The Notes Application

System Specifications

[SE1 - 2020]

Group H:

Roman Shevchuk

Kiryl Volkau

Illia Manzhela

Mohd Wafa

M Hasibur Rahman

Instructor: Maciej Bednarz

Content

1. General Overview

1.1 Introduction

The main purpose of this document is to present a detailed description of the Notes application that we are going to develop. The paper will explain all the features of the system, user requirements as well as the internal, external functionalities, and non-functional requirements of the system.

A visual representation of the system:

The main idea is to create a flexible system to allow the user to manage his/her notes and to arrange them in a convenient way so that it can become a very handy tool in one's everyday life. The project is also portable for mobile devices since we are going to develop a mobile application. It will allow the user to easily and quickly access and edit his memos if the PC or a laptop are not available. Some small mobile widget displaying

recent notes will also be available and can make some positive impact on different life situations.

1.2 Scope of the project

This software system will be a typical note-taking application with a target audience ranging from students to average users who would like to store their ideas, memories or notes. This system will be designed to maximize the convenience of creating any of those by providing portable accessibility to the system at all times by using both a mobile and a web application. The system will meet the average user's needs of a private and instant note storing system with a bunch of additional features to extend the range of possibilities.

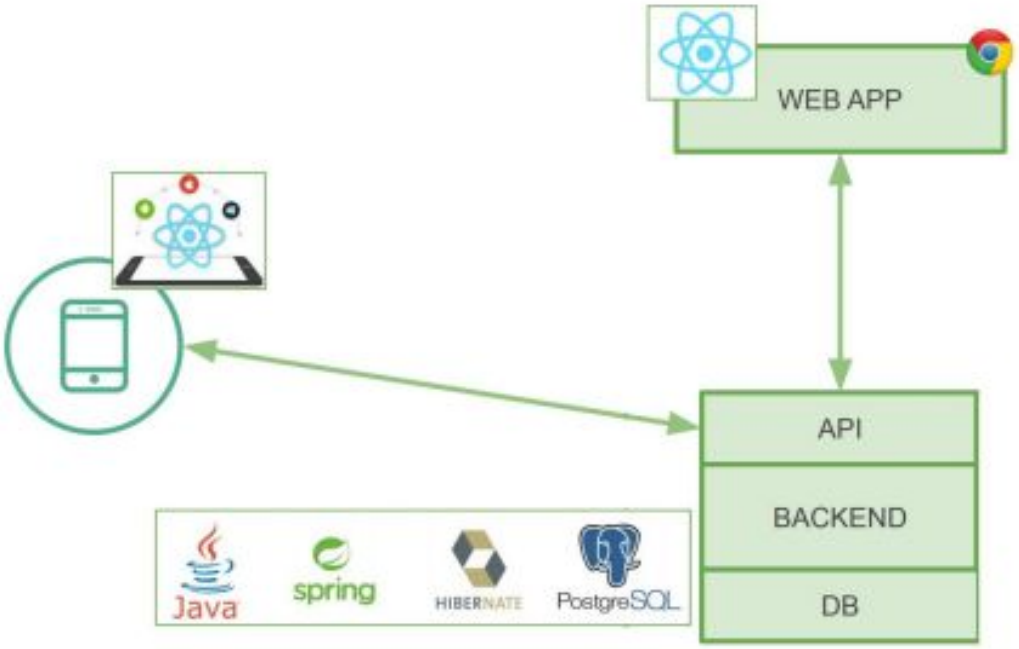
More specifically, the application will consist of 3 components:

- Web Application
- Mobile Application
- Backend Server

Web Application which will run on the client-side will serve for convenient user experience for those who use browsers. It will communicate with Backend Server through REST APIs and guarantee continuous connection to the source of data and possible actions until the user logs out. Client-side application will be written in ReactJS framework. Backend Server will be connected to the database and will handle all the requests for storing data or retrieving it and sending it back to the user. We will use the Java framework called Spring for the backend component. Mobile Application is assumed to be for Android devices and the React Native is assumed to be used as the framework for the development. Basically, Mobile and Web versions will share the same functions: display, filter, add, update and delete notes. Each note will also have a category, so after the creation, it can be added to any of the existing notebooks or a

new one. There also will be a possibility to pin a note, so that it will always appear at the top of any selections, being an important one. Each note can also be shared with the ‘outside world’ by the URL address which can be generated and used to access one specific note without being logged in. In this way, we add the possibility for users to share some notes with each other (in this case the accessed note is read-only). Users can also export and import the content of a note by pressing the button and choosing either the destination folder and name or just specifying the .txt file to get the note body from. The backend server will be exposing APIs so that both web and mobile clients can execute all the necessary operations. All the notes data will be stored in the database and will be used by the server to ensure both applications functioning. This design will allow users to execute desired operations related to notes and have an instantly updated system across multiple platforms.

The communication diagram and the technology stack:



2. System Requirements

2.1 System requirement overview

The application will be a permanent, multiple platform-based system allowing to create and manage notes. It has three components: Client-side application, Backend server, and a Mobile application. All of those are used to maintain a continuous real-time updated system that shows the latest modified and stored data on both PC and mobile devices. In order to get all privileges and be able to perform all the possible actions, a user should be registered in the system and logged in later on. Otherwise, he/she is considered as an anonymous user and is only capable of viewing shared notes.

2.2 Functional requirements

2.2.1 User stories

- As a user, I want to organize my notes, so that I can easily access them in less time.
    - Given notes, when a user puts a note in a notebook, he can later access it and all the other notes related to that notebook.
- As a user, I want to edit notes, so that I can update them whenever I want.
    - Given notes, when a user wants to update the content, he is able to change a specific note.
- As a user, I want to filter notes, so that I can view the notes that I am only interested in.
    - Given notes, when the user filters notes with specific identification, then he is able to view only those desired notes.
- As a user, I want to delete notes, which I do not need anymore or which are wrong.
    - Given a not needed note, when the user deletes a note, then it is no longer visible in the notebook.
- As a user, I want to display only the notes from one notebook.

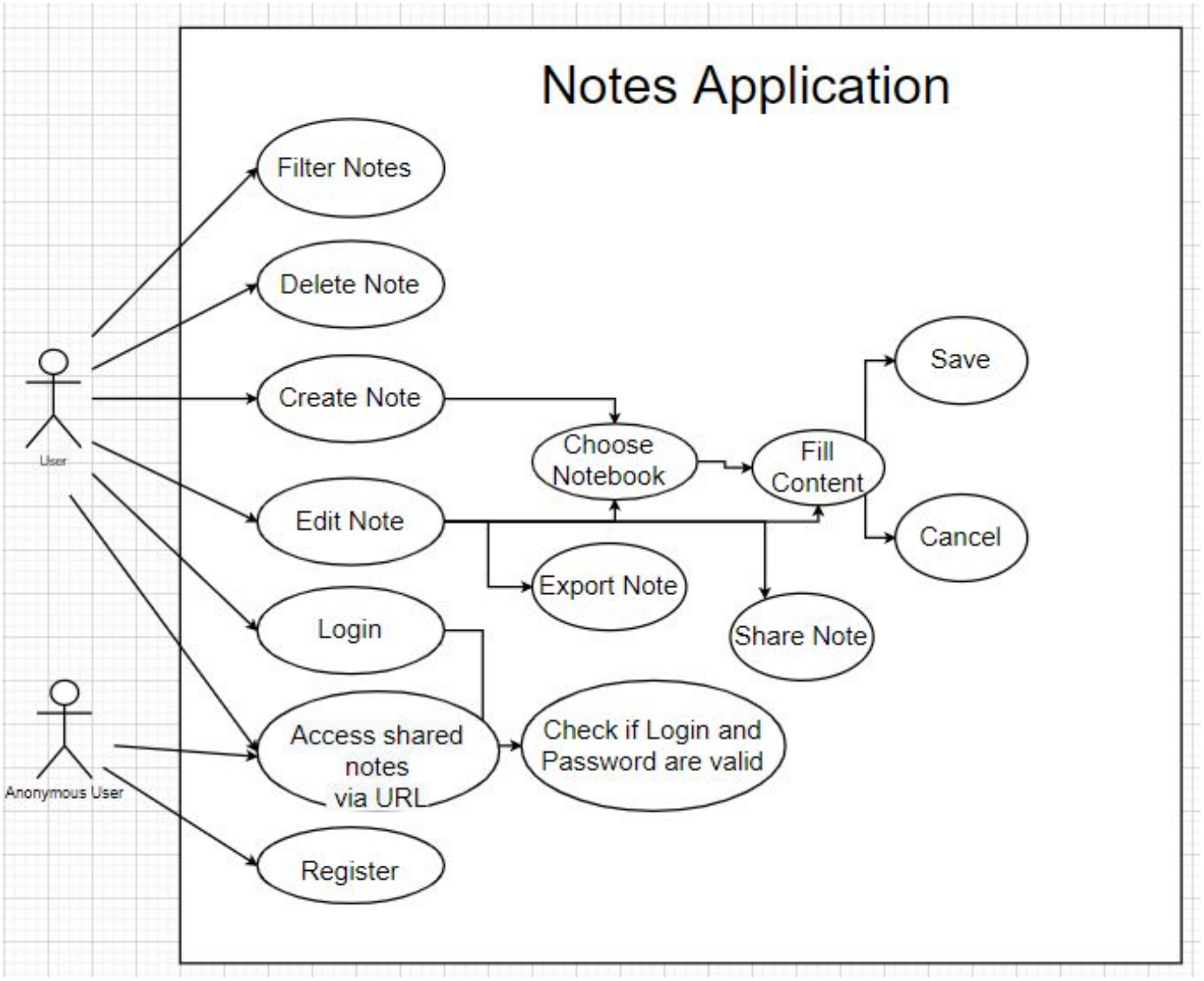
    - Given notes, when the user displays notes of a specific notebook, then only those notes are visible.
- As a user, I want to add notes in an existing notebook, so that I can easier access them later.
    - Given notes, when a user adds a note in a specific notebook, then it can later be found in that notebook.
- As a user, I want to export some notes as .txt files, so that I can have them saved on my computer.
    - Given notes, when the user exports a note, then that note is saved on the computer.
- As a user, I want to import some .txt files, so that I can have them in my notes.
    - Given notes, when the user imports a txt file, then that file gets saved as a note.
- As a user, I want to share some notes, so that some person would have access to it.
    - Given notes, when the user wants to share some notes, an URL address gets generated via which the note can be accessed.

2.2.2 Use Case Diagram

As we can see from the diagram, there are two actors which represent Anonymous User (not logged in) and User (logged in), one system called Notes Application, and many use cases. Let's dive into the cases. Since the app is about note-taking, their saving, and sharing, User can Create Note specifying one of the stored notebooks, Delete Note, Filter Notes which filters all the notes, Edit Note which contains content edition, and a possibility to either export the chosen note or import its content. The login case means logging in to some existing account, which triggers a password validation function, whereas Register means signing up. Finally, Access Shared Notes via URL represents a function that would generate a link that can be shared in order to

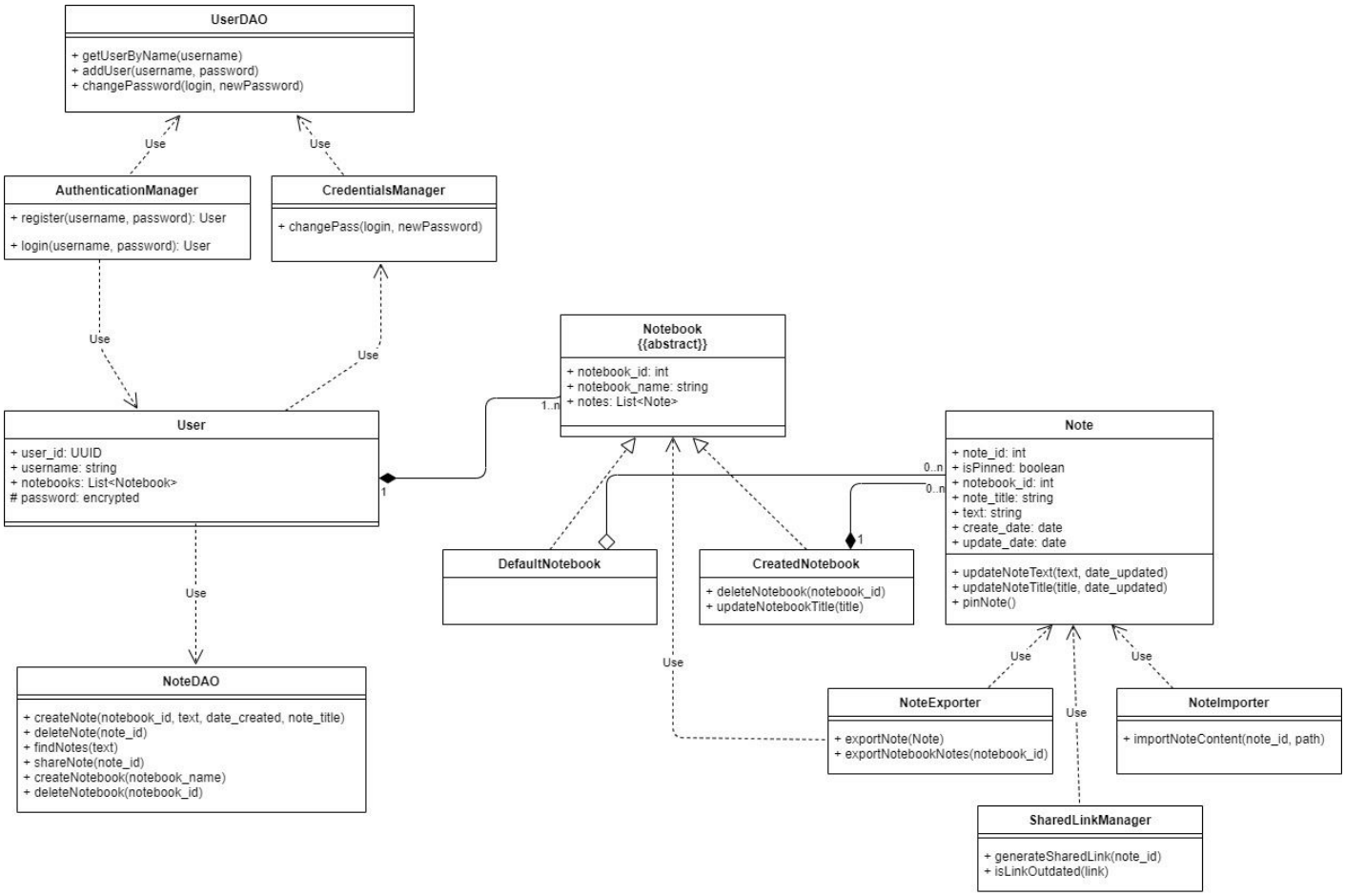
get read-only access to some notes without the need to have an account to make the service more delightful.

The user case diagram for our application :



2.2.3 Class Diagram

The user class diagram for our system is as follows :



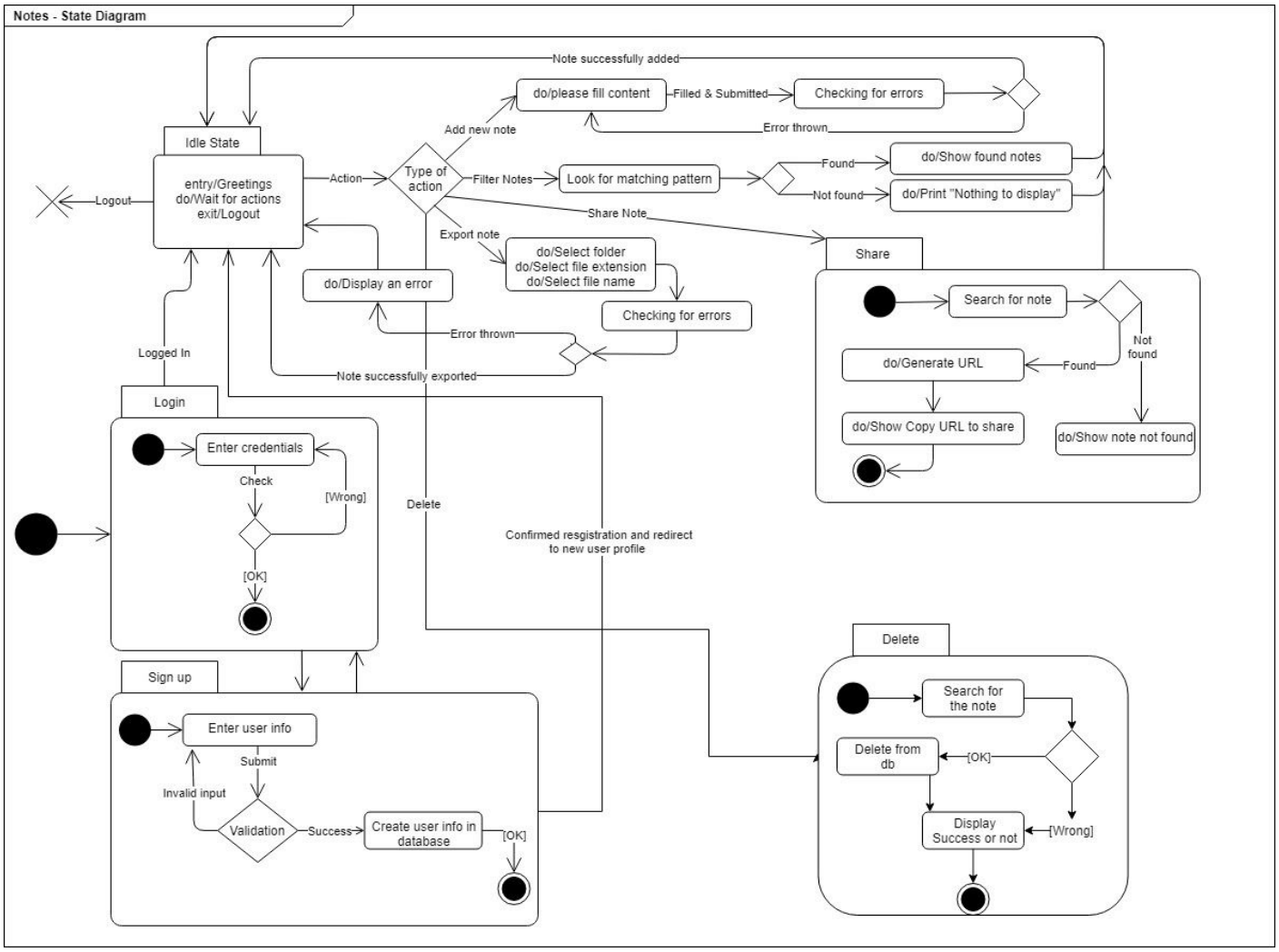
As seen from the diagram above, Notebook is an abstract class. The classes DefaultNotebook and CreatedNotebook have a realization relationship with NoteBook. DefaultNotebook is empty since it defines the notebook which exists by default for every user and thus contains no methods such as deleteNotebook(notebook_id), which is present in a CreatedNotebook. Thus CreatedNotebook and DefaultNotebook act as our subclasses.

Class Note is the most important part of the system and undergoes various operations and thus has many dependencies and relations. The NoteExporter, NoteImporter and

SharedLinkManager classes depend on Note since the methods of these classes will give a successful output only if the note exists in the database. The same principle applies between the NoteExporter and Notebook dependency relation. The note has an aggregation relation with DefaultNotebook and a composition one with CreatedNotebook since Note may or may not be in the default Notebook but a note cannot exist without a Notebook i.e. notes live and die with its associated notebook. Moreover, the same principle applies to the composition relation between Notebook and User that is, a Notebook cannot exist without a user.
Classes defined as DAO are classes that communicate with the Database. All the methods described in the DAO classes are the functionalities that a user is able to execute and thus have a dependency relation such that Users depend on NoteDAO. AuthenticationManager depends on User input to either register or allow the user to login. AuthenticationManager communicates with the UserDAO to perform an action according to user input i.e create or log in to existing user accounts. CredentialsManager also communicates with the UserDAO and upon successful changes in the database, the User is able to change their credentials.

2.2.4 State diagram

The state diagram for our application is as follows :



In the initial state for our application, the first transition is towards the login state, which asks for credentials and checks whether they are correct, and then it proceeds out of the page. The user also has the option to sign up for the application by entering the user info, and after validation, the user is created in the database.

In the idle state of the application, the user has the options to add a new note, filter notes, share notes, export, and delete notes. They also have the option to logout of the application, and by doing so, terminates the node.

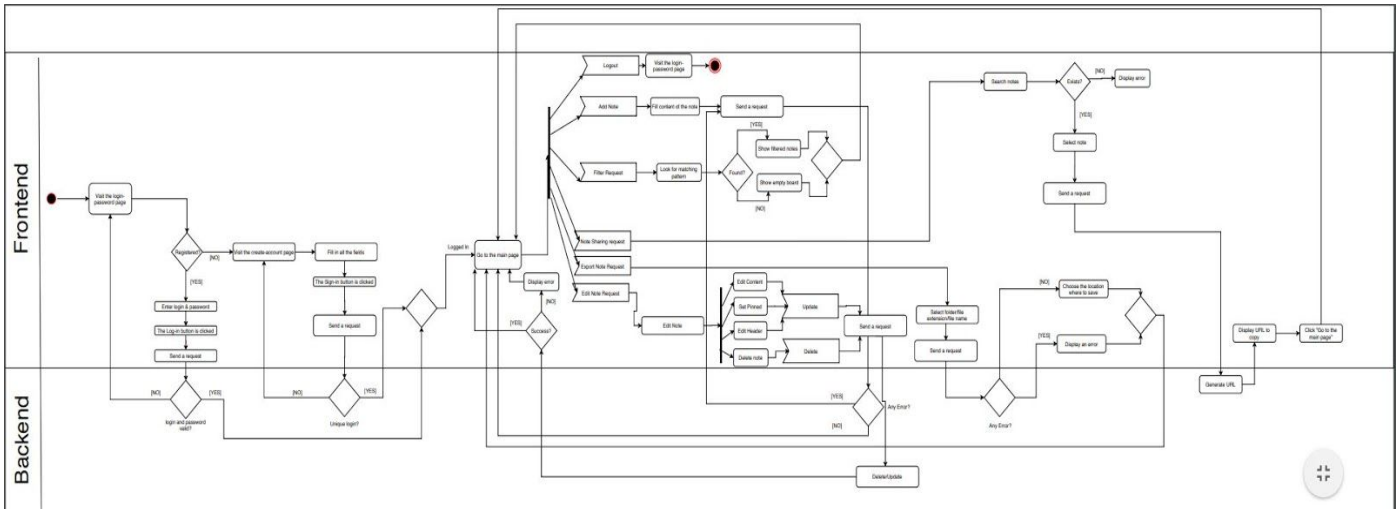
All the actions selected by the user, after completing them successfully, return to the idle state of the application. For adding a new note, the user first fills the content and checks for any errors in the content submitted. To filter notes, the application looks for matching patterns as entered by the user; if such a note is found, it displays them, otherwise, it prints the error " nothing to display ".

To share the note, the application searches for the note that the user wants to share, if the note is found, the application generates a URL and allows the user to copy the URL. If a note is not found, the error is printed, "Note not found ".

And finally, to delete a note, the state starts by searching for the note, finds it, and deletes it from the database. If the note is not found, it displays the error and ends the state.

2.2.5 Activity diagram

The activity diagram for our application is as follows :



Full PDF Diagram

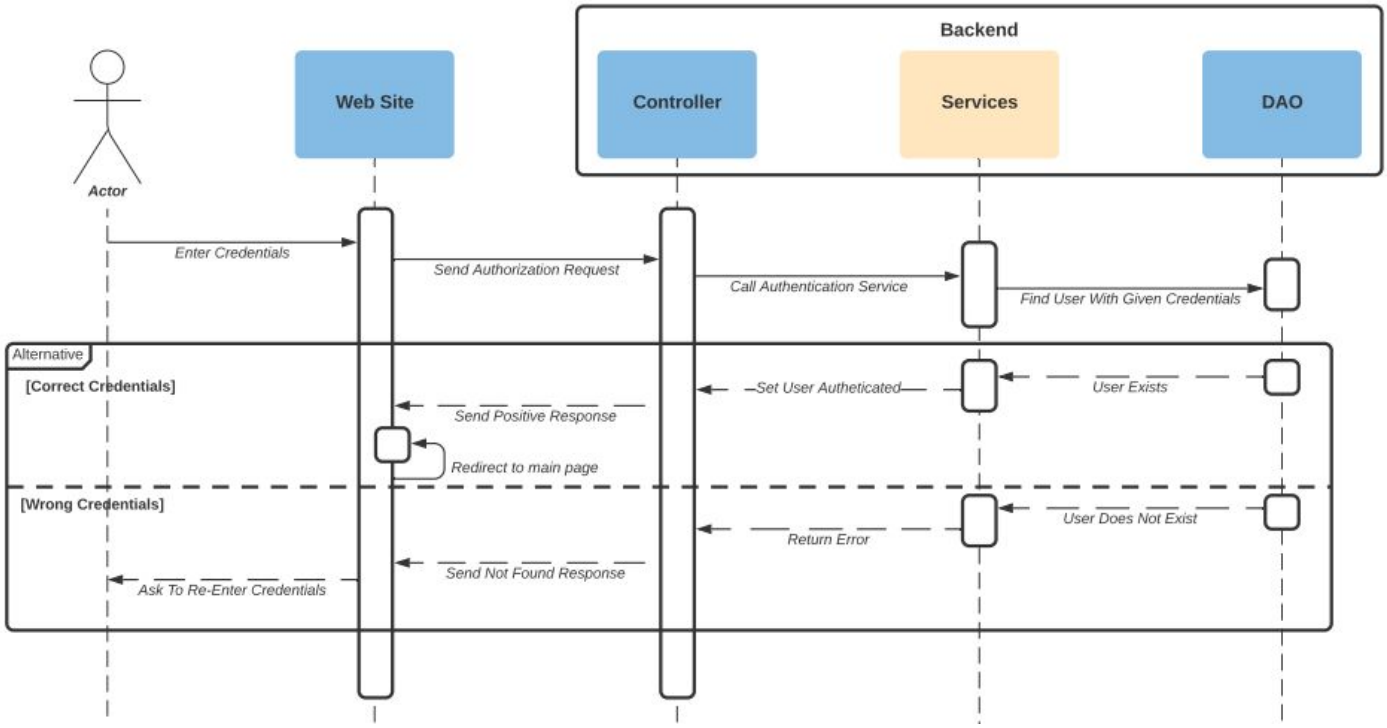
The above diagram refers to the steps involved in the execution of user cases. It allows us to understand the constraints and conditions for particular events. We have implemented the flow of actions that also occur in another partition i.e. the backend. From the initial node the first transition to the activity of visiting the login/register

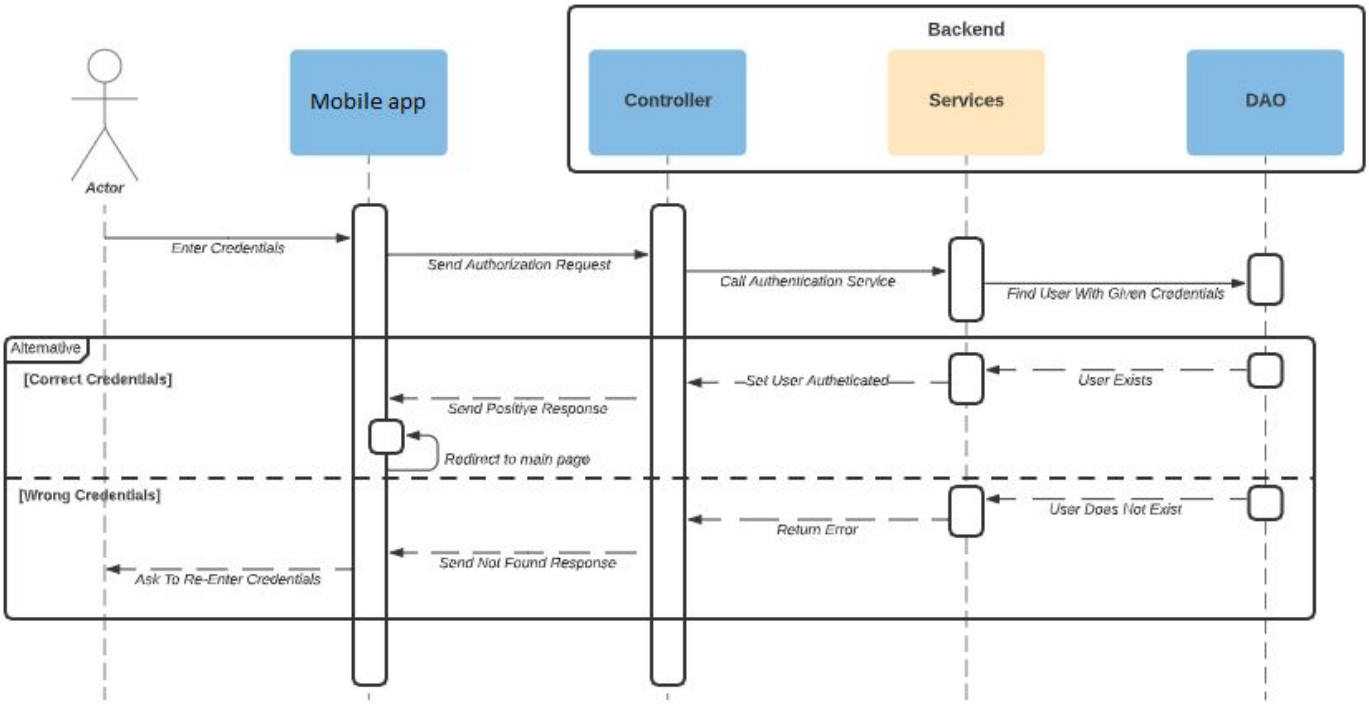
page. We then perform a decision node that controls flow according to whether the user is registered or not. In each of the control flow, the backend performs a decision and plays the role of the last activity that gives the verdict of failure or success of reaching the activity of being directed to the main page. Once on the main page there, users can decide to choose several activities and thus a fork node is used, and a concurrent flow of activities is described.
The concurrent activities such as, ‘Add node’ are defined as Accept (time) event action as they only will start executing when the corresponding event occurs. Let's take a look at one of the actions of the forked node - ‘Filter Request’. When Filter Request is initiated we first look for matching patterns or related tags of notes with the given request. We then show the filtered notes on success to the main page. On failure, we display an empty board on the main page. The redirection to the main page is done through a merge node that brings together the success or failure decision control flow of the FIlter Request.
Let's look at an accept time event action that involves the object flow to the backend - ‘Add note’. The added note must be stored in the database hence, in the activity diagram the control flow involves sending request to the backend, and upon success, the user is directed to the main page, and upon failure, a user is redirected back to the ‘Add Note’ object flow. Since our main page is in an idle state, the activity final node ensues only when users logout.

2.2.6 Sequence diagram

This diagram consists of five different specific diagrams that cover the different functionalities of the project. In all the diagrams there are 4 different object symbols that specify the Web site, Controller, Services, and DAO. The latter three belong in the Backend.

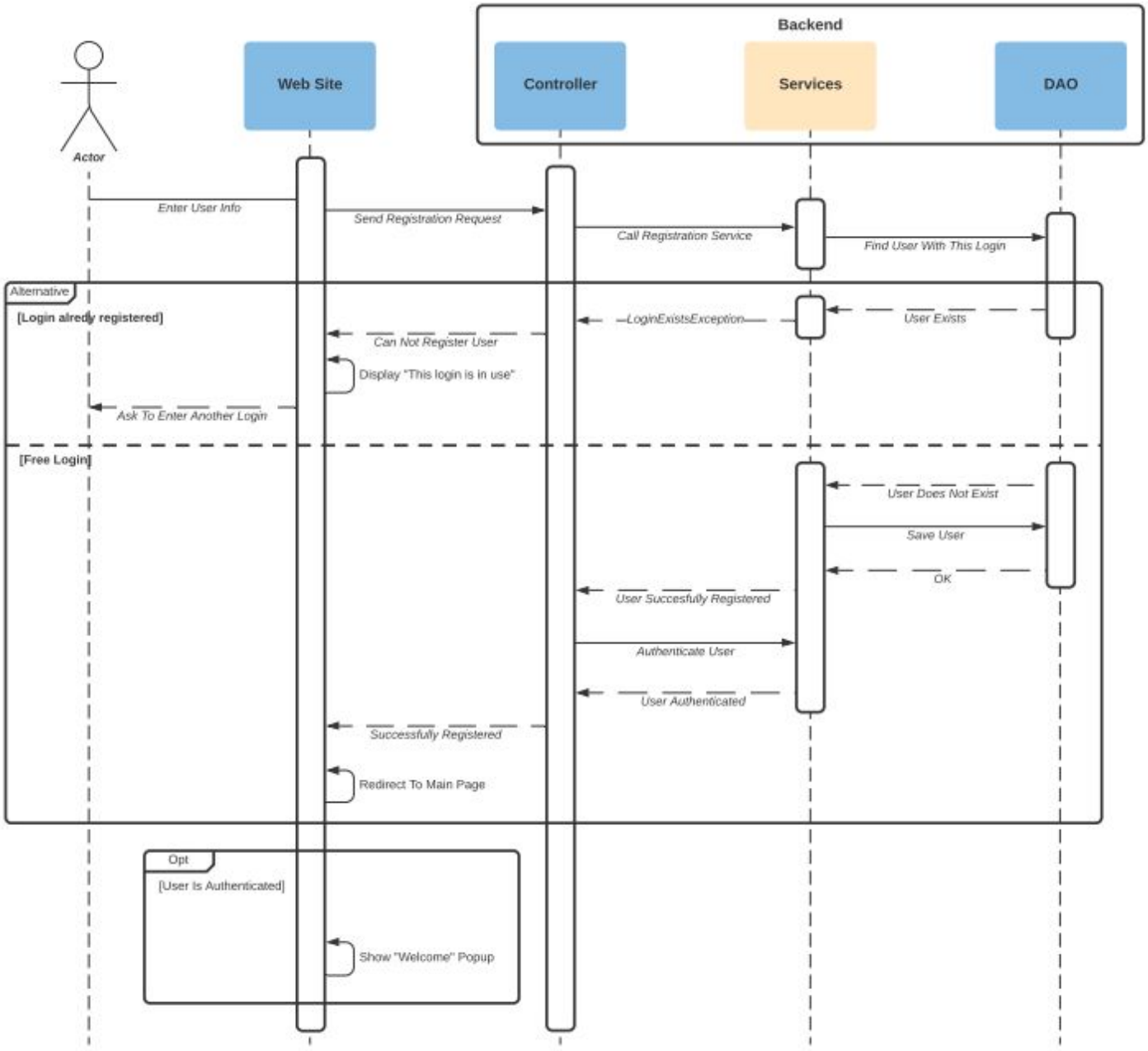
Login Diagram

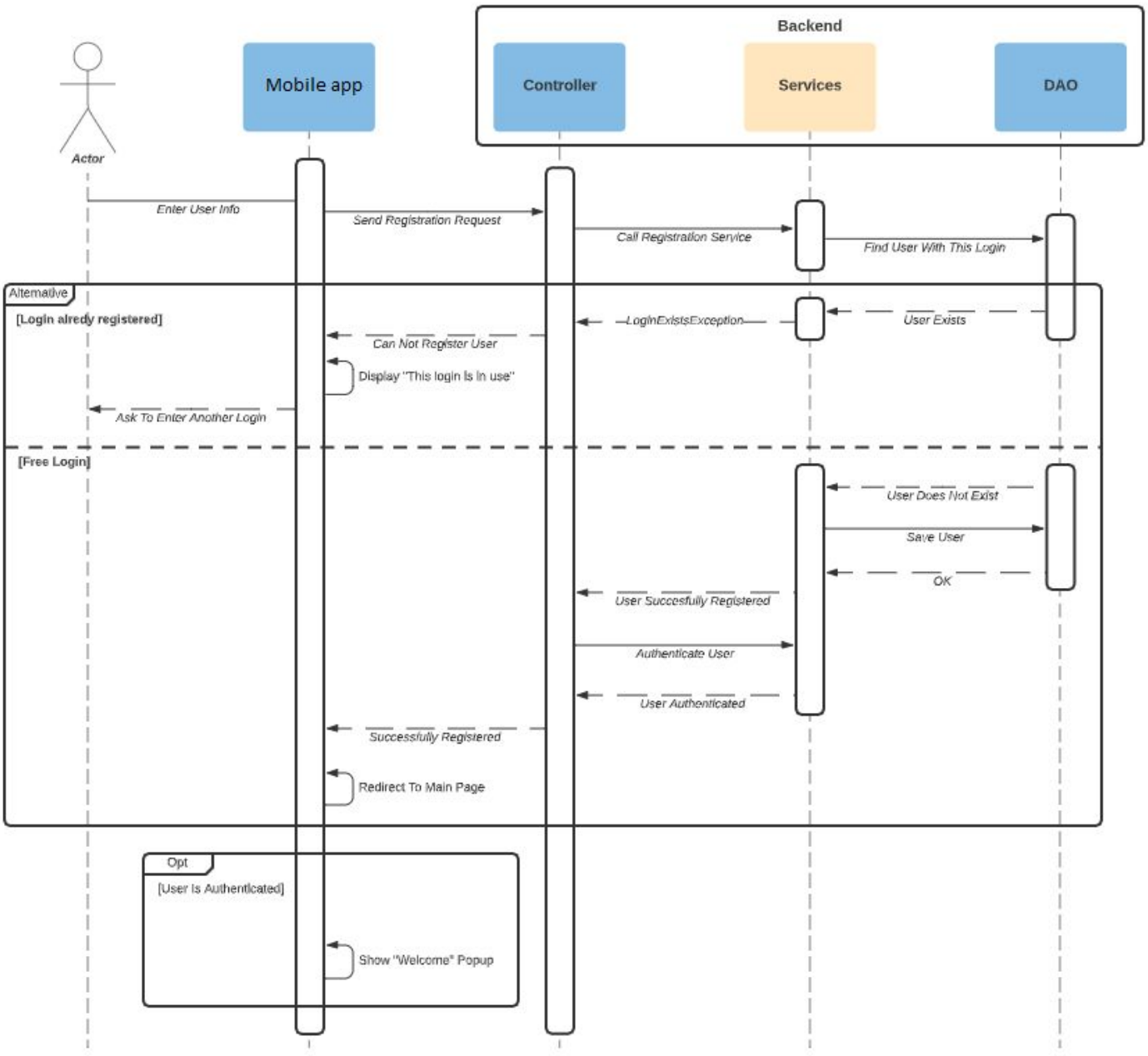




The process begins with the actor entering the credentials, then the authorization request is sent to the Controller, which sends a authentication service message to the services and the user is found with the entered credentials in the database. There is an alternative action while entering the credentials, as it is checked first whether the user exists in the database or not. If the user does not exist, then they are asked to re-enter their credentials.

Registration Diagram

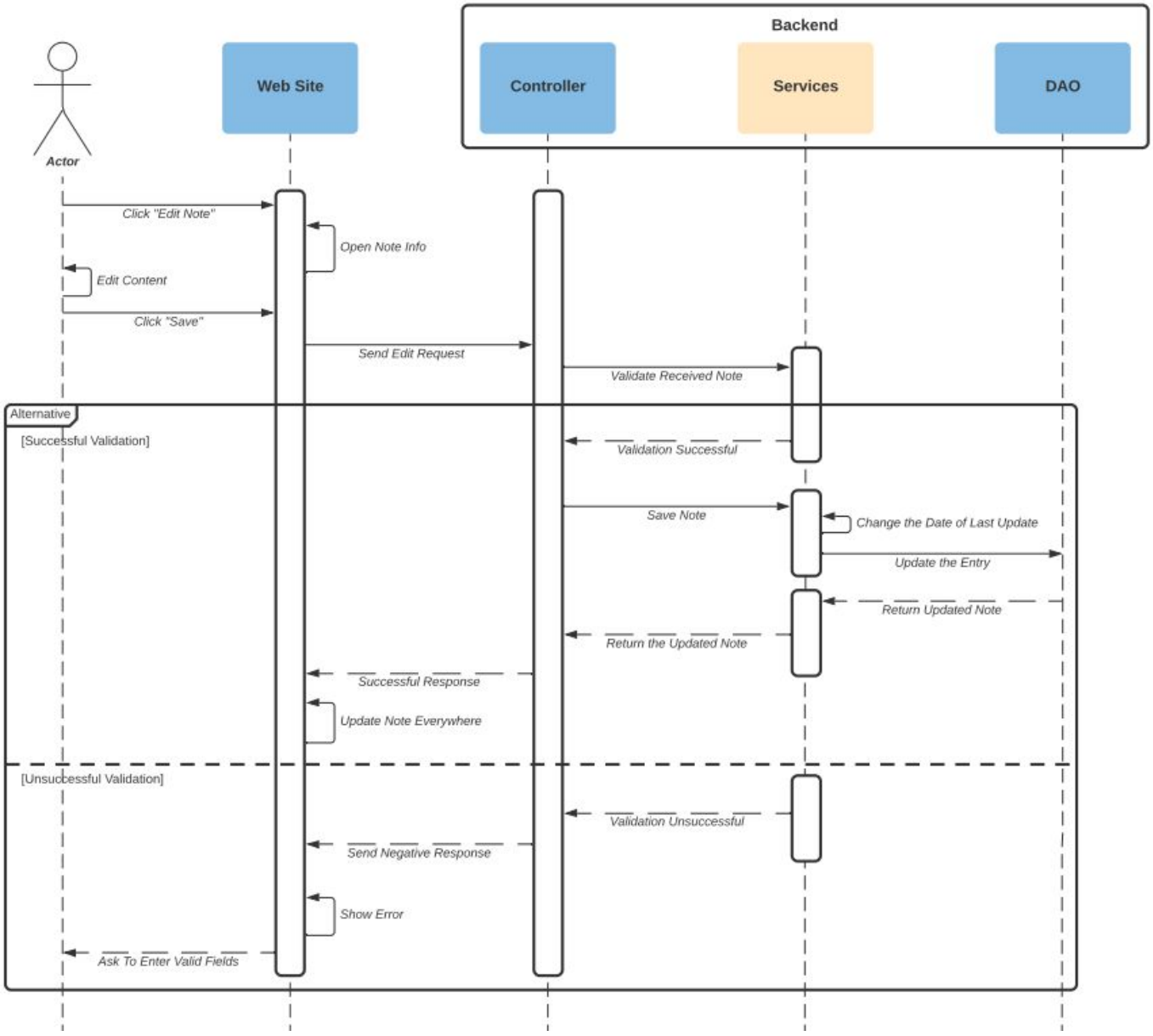


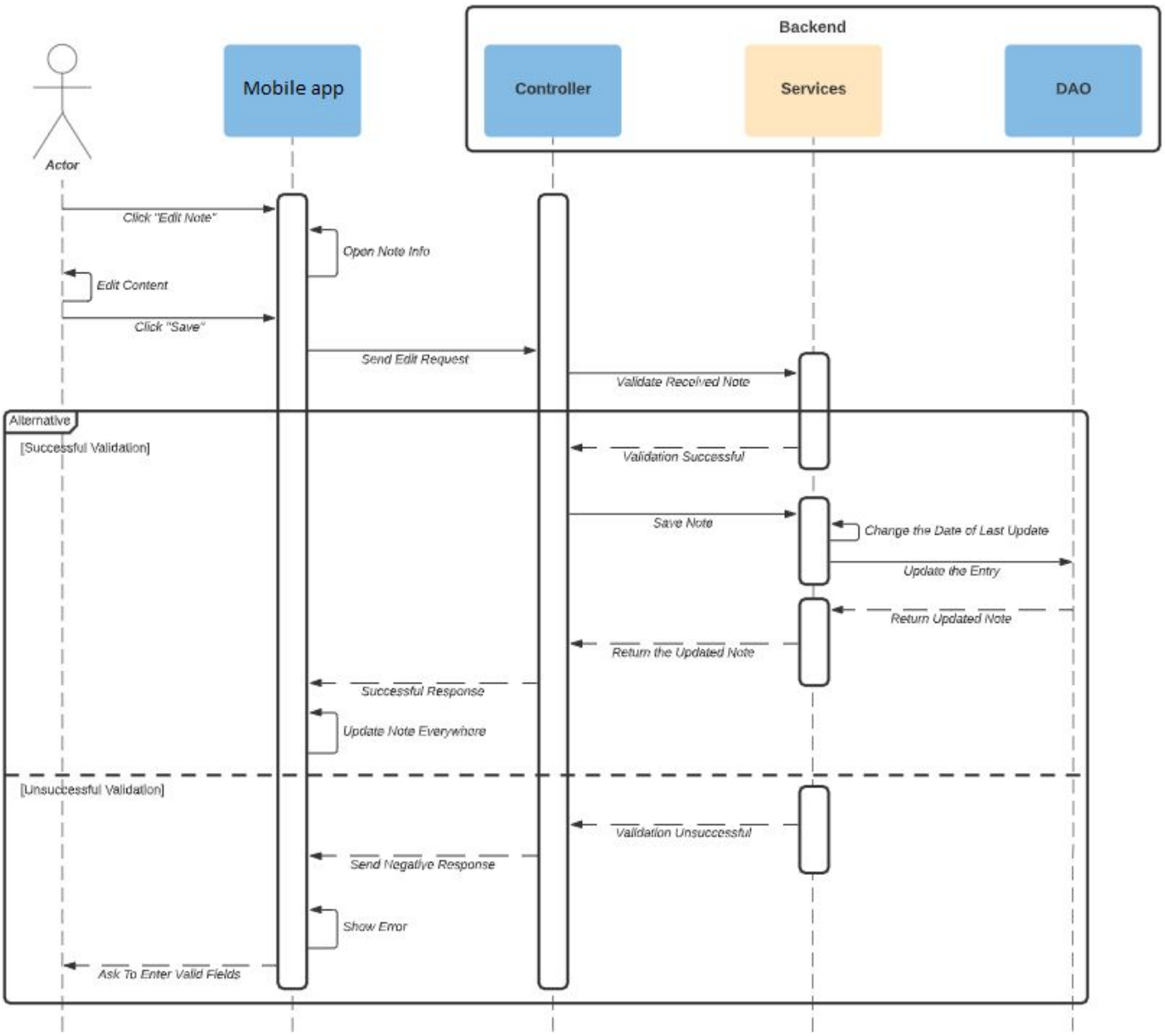


The user is asked to enter the user info necessary to make the registration, and the request is sent to the controller, which sends a message to the services, then the user with this login is found in the database. If the user already exists, the error message “this login is in use” is shown and the user is asked to enter another login.
If the user does not exist in the database, the user is successfully registered, is redirected to the main page, and the user info is saved in the database.

In the optional loop system, the user is authenticated, and is shown the “Welcome” popup message.

Edit Note Diagram

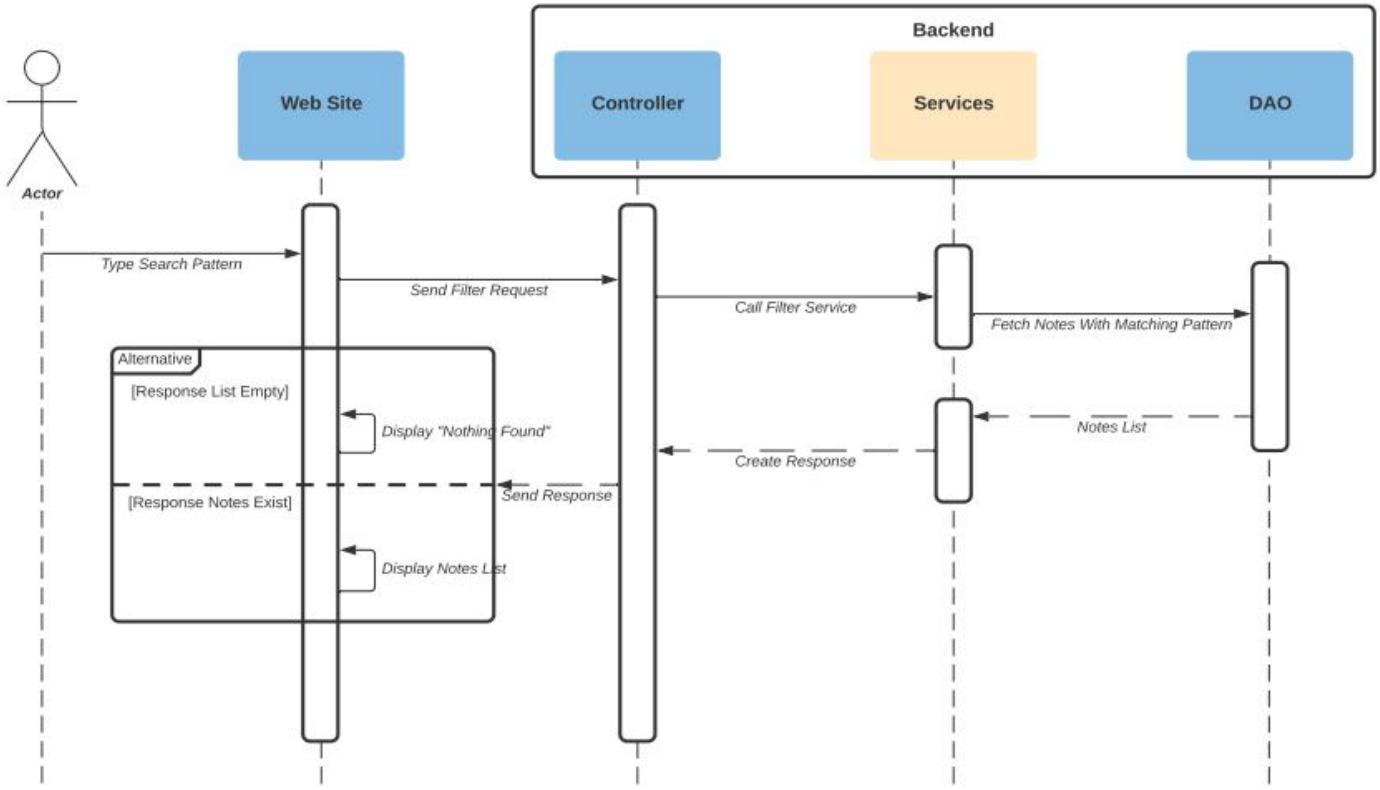


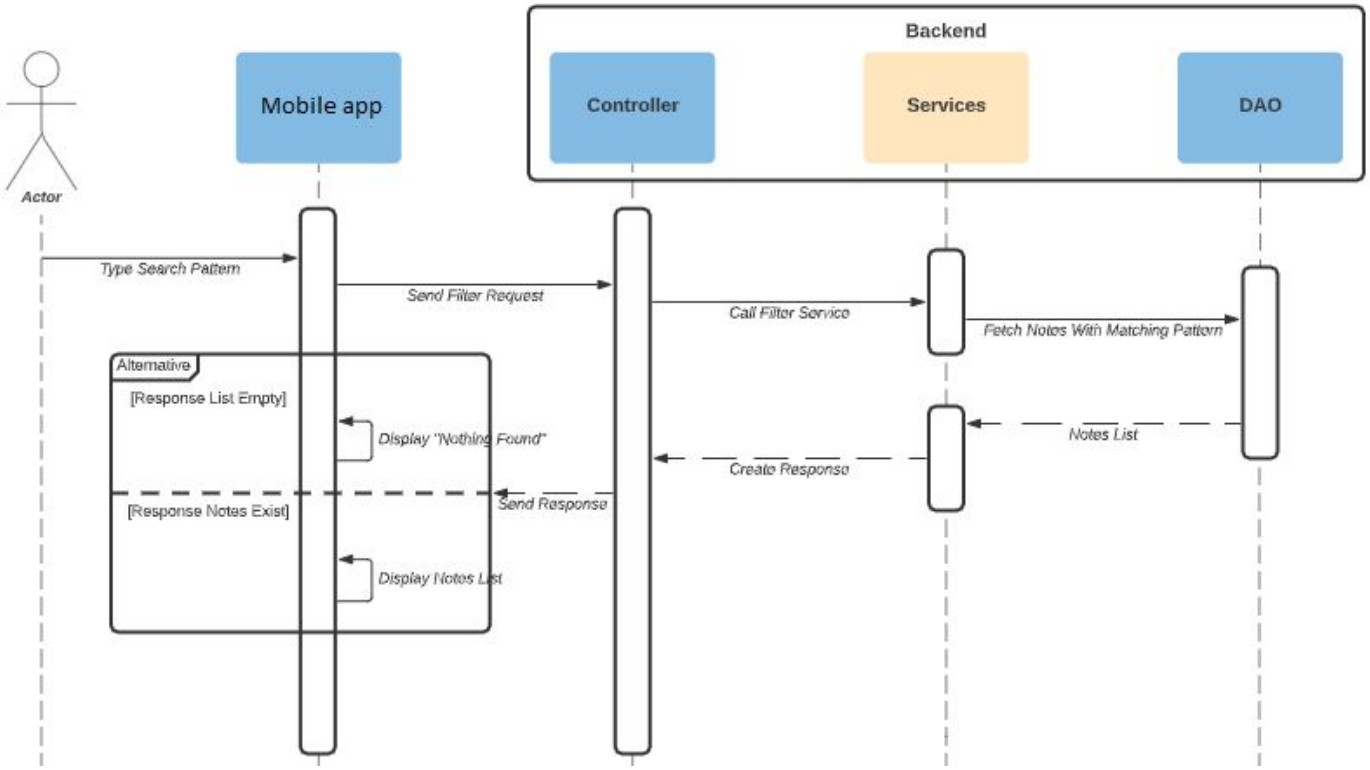


The user is asked to click “Edit note” which opens the note info and sends an edit request to the Controller, and the note is sent for validation to the backend services. If the note validation is successful, the note is saved, the entry is saved and the date of last update is updated. The updated note is sent to the controller, and the website/mobile application gets a positive response.

If the validation was unsuccessful, a negative response is sent to the website/mobile application, and an error is shown and the user is asked to enter the valid fields.

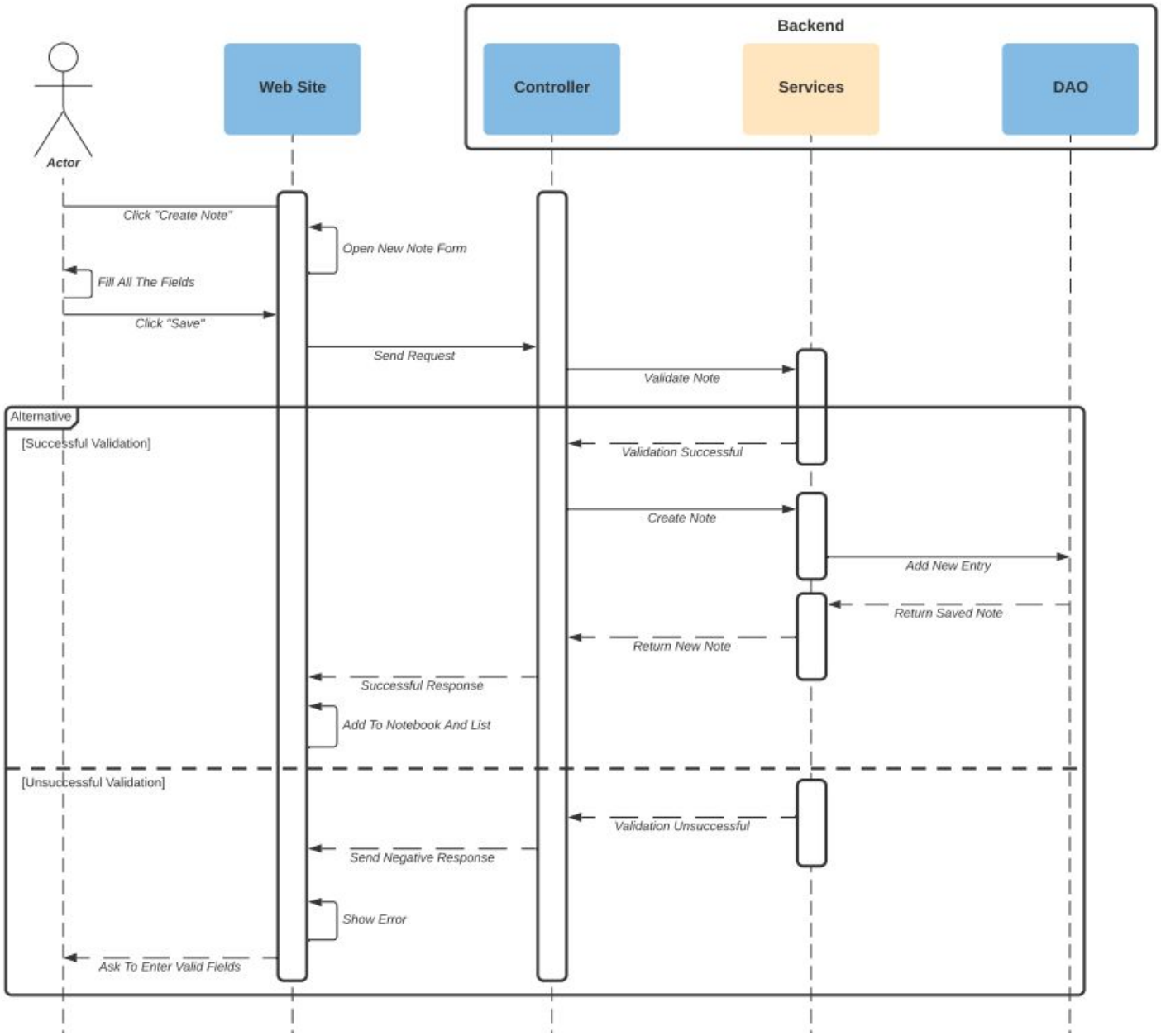
Filter Diagram

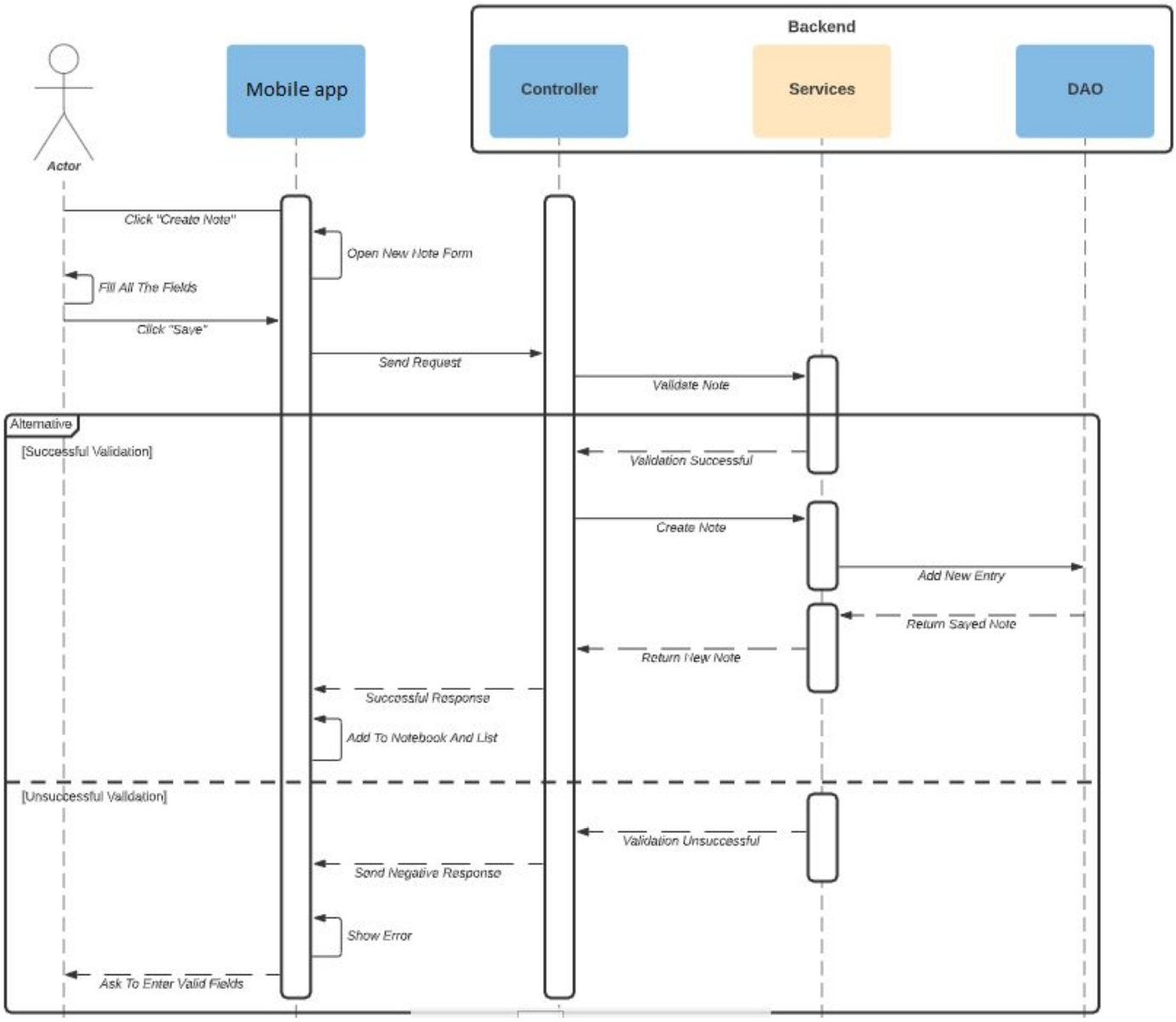




The user is asked to enter the search pattern of the required note, the filter service is called in the services section of the backend, and the notes with the matching pattern are acquired from the database. Then the response is sent to the website/mobile application, it either displays the notes list if anything is found. If nothing is found, then the error message “Nothing found” is printed.

Create Note Diagram





The user is asked to click the “Create note” button and a new note form is opened. The user then has to enter the fields to make a note. When the user selects to save the note, it sends a save request and the note is sent for validation.

If the validation was successful, the note is created and the new note is added as a new entry in the database. Then the controller sends the successful response to the website/mobile application, and the note is added in the notebook and the list of notes.

If the validation is unsuccessful the backend services send the validation unsuccessful message to the controller, and the error is displayed in the website/mobile application, and the user is asked to enter all the valid fields.

2.2.7 Functionalities

- Users can log in to the app and register.
- Users can log out.
- Users can create notes.
- Users can create notebooks.
- Users can move notes from one notebook to another.
- Users can rename and edit notes.
- Users can rename notebooks.
- Users can export and import notes.
- Users can share notes via links.
- Users can delete notes.
- Users can pin notes.
- Users can delete notebooks.
- Users can change their authentication data.

2.3 Non-Functional requirements with FURPS+

- Usability
    - Users are expected to have basic knowledge on how note taking apps work. Additionally, it is assumed that they understand the structure of the file systems (folder-subfolder-file).
    - Users use keyboard and mouse/touchpad as an input device in case of working with the computer and screen in case of working from the mobile app. The output device in both cases is the display.
    - Proper tutorial for completely new users is provided.

- Reliability
    - The app is available from any browser and on the iOS and Android mobile operating systems.
    - The system is a note-taking app for educational purposes, so it may contain bugs and minor errors that can be tolerated.
    - Data is preserved in the database on the remote server and is not lost in case of the unexpected system termination (unless the database has become inaccessible or has been dropped).
    - Notes can be edited only by the owning user.
- Performance
    - The system should hold the number of users with notes not exceeding the physical limitations of the server.
    - All notes operations should be atomic.
- Supportability
    - The developing team fixes bugs after they have been reported.
    - Updates are done by simply updating code on the server, not requiring the user to install something (except for the one-time install of the app on the mobile).
- Implementation requirements
    - Graphic user interface is required.
    - Languages of implementation are limited to those supporting building REST APIs and supporting mobile development.
    - Platform of implementation should be *nix-like (for hosting) and mobile ones, as listed above.
- Physical requirements
    - Should run on all new browsers and mobile phones running under Android.

- Operations
    - The developers’ team is responsible for delivering new versions of the product and extending features, deploying it, and fixing bugs.

3. Communication Protocol

3.1. Messages

All the messages are sent using the HTTP protocol. All requests and responses will be using JSON as a format for defining objects.

Two main entities that will be passed across the components will be **Note** and **Notebook**. Their structure can be seen here:

Note Object

```
{
    "noteId":"number",
    "pinned":"boolean",
    "notebookId":"number",
    "noteTitle":"string",
    "content":"string",
    "createDate":"date",
    "updateDate":"date"
}
```

Notebook Object

```
{
    "notebookId":"number",
    "name":"string",
    "notes":"[]"
}
```

After successful login in a user will receive an **Authenticated User Object** which will contain the necessary information for future communications as well as a generated JWT token to access the secured APIs. The following JSON represents our **Authenticated User Object** object.

```
{
    "userId":"number",
    "firstName":"string",
    "notebooks":"[]",
    "token":"string"
}
```

Meanwhile, the full **User** object which will be used on the backend and saved after registration looks as follows:

3.2 Endpoints

Endpoint - Adding New Note:

Backend should expose the endpoint to add a new note. Endpoint should be secured. Endpoint details:

Content-type: application/JSON

POST: /notes/

Request Payload: Note's details

Response Status Code: 201 CREATED, 403 FORBIDDEN

Response Payload: Note object

Endpoint - Filtering a Note:

Backend should expose the endpoint to find some given pattern in notes. Endpoint should be secured. Endpoint details:

Content-type: application/json

GET: /notes/filter/{pattern}

Request Payload: Empty

Response Status Code: 200 OK, 404 NOT FOUND, 403 FORBIDDEN

Response Payload: A list of notes

Endpoint - Editing a Note:

Backend should expose the endpoint to be able to edit a note. Endpoint should be secured. Endpoint details:

Content-type: application/json

PUT: /notes/

Request Payload: Note object

Response Status Code: 200 OK, 403 FORBIDDEN, 404 NOT FOUND

Response Payload: Updated note

Endpoint - Deleting a Note:

Backend should expose the endpoint to be able to delete a note. Endpoint should be secured. Endpoint details:

Content-type: application/json

DELETE: /notes/{noteId}

Request Payload: Empty

Response Status Code: 202 ACCEPTED, 404 NOT FOUND, 403 FORBIDDEN

Response Payload: Empty body

Endpoint - Sharing a Note:

Backend should expose the endpoint to be able to share a note. Endpoint should be secured. Endpoint details:

Content-type: application/json

GET: /notes/share/{noteId}

Request Payload: Empty

Response Status Code: 200 OK, 403 FORBIDDEN, 404 NOT FOUND

Response Payload: Generated URL to the note

Endpoint - Log in:

Backend should expose the endpoint to be able to log in to your account. Endpoint details:

Content-type: application/json

POST: /login

Request Payload: { login: “login-value”, password: “password-value” }

Response Status Code: 200 OK, 400 BAD REQUEST

Response Payload: Authenticated User Object

Endpoint - Log out:

Backend should expose the endpoint to be able to log out of your account. Endpoint should be secured. Endpoint details:

Content-type: application/json

GET: /logout

Request Payload: Empty

Response Status Code: 200 OK, 403 FORBIDDEN, 400 BAD REQUEST

Response Payload: Empty body

Endpoint - Sign up:

Backend should expose the endpoint to be able to create an account. Endpoint should be secured. Endpoint details:

Content-type: application/json

POST: /signup

Request Payload: User Object

Response Status Code: 200 OK, 400 BAD REQUEST

Response Payload: Empty body

Endpoint - Save Notebook:

Backend should expose the endpoint to be able to add or update a notebook. Endpoint should be secured. Endpoint details:

Content-type: application/json

POST: /notebooks/

Request Payload: Notebook object

Response Status Code: 200 OK, 403 FORBIDDEN, 404 NOT FOUND

Response Payload: Empty body

Endpoint - Export Note:

Backend should expose the endpoint to be able to export a note. Endpoint should be secured. Endpoint details:

Content-type: application/octet-stream

GET: /notes/export/{noteId}

Request Payload: Empty

Response Status Code: 200 OK, 400 BAD REQUEST, 404 NOT FOUND

Response Payload: Note in a form of txt.

3.3 Security

Each request except login should contain a security token in the header. If the security token is valid, then the user is authorized to use the resource. Security token is valid, when it is checked by the server and it is not yet expired. If the token is not valid then the backend returns status code 403.

3.4 Abnormal behavior

In case of any unexpected situation which may happen on either server-side or client-side, User will receive a popup error message explaining that something went wrong. It also covers the internet connection troubles, when User will not be able to perform any update or save action while the connection is not stable and is not set.

The client application will be able to show the last loaded data and work in offline mode. If for any reason during a database transaction any exception will be thrown, all the changes made so far will be rolled back and User will be informed that his action was not successfully finished. Overall our application is going to be ready to handle any make the best user experience.

4. Simulation of User's Activities

1. After visiting the website or opening the mobile application, the user will be asked to log in or to create an account.
2. By clicking on the "Register" button a new form will appear asking to fill all the fields and to create an account in our system which will then serve for user authentication.
3. Once logged in, the user will be redirected to the main page where all the functionality is located. user will see his recently saved notes taking most of the screen and a list of notebooks which can be clicked and selected.
4. Once the user clicks on any notebook, its stored notes will be displayed and replace current ones.
5. By double-clicking on the notebook name, the editor field will appear and its name can be changed.
6. There will be a search input at the top of the page where some string can be typed and if any note contains that string either in the header or the body, it will be filtered and displayed.
7. By clicking on the edit button which is present on every note, a user has the possibility to update the content of that note, delete it, export to .txt, pin or share with others.
8. In case he/she chooses to pin it, the note will get a special status "Pinned" and will appear at the top of any selections.

9. In case he/she chooses to share it, a link will be generated and displayed, and anybody can access the content of that note by following that link.
10. On the right side of the search input, there are two buttons “Add Note” and “Add Notebook”
11. By clicking on “Add Note”, a new modal will be shown where the user will have to fill in the new note data and click “Save” to save it.
12. By clicking on “Add Notebook”, a new modal will be shown to give a name for the new notebook, and after that, it will appear empty on the list.
13. At the top right side, there will be a “Logout” button which will remove the user’s authorization and he/she will be redirected to the Login page.

5. Conclusion

We want to design a note storing system—which allows saving, editing, and grouping in notebooks—for browsers along with a mobile application. The whole project is made out of three components: Web Application, Mobile Application, Backend Server. We will require users to create an account since in that way we can identify him/her every time when he/she logs in and we can store and manipulate data without losing it. We choose to use a client-server system as we believe that it will bring the most benefits to the project. We will use only HTTP for interactions between clients and the server. Moreover, we described all special system states to give a better idea of this useful and elegantly architectured project. We believe that this project will be of great use for everybody since we want to combine a simple and intuitive user interface with a lot of useful functionalities.