database shell it is possible to communicate between the backend system and the database. A connection must be opened between the data access object of the backend system and the PostgreSQL database.

3.2.3.2 Restrictions/Limitations

The records in the database must hold for some restrictions like uniqueness, check constraints etc. The queries that have been executed must not violate the conditions of related tables. Also, the backend system cannot access the data directly and a connection must be opened between them.

3.2.3.3 Performance issues

By default, the columns are not indexed and for some columns that are having transactions frequently may get more performance when they are indexed. Also dividing the data into tables will result in most of the queries to join tables which the repetition will multiply the time to reach the data. This will negatively affect performance property.

3.2.3.4 Design constraints

Multiple tables should be created in order to classify what the received data belongs to. This is necessary because holding the information in just a table kills the performance. All the records that are not necessary for an operation will be tried to return as output which decreases the performance hugely and is not a preferred way to design a database.

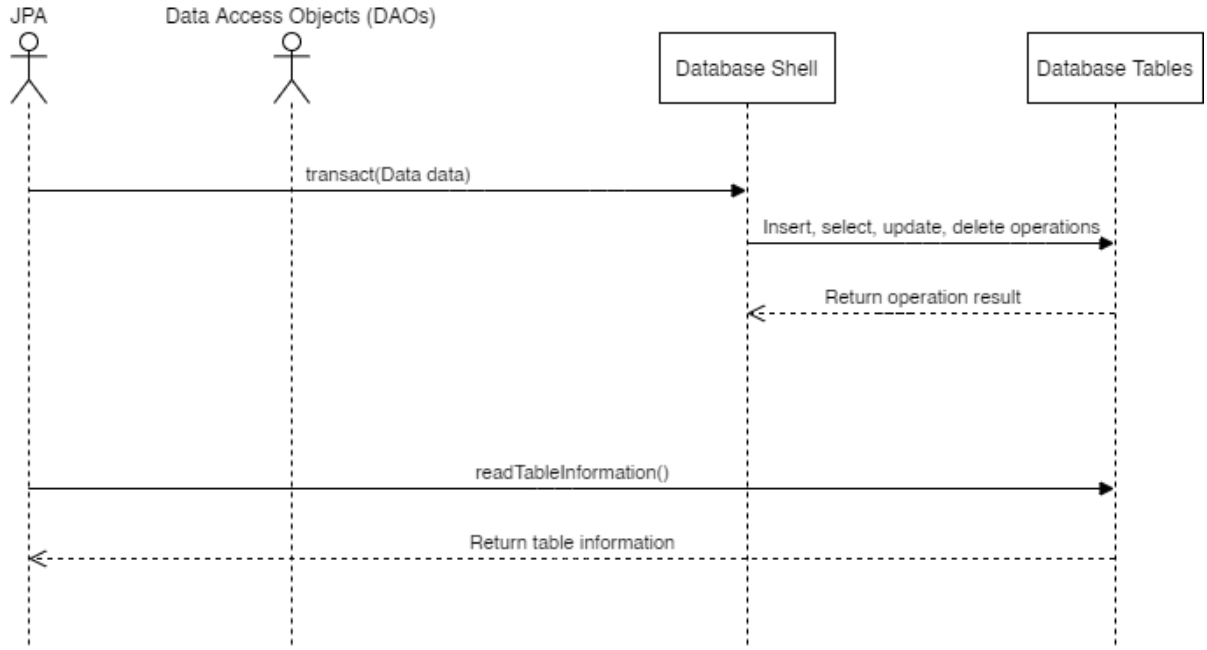
3.2.3.5 Processing detail for each operation

- Basic Operations: Basic operations like select, insert, update and delete can be performed over records via queries. More complex operations can be done with the combination of basic operations.
- Join Operations: By using join operations, the records can be joined within specified tables. The joined data can be modified or returned to the backend system.
- Table Management: The data within tables may require changes for the columns. The columns and their properties can be managed by table management.
- Backup - Recover Operations: The data can be corrupted or get lost when unwanted cases occur. Backup operations load the data within the databases and recovery operations recover the data from a backup file.

3.3 Dynamic Behaviour For Databases

The tables can get in interaction by join operations and the joined data can be mutated and returned to the backend system of the application. Other than table records, procedures, views and triggers get placed when wanting to mutate the data with different kinds of options.

3.3.1 Interaction Diagrams



Component: AuthService

3.2 Description

This service is responsible for authentication actions like register and login functionalities.

3.2.1 Processing narrative (PSPEC)

This service tries to login and register the unauthenticated user. Register method takes user information and creates a user in the database. The password that is entered is hashed (encoded) and then stored in the database.

Login method takes the user email and password and tries to match the password by hashing it and comparing it with the password that is in the database. If the password matches with the password that is in the database, the JWT is returned to the client for future use.

3.2.2 Interface Description

For registering, full name, phone number (must be unique), email (must be unique), password, the city that is living should be input parameters of the method. That information is processed as creation of the user. Output is the created user information.

For login, email and password information should be input parameters of the method. By using an authentication manager, email and password is tried to match with the

information in the database. If the authentication is successful a JWT is created through JwtTokenUtil class. Created JWT is returned back to the client for future use.

3.2.3 Processing Detail

3.2.3.1 Design Class Hierarchy

As each service is placed in the same layer, this service is generally called from auth controller class. The service access database by using a DAO layer which is a UserRepository for retrieving user information and inserting created user information. The outputs are redirected to the authentication controller.

3.2.3.2 Restrictions/Limitations

For the security reasons a generated user's information can't directly have the password what user entered. Therefore, an encode phase needs to be spent.

3.2.3.3 Performance issues

As mentioned in the restrictions/limitations part, for security reasons the password must be encoded and then stored in the database. This transformation of password takes some time as usual. Therefore, a transformation of password may decrease the performance.

3.2.3.4 Design constraints

Register method cannot return all the details of the user. The password is hashed and it is not wise to deliver a user to his/her hashed password. Therefore a DTO object should be returned with necessary information of the user.

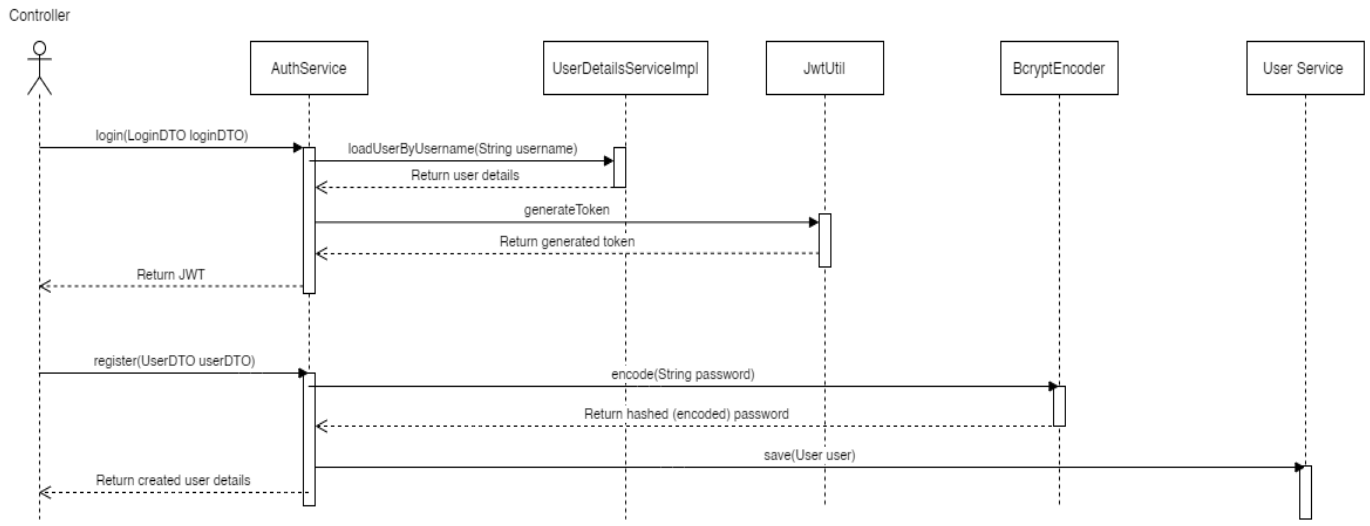
3.2.3.5 Processing detail for each operation

- login: Login method takes a login DTO object which carries email and password of the user. The password and the username are transferred to an authentication process of the authentication manager. If the hashed passwords are the same that means the user is authenticated and a JWT can be generated with JwtTokenUtil's generateToken method. A login resource is returned to the controller which contains the created JWT.
- register: Register method takes a user DTO object where it contains necessary information of the user. The user POJO is created and before returning the created user's information to the controller the password is hashed and stored.

3.3 Dynamic Behaviour For Authentication Service

This service retrieves login and register information from the authentication controller. Login method uses an authentication manager object to authenticate the user. If authentication is successful a JWT is created with the help of JwtTokenUtil class. For registering, BcryptEncoder class is used to hash (encode) a user's password and store it. Both methods access the database with UserRepository class which is a data access object.

3.3.1 Interaction Diagrams



4. Restrictions, limitations, and constraints

- The clients should have a reliable internet connection
- The backend should control the number of data it sends since we will not be implementing pagination
- We will not be able to deploy to iOS devices since we do not have Apple developer account which costs $100 per year

5. Conclusion

In conclusion we talked about the software design specifications of our software, the purpose, constraints and the architectural level design of the software. We also explained the Processing narrative, interface description and the component processing detail for each of our components

6. Contributions

		Talha	Zeynep	Ahmet
1.1. Purpose		+		
1.2. Statement of scope		+		
1.3. Software context		+		
1.4. Major constraints		+		
1.5. Definitions		+		
1.6. Acronyms and Abbreviations		+		
1.7. References		+		
---	-	---	---	---
2.1. Design Assumptions and Dependencies				+

2.2. General Constraints				+
2.3. System Environment				+
2.4. Development Methods				+
----------	-	----------	----------	----------
3.1. System Structure		+		
3.1.1. Architecture diagram		+	+	
3.2. Description for Component Client GUI				+
3.3. Dynamic Behavior for Component Client GUI				+
3.2. Description for Component DAO			+	
3.3. Dynamic Behavior for Component DAO		+	+	
3.2. Description for Component Controller		+		
3.3. Dynamic Behavior for Component Controller		+		
3.2. Description for Component Security		+		
3.3. Dynamic Behavior for Component Security		+		
3.2. Description for Component Spring Security Configuration		+		
3.3. Dynamic Behavior for Component Spring Security Configuration		+		
3.2. Description for Component Controller JPA		+		
3.3. Dynamic Behavior for Component Controller JPA		+		
3.2. Description for Component Database		+		
3.3. Dynamic Behavior for Component Database		+		
3.2. Description for Component Auth Service		+		
3.3. Dynamic Behavior for Component Auth Service		+		
3.2. Description for Component User Service			+	
3.3. Dynamic Behavior for Component User Service			+	
3.2. Description for Component Post Service			+	
3.3. Dynamic Behavior for Component Post Service			+	
3.2. Description for Component Comment Service			+	

3.3. Dynamic Behavior for Component Comment Service			+	
3.2. Description for Component Book Service			+	
3.3. Dynamic Behavior for Component Book Service			+	
----------	-	----------	----------	----------
4. Restrictions, limitations, and constraints				+
----------	-	----------	----------	----------
5. Conclusion				+