**Requirements:**

**Overview:**

The scope of the system is to provide a document management system. The users will be authenticated based on various schemas and then based on the user permissions they will either be able to view, edit or delete the documents. The user will also be able to add other users to view, edit or delete the document.

**Product Functions:**

The following are the major functional requirements of the web application.

1. Authentication / User Management:

The user will be provided an interface to login into the tool. If the user is not registered, he will be given an option to register. The user will be provided with multiple options to login to the tool.

1. Document Viewing / Creation / Deletion:

The user will have an ability to view the documents created by them, the ability to edit them and the ability to delete them.

1. Document Sharing:

The user will have the ability to share the document with other known users and add them to the shared with entry. The document owner will decide the level of access of the shared document with other users.

The user will be able to view, edit and/or delete the document based on the permission of the document shared by the owner of the document.

1. Document Locking:

The user can check out a document and lock it if he decides to edit it. The other users will see the edit options disabled with the name of the other user who has locked the document.

1. Parallel Uses:

The application should be able to support multiple users accessing the tool through multiple clients.

**Operating Environment:**

1. Hardware Requirements:
   1. Client:
      1. Any system with an ability to download a browser
      2. System connected to a network that will be able to connect to the server.
      3. Keyboard and Mouse connected
   2. Server:
      1. Webserver to host client application, NGINX, Apache HTTP server, JBoss, Apache Tomcat, Node Express Lite
      2. Application server with minimum 16GB RAM, Linux Mint or Cent or Ubuntu
      3. Access to distributed network with name node and data nodes.
   3. Database:
      1. Min 2 nodes in the cluster
      2. Support parallel reads and writes.
2. Software Requirements:
   1. Client:
      1. Chrome version 76.0.0 and up, Firefox version 68.0.0 and up
      2. GUI supported operating system
   2. Server:
      1. Angular 8.0.1
      2. rxJS 6.4.0
      3. typescript 3.4.3
      4. Node JS 10.16.0
      5. YARN 1.6.0
      6. Cookie-parser 1.4.4
      7. Passport JS 0.4.0
      8. Express JS 4.16.1
      9. Mongoose JS 5.6.9
      10. Cors 2.8.5
      11. Jade 1.11.0
      12. Nodemon 1.19.1
   3. Database:
      1. MongoDB 4.0

**System Features:**

1. Home Page:
   1. Authenticated users should go the dashboard.
   2. Authenticated users will see their username in the header.
   3. Unauthenticated users should be redirected to the login page.
2. Existing User Authentication:
   1. The user should be able to login using a username and password
   2. The user should not be allowed to proceed if the password is incorrect
   3. The user should be able to login using Facebook, GitHub and the username should be defaulted to the Username of the used schema.
3. New User Authentication:
   1. The user should be shown a link to register to use the application.
   2. Duplicate usernames should not be allowed registration.
   3. Error should be shown for duplicate user registration.
4. Logout:
   1. Authenticated users will be displayed a logout button.
   2. The logout will kill the client and server session.
5. Dashboard:
   1. The user will see all the documents he is permissible to view.
   2. If the user can create a new document, he will see a create new button.
   3. If the user can edit a document edit button will be displayed.
   4. If the user can delete a document delete button will be visible.
   5. The owner of a document will be visible next to each document.
   6. If the document is locked the edit button will be disabled.
   7. A locked document will have the name of the user who has checked out the document.
6. Document Creation:
   1. When the user clicks add he will be redirected to the add page.
   2. The user will add the title through an input text box.
   3. The user will add the body through a text area.
   4. The current user will be defaulted as the owner.
   5. The user will click save button to store the document.
   6. On save the user will be redirected to the dashboard.
   7. The new document will be present in the list on the dashboard.
7. Document Edit Mode:
   1. When the user clicks edit, he will be redirected to the edit page.
   2. The user can change the title and body of the document.
   3. The user can add new users to the shared with list.
   4. The user can check the read-only box to make the document read-only for the specific user.
   5. The user will click the update button to save.
   6. On save the user will be redirected to the dashboard.
   7. The changes to the document will be reflected on the dashboard.
8. Document Deletion:
   1. When the user clicks delete the document will be deleted and the list is updated in the dashboard.

**Non-Functional Requirements:**

The system will be a hosted on a server with high speed internet. The software assumed the server will be UNIX based and will have its IP whitelisted from the cloud database. The system is based on NodeJS which support parallel connections.

The system will encrypt passwords before sending over the network.

By using distributed system, the system will maintain high fault tolerance.

The speed of the application will depend on physical characteristics like network speed and browsers speed.

The maximum number of parallel connections to the server and the database can be configured.

**Use Cases:**

1. User Logs in:
   1. The user opens the URL the client solution is hosted on
   2. The user enters the username in a text input
   3. The user enters password in a text input
   4. The user clicks login.
   5. If login is successful, the user is redirected to dashboard
   6. If the login fails appropriate error is surfaced.
2. User Registers:
   1. User clicks register on the login page
   2. User enters username
   3. User enters password
   4. User clicks on continue
   5. If the user is created, he will be redirected to dashboard
   6. If the user creation fails appropriate errors will be displayed
3. User Adds new Documents:
   1. User clicks create new
   2. User enters title
   3. User enters body
   4. User clicks save
   5. If the save is successful, the user is redirected to dashboard and the new document is visible in the dashboard
   6. If the save fails appropriate error is displayed.
4. User Edits Documents:
   1. User clicks create edit on an existing document
   2. User enters title
   3. User enters body
   4. User clicks save
   5. If the save is successful, the user is redirected to dashboard and the new document is visible in the dashboard
   6. If the save fails appropriate error is displayed.
5. User Deletes Documents:
   1. User clicks delete on an existing document.
   2. If the delete is successful, the dashboard reloads with updated list of documents.
6. User Shares Document:
   1. User clicks on edit on an existing document
   2. User adds users using an input box
   3. User can check the read-only box next to the added user.
   4. User clicks update
   5. If the save is successful, the user is redirected back to the dashboard.
   6. If the save fails appropriate error is displayed.
7. User Logs Out:
   1. User clicks on logout
   2. If the logout is successful, the user is redirected to login with the client and server session destroyed.
   3. If the logout fails appropriate error is displayed.

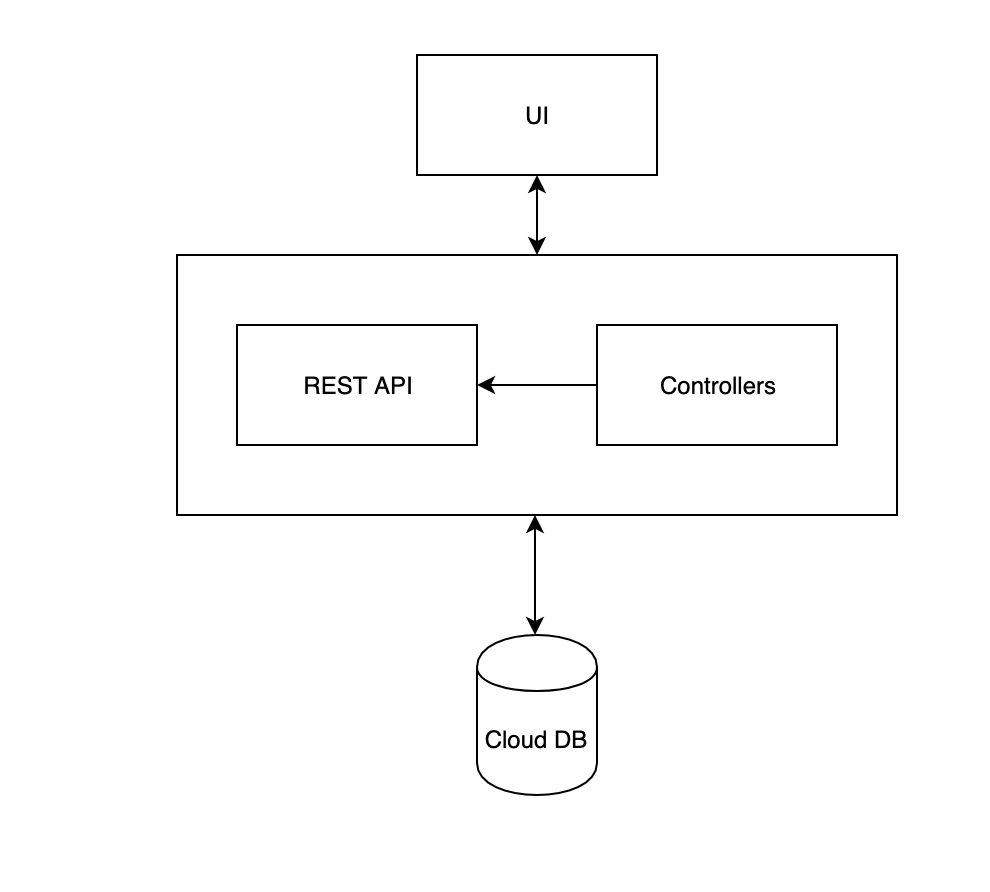
**Design:**

**Overview:**

The scope of the system is to provide a document management system. The users will be authenticated based on various schemas and then based on the user permissions they will either be able to view, edit or delete the documents. The user will also be able to add other users to view, edit or delete the document.

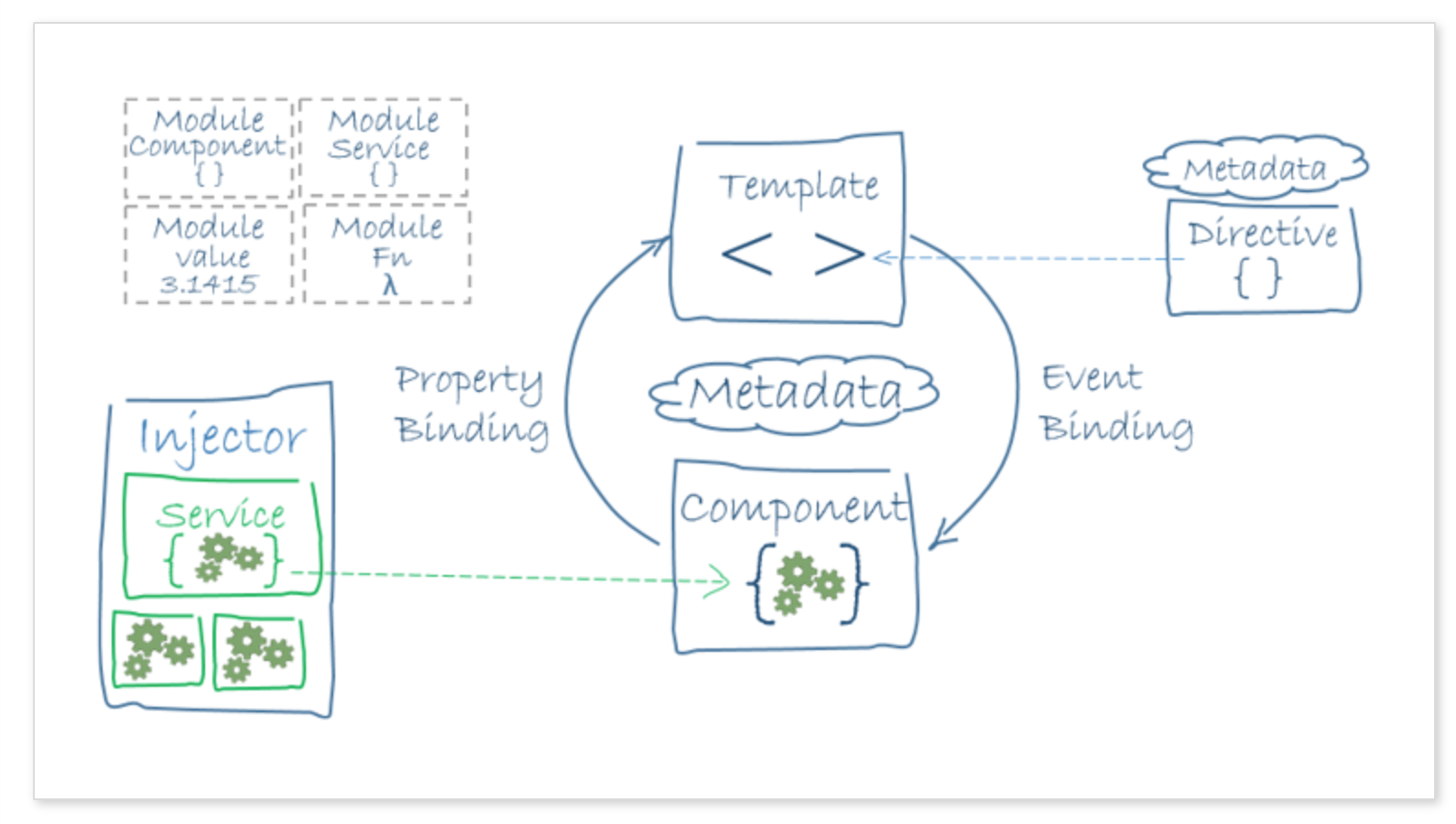
**Architectural Design:**

The base architecture of the system is a client – server model. The server provides a layer of REST apis to interact with the database.

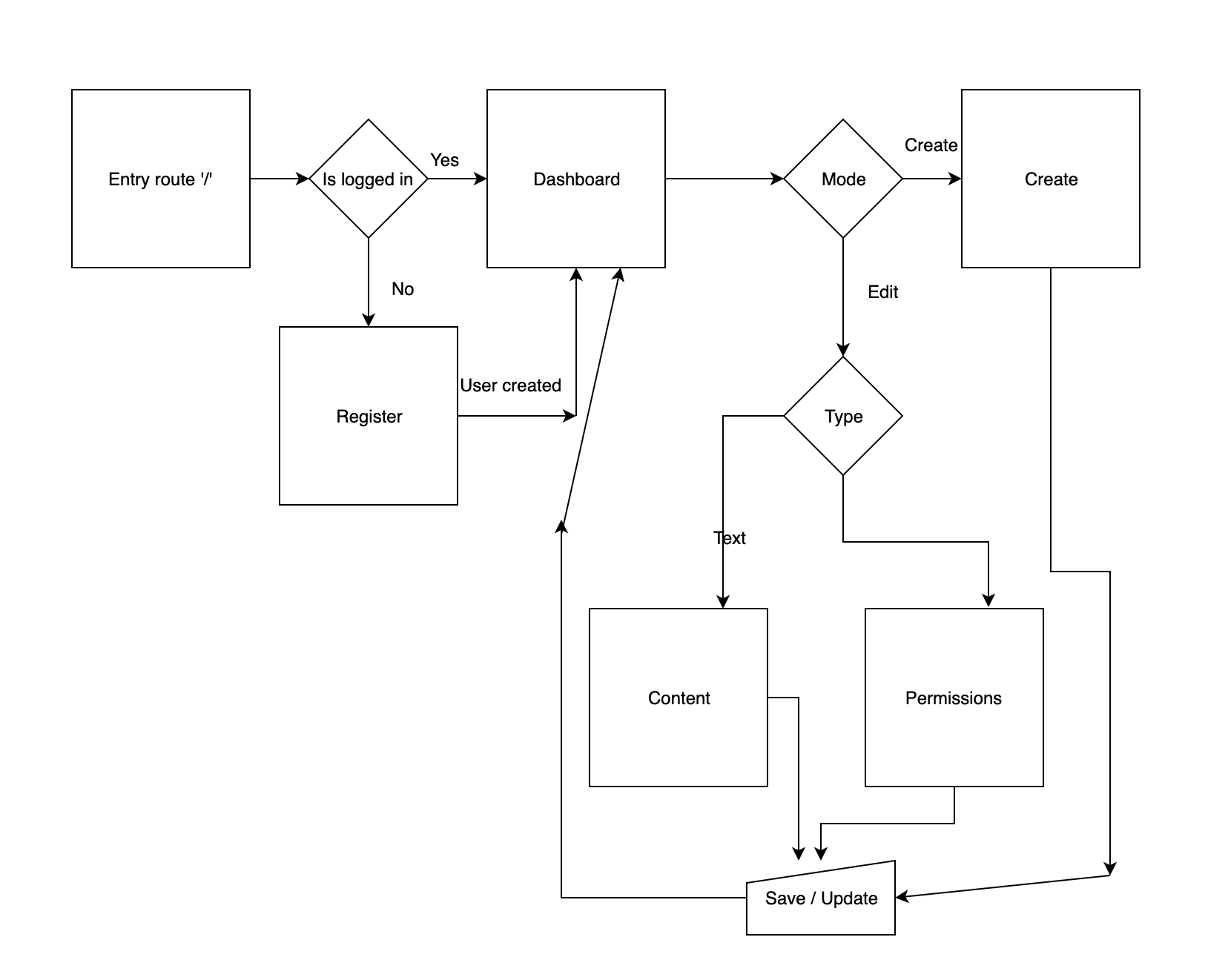


The client will be built in Angular 8.0.1. It is a single page application which follows a MVC architecture. Each component individually follows a MVC pattern, the data and routing is provided by services which follow singleton pattern, the entire UI solution is chunked into modules.

Reference: <https://angular.io/guide/architecture>



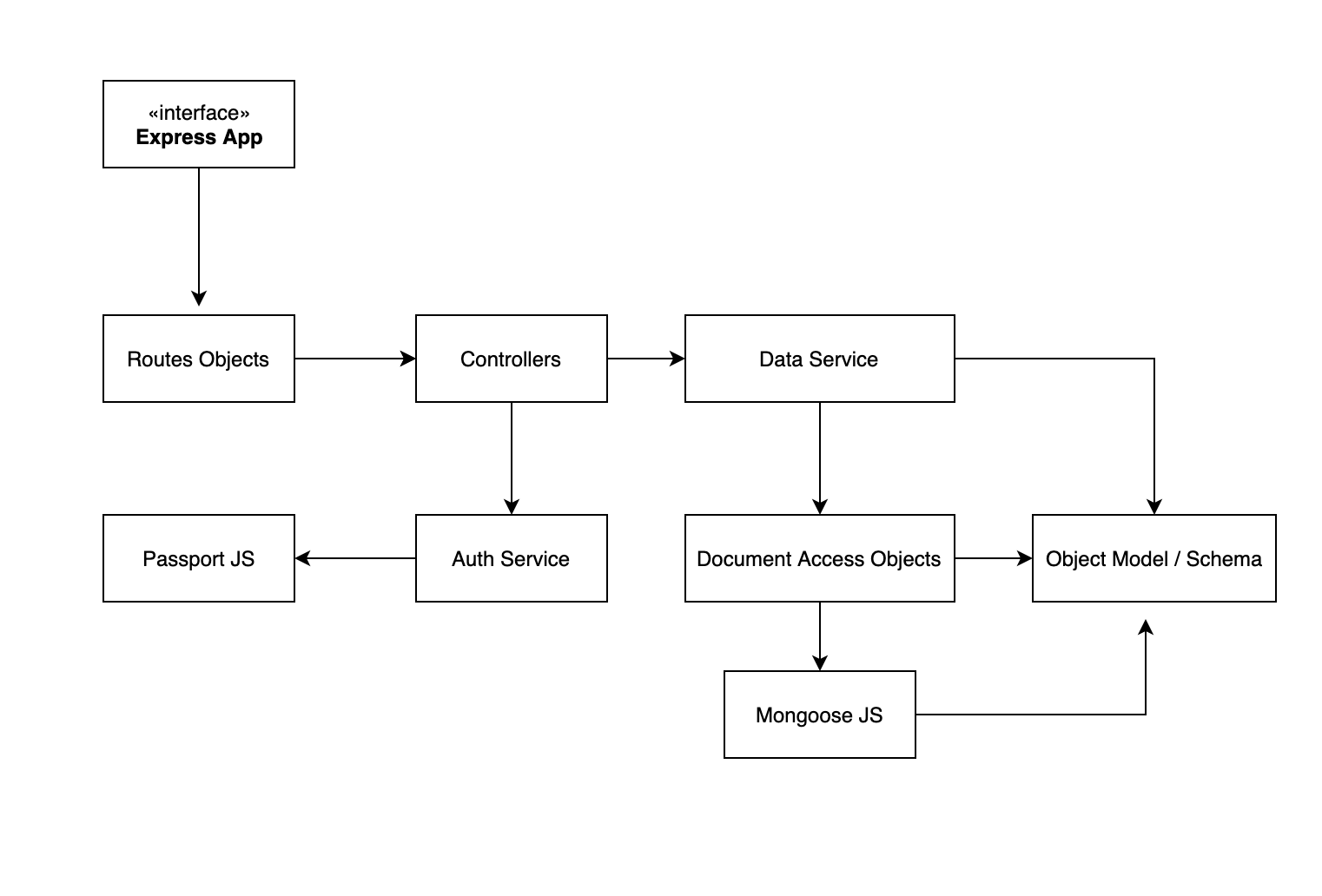
UI states:



Server Architecture:

Express JS provides the basic server framework. The routes are registered in the main file. Each route is attached to its controller in the routes object which is imported in the main app. The controllers call the required data from the user or document service.

The service then calls the document access objects which have methods to do CRUD operations on the objects. The models or schemas are defined separately are injected into other objects.



**Non-Functional Requirements:**

1. Usability:
   1. Source of Stimulus:

The user of the application will be the source of stimulus

* 1. Stimulus:

The use cases will be used as stimulus for UAT

* 1. Response:

The response will be the result of the use case.

* 1. Response Measure:

The response measure will be if the actual result matches the expected result.

* 1. Environment:

There will one environment for this, that is after the integration is done.

* 1. Artifact:

The artifact will be the entire solution.

Tactic Used:

The server and client are separate systems. The client is a single page application and UX will be seamless.

1. Testability:
   1. Source of Stimulus:

The tester will be the source of stimulus

* 1. Stimulus:

The test cases will be the stimulus.

* 1. Response:

The response of the test case will be stimulus.

* 1. Response Measure:

The response measure will be the actual result of the test case.

* 1. Environment:

There will be multiple environments for this. First will be the component testing, then integration testing and then system testing.

* 1. Artifact:

For the first environment all the systems can be separately tested. For integration testing and up the artifact is the entire system

Tactic Used:

With correct whitelisting and preconditions each system can be individually tested. Abstracting Data sources and limiting complexity makes this more testable.

1. Extensibility:
   1. Source of Stimulus:

The source will be the developer adding new requirements.

* 1. Stimulus:

Add / Delete / Modify a requirement.

* 1. Response:

Make modification, test modification and deploy the modification.

* 1. Response Measure:

The measure will be the time taken and the level of effort to make the modification.

* 1. Environment:

The environment is development time.

* 1. Artifact:

The artifact will be the entire system.

Tactics Used:

The UI and API are sectioned into modules. The changes will be localized for most changes. The document access layer is different than the services than the functions driving the api handler.

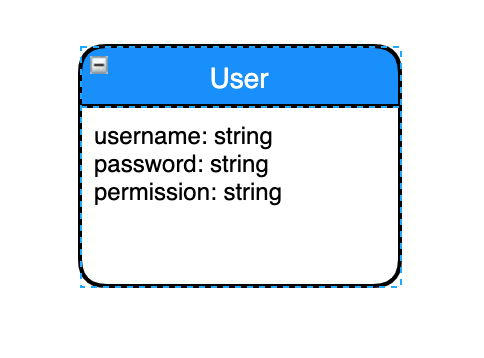
In case of the UI each section that can be isolated is a different component and the common things are driven through services.

**Detailed Design:**

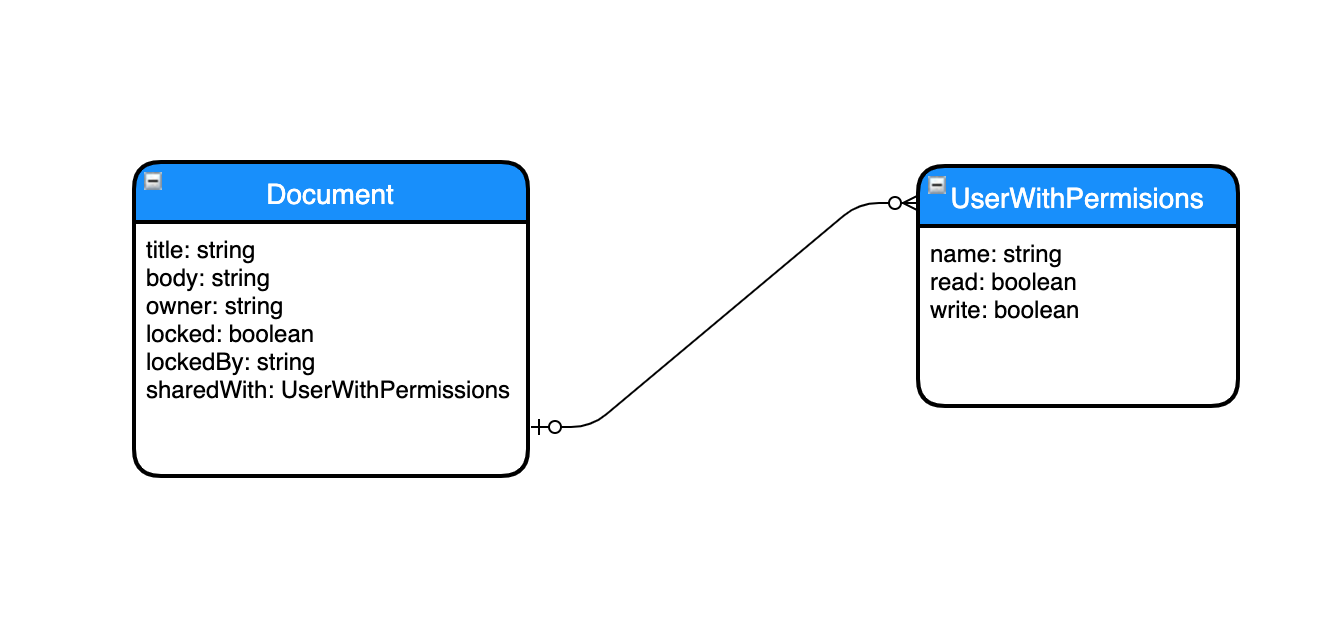
Data design:

MongoDb is used to store the user data and document data. Since the data is in the form of a NoSQL database it is non tabular and in object format

User:



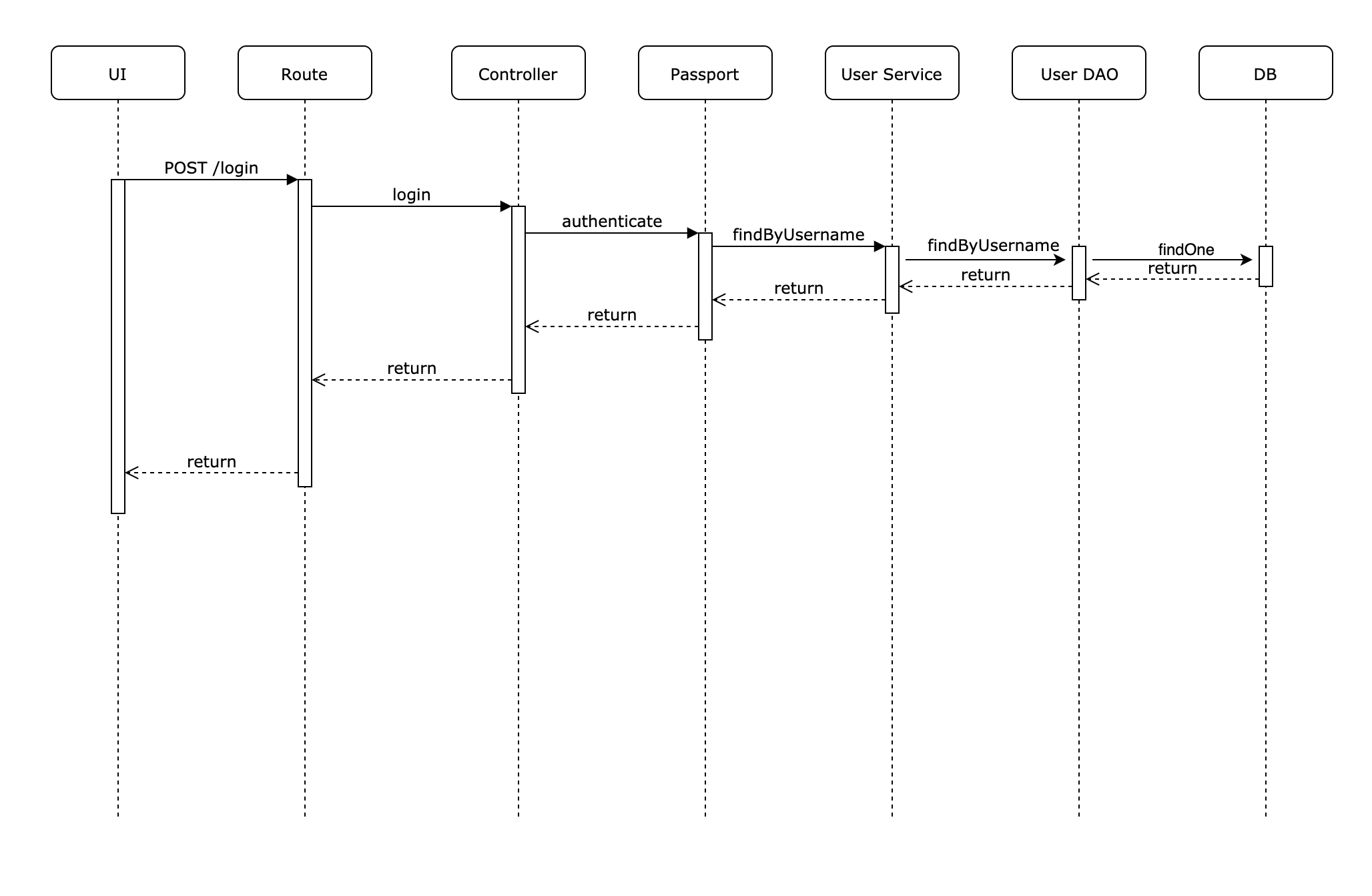
Document:



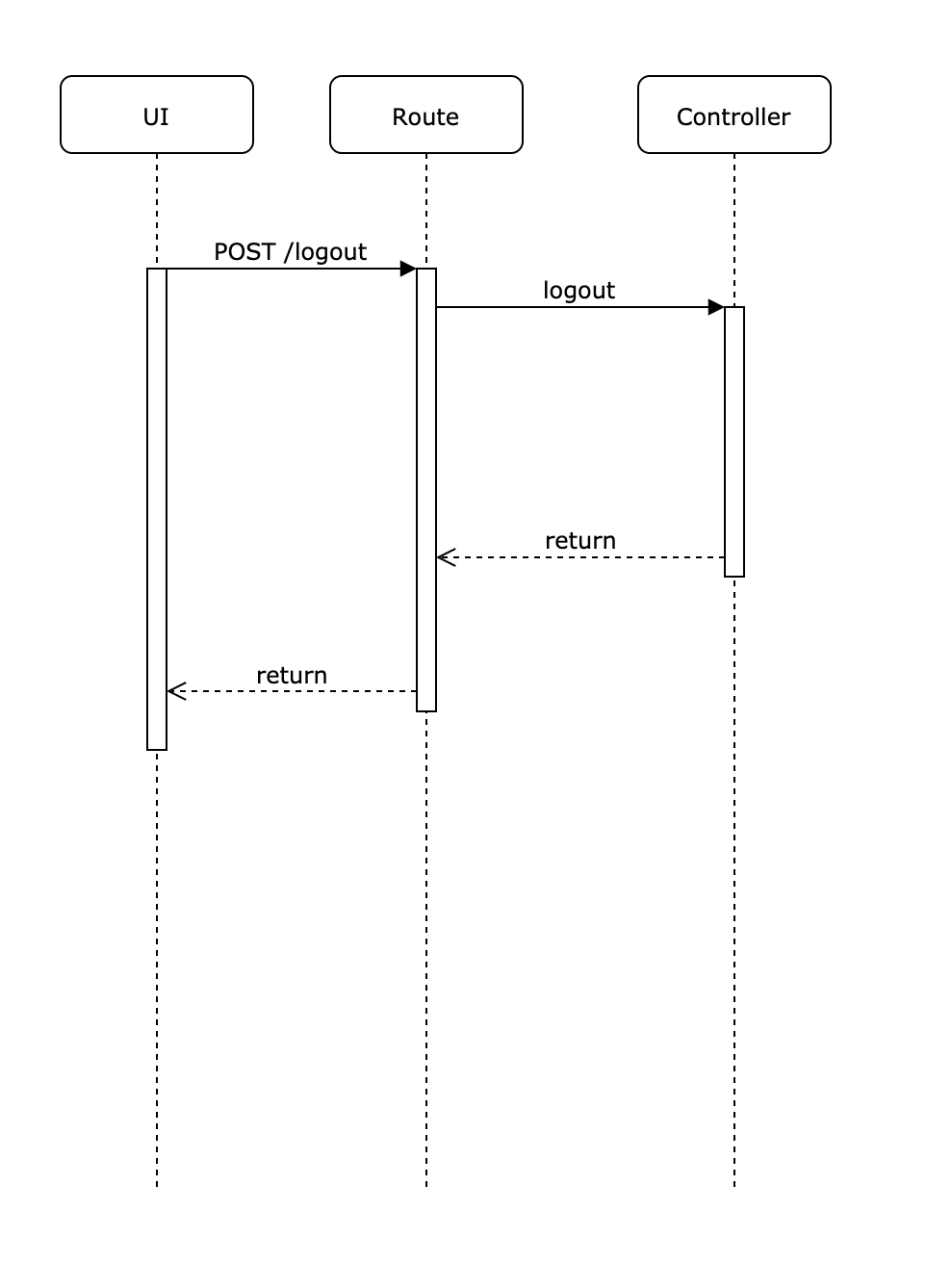
System Flows:

Below are the sequence diagrams for the flows:

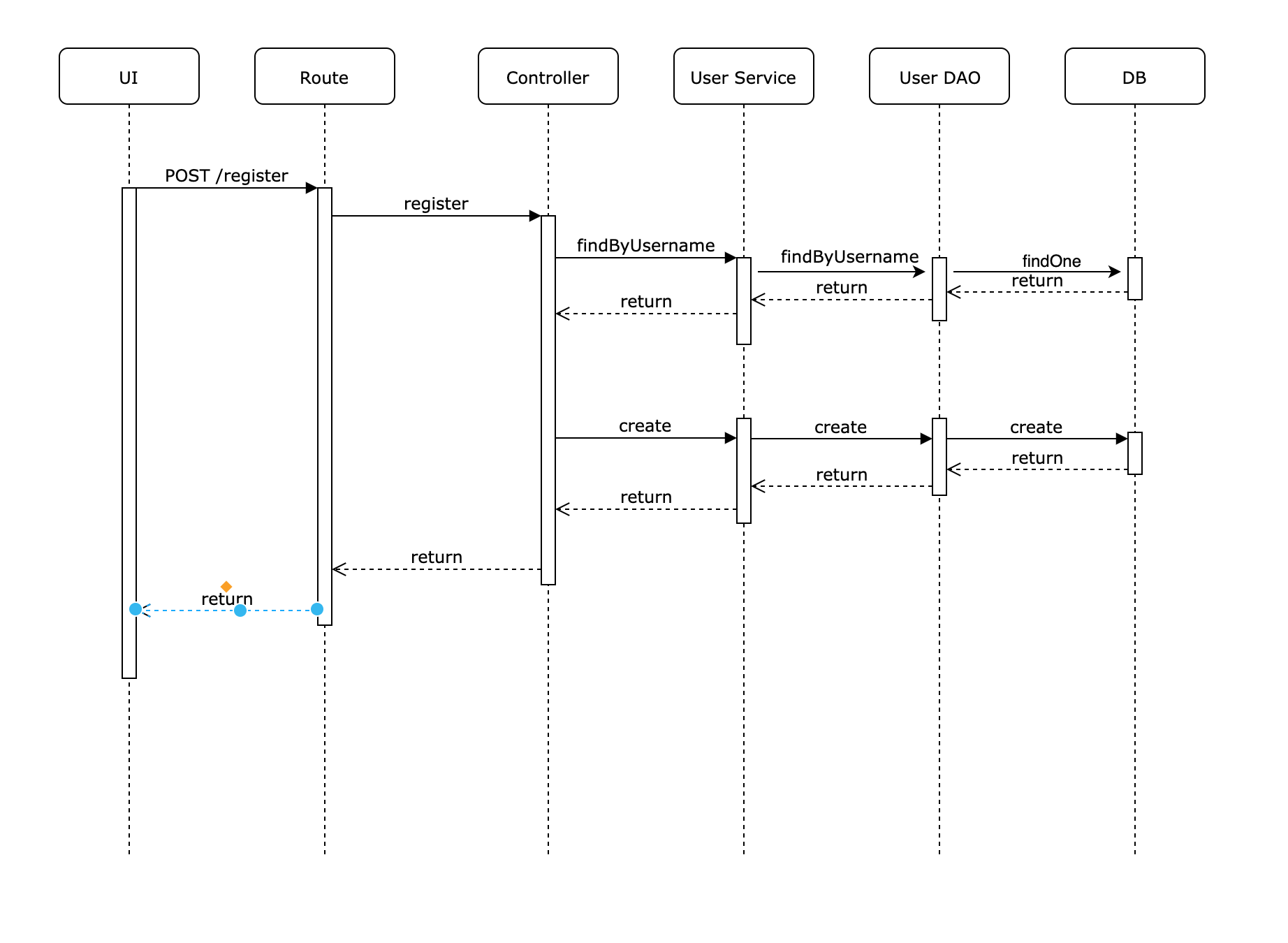
Login:



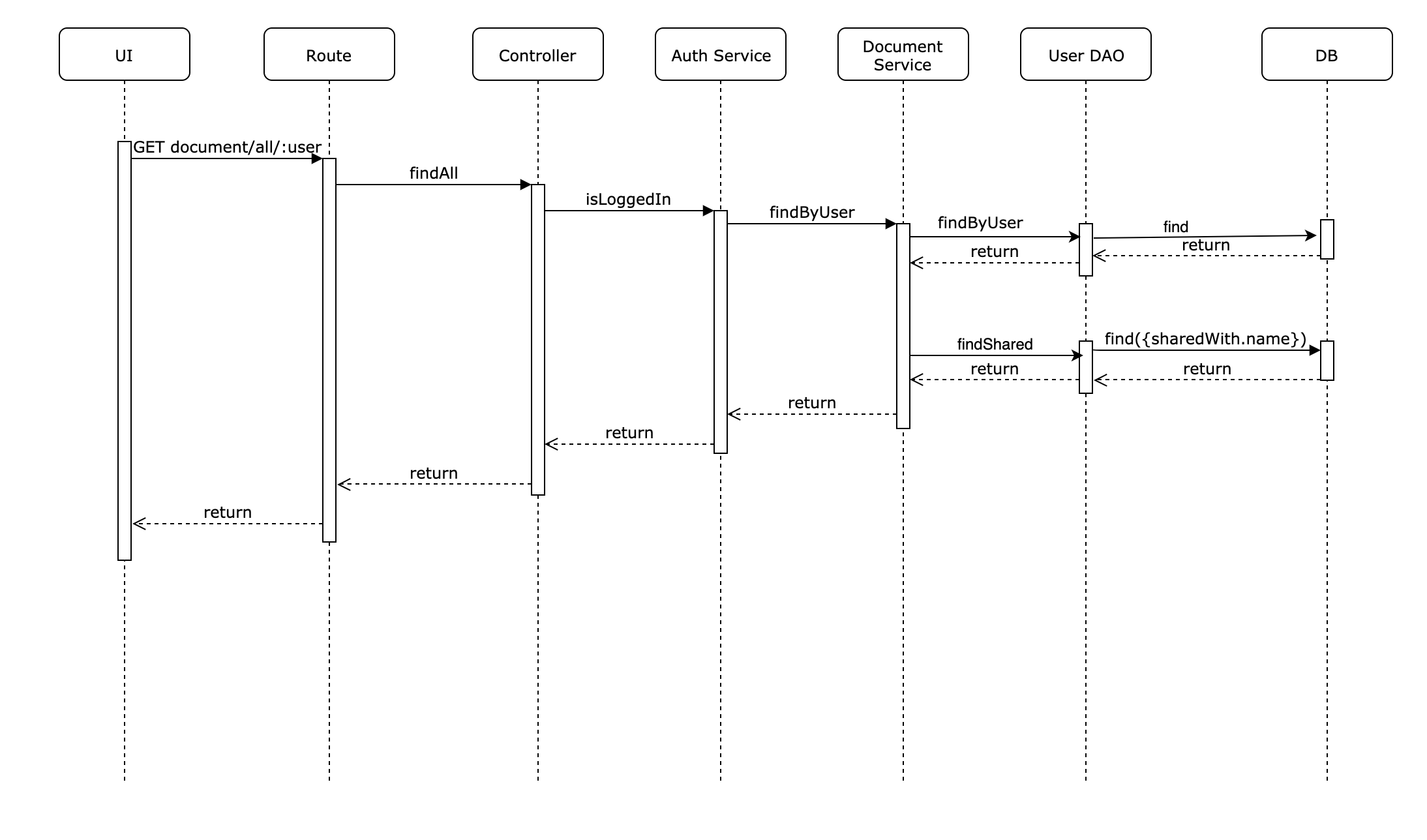
Logout:

****

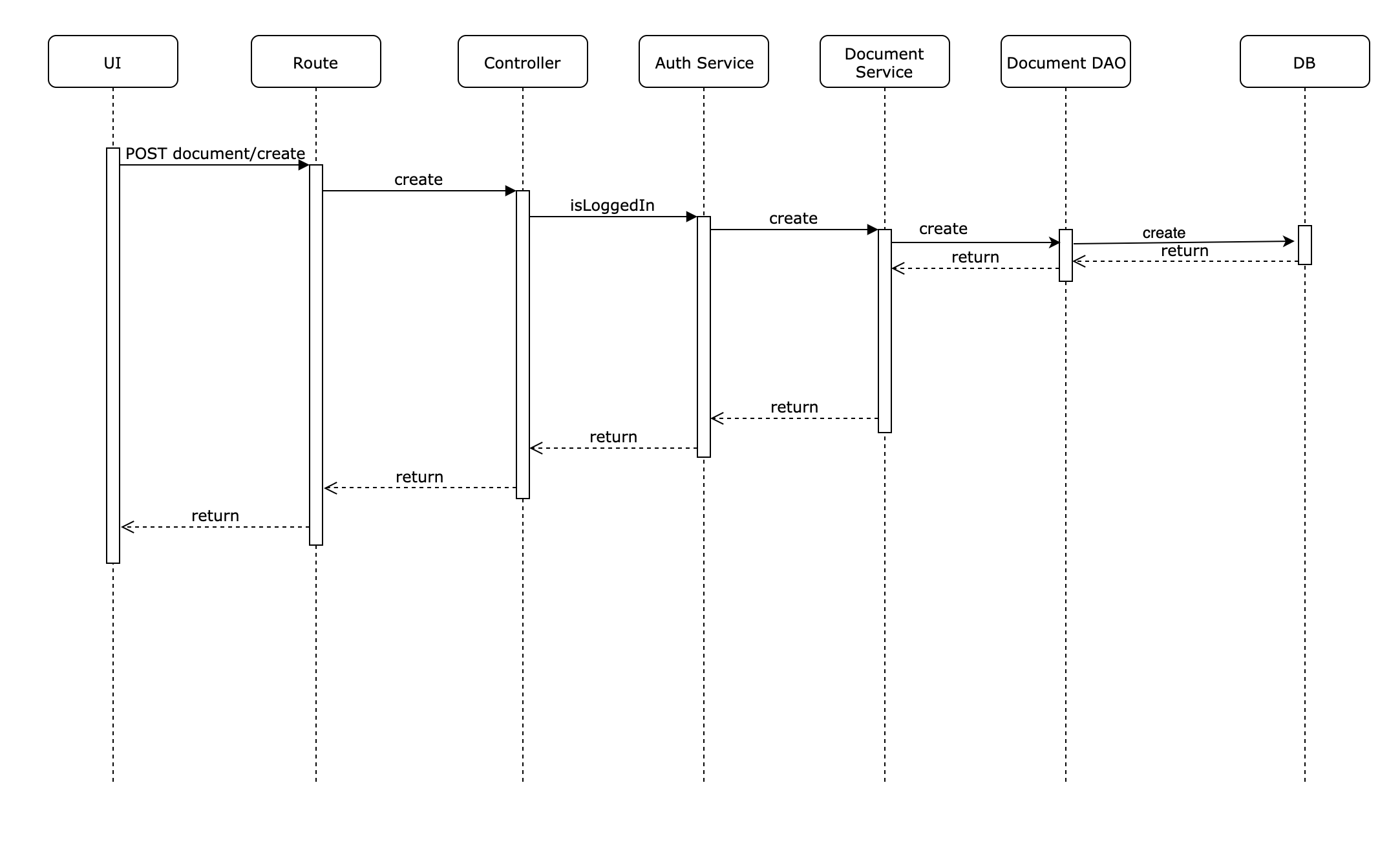
Register:

****

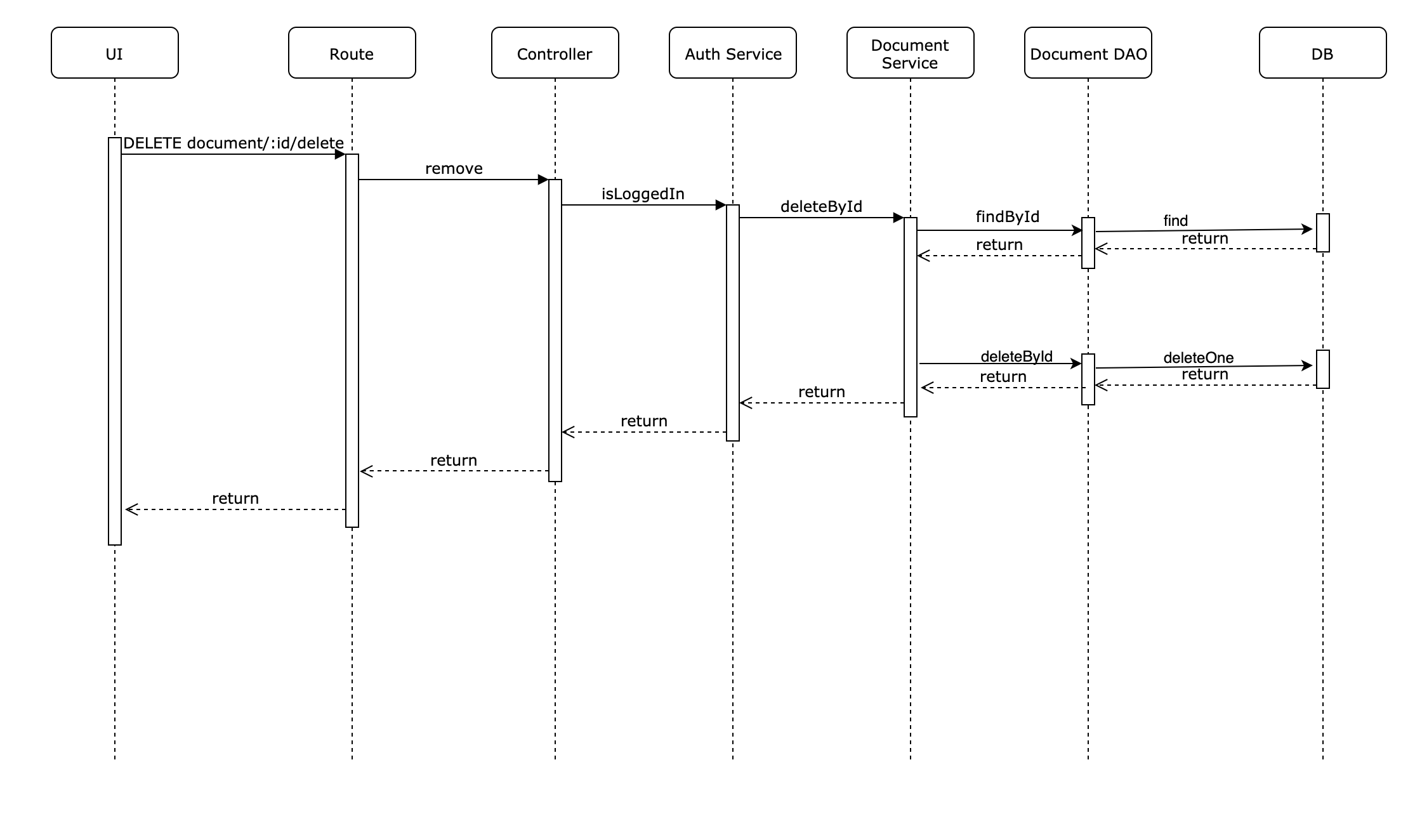
Viewing Owners Document:



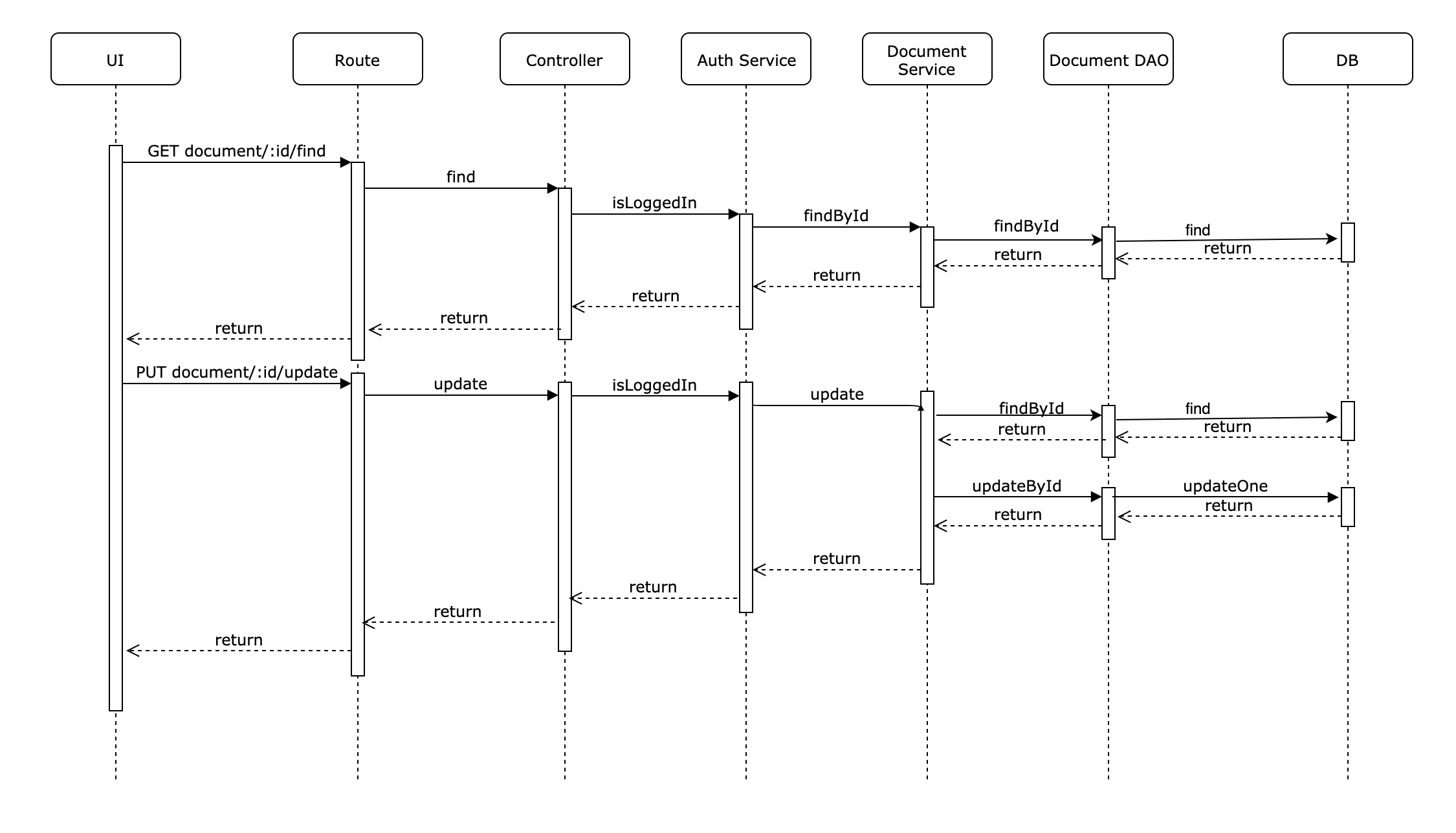
Adding a Document:

****

Removing a Document:



Updating a Document:

****

**Requirements Traceability Matrix:**

|  |  |  |
| --- | --- | --- |
| **Requirement Number** | **Requirement Description** | **Section Covering it** |
| **1.a** | Authenticated users should go the dashboard | Sequence diagram for Login |
| **1.b** | Authenticated users will see their username in the header. | Sequence diagram for Login |
| **1.c** | Unauthenticated users should be redirected to the login page. | UI States |
| **2.a** | The user should be able to login using a username and password | Sequence diagram for Login |
| **2.b** | The user should not be allowed to proceed if the password is incorrect | Sequence diagram for Login |
| **2.c** | The user should be able to login using Facebook, GitHub and the username should be defaulted to the Username of the used schema | Sequence diagram for Login |
| **3.a** | The user should be shown a link to register to use the application | UI States |
| **3.b** | Duplicate usernames should not be allowed registration. | Sequence diagram for Register |
| **3.c** | Error should be shown for duplicate user registration. | Sequence diagram for Register |
| **4.a** | Authenticated users will be displayed a logout button. | UI States |
| **4.b** | The logout will kill the client and server session. | Sequence diagram for Logout |
| **5.a** | The user will see all the documents he is permissible to view. | Sequence diagram for Viewing a Document |
| **5.b** | If the user can create a new document, he will see a create new button. | Data Design |
| **5.c** | If the user can edit a document edit button will be displayed. | Data Design |
| **5.d** | If the user can delete a document delete button will be visible. | Data Design |
| **5.e** | The owner of a document will be visible next to each document. | Data Design |
| **5.f** | If the document is locked the edit button will be disabled. | Data Design |
| **5.g** | A locked document will have the name of the user who has checked out the document. | Data Design |
| **6.a** | When the user clicks add he will be redirected to the add page. | UI States |
| **6.b** | The user will add the title through an input text box. | UI States |
| **6.c** | The user will add the body through a text area. | UI States |
| **6.d** | The current user will be defaulted as the owner. | UI States |
| **6.e** | The user will click save button to store the document. | UI States |
| **6.f** | On save the user will be redirected to the dashboard. | Sequence diagram for Adding a Document |
| **6.g** | The new document will be present in the list on the dashboard. | Sequence diagram for Viewing a Document |
| **7.a** | When the user clicks edit, he will be redirected to the edit page. | UI States |
| **7.b** | The user can change the title and body of the document. | UI States |
| **7.c** | The user can add new users to the shared with list. | Data Design |
| **7.d** | The user can check the read-only box to make the document read-only for the specific user. | Data Design |
| **7.e** | The user will click the update button to save. | Sequence diagram for Updating a Document |
| **7.f** | On save the user will be redirected to the dashboard. | UI States |
| **7.g** | The changes to the document will be reflected on the dashboard. | Sequence diagram for Viewing a Document |
| **8.a** | When the user clicks delete the document will be deleted and the list is updated in the dashboard. | Sequence diagram for Deleting a Document |

**Development Setup:**

Front End:

* Install Angular CLI 8.2.1
* Run Yarn
* Type `yarn start`
* The UI is now started at localhost:4200

Back End:

* Install NodeJS 10.16.0
* Install YARN 1.6.0
* Install Express generator v4.16.1
* Type yarn into console
* Type ` set DEBUG=myapp:\* & npm start` to start the server
* The server is up at localhost:3000

Database:

* Login to the mongoDb atlas
* Create a collection
* Create a User and Document table
* Add the current IP to whitelist in Network Settings
* Remove all empty indexes from the tables
* Click on connect and select application
* Copy the connection string
* Replace the connection string in the Mongo Service

**Testing Plan:**

**Introduction:**

This is the test plan to test the basic document management system. The scope of the test plan includes

* Authenticating a user
* Adding a new user
* Adding new document
* Editing a document
* Deleting a document
* Viewing the users allowed list of documents after all the above actions
* Logging out the user.

**Test Procedure:**

For this test plan there will be following use cases

|  |  |
| --- | --- |
| **Use Case ID** | **Decription** |
| **UC1** | Authenticate with a user |
| **UC2** | Add a new user |
| **UC3** | Create new document |
| **UC4** | Edit an existing document |
| **UC5** | Delete an existing document |
| **UC6** | Share the document with others |
| **UC7** | The document is locked when other users are editing it |
| **UC8** | Logout |

The use cases discussed above will be covered by the following test cases.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test Case ID** | **Test Description** | **Use Case Id** | **Expected Result** | **Actual Result** |
| **TC1** | Login with an existing user with correct credentials | **UC1** | The user should login and see his name in the header and a logout button |  |
| **TC2** | Login with an existing user with incorrect credentials | **UC1** | The login should fail |  |
| **TC3** | There should be a link to add new user | **UC2** | The link should take the user to a register form |  |
| **TC4** | Create a new user with unique username | **UC2** | The user should be created and redirected to dashboard and the user should see his name and logout |  |
| **TC5** | Create a user with duplicate user name | **UC2** | The user creation should fail |  |
| **TC6** | Add new document button should be visible | **UC3** | There should be a button to add new document and this will redirect the user to add screen. |  |
| **TC7** | The user should be able to add new document | **UC3** | After save the user should see the new document in the dashboard |  |
| **TC8** | The user should see an edit button for the document owned by him | **UC4** | There should be a button to edit and it should take the user to the edit form. |  |
| **TC9** | The user should see a button for the document shared with him in write mode | **UC4** | There should be a button to edit and it should take the user to the edit form. |  |
| **TC10** | The user should not see a button for the document shared with him in read only mode | **UC4** | The button to edit should be hidden |  |
| **TC11** | The user should be able to edit the content of the document | **UC4** | On save the user will be redirected to dashboard and the changes should be visible. |  |
| **TC12** | The user should be able to see the delete button for owned and documents shared in write mode | **UC5** | There should be a link to delete. |  |
| **TC13** | The user should be able to delete the document | **UC5** | After clicking delete the document should be deleted and the dashboard should no longer show it. |  |
| **TC14** | The user should not see delete for read only documents | **UC5** | The delete button should not be visible. |  |
| **TC15** | The user should be able to share the document in read mode | **UC6** | After the user adds a new user in read mode, the other user should see the document and should not be able to edit or delete it. |  |
| **TC16** | The user should be able to share the document in write mode | **UC6** | After the user adds a new user in write mode, the other user should see the document and should be able to edit or delete it. |  |
| **TC17** | The document should be locked when others are using it. | **UC7** | The document actions should be hidden and the status locked and locked by should be visible to current user. |  |
| **TC18** | The user should be able to logout successfully | **UC8** | The logout should not error |  |

**Requirements Traceability Matrix:**

|  |  |  |
| --- | --- | --- |
| **Requirement Number** | **Requirement Description** | **Test Case Number** |
| **1.a** | Authenticated users should go the dashboard | **TC1** |
| **1.b** | Authenticated users will see their username in the header. | **TC1** |
| **1.c** | Unauthenticated users should be redirected to the login page. | **TC1** |
| **2.a** | The user should be able to login using a username and password | **TC1** |
| **2.b** | The user should not be allowed to proceed if the password is incorrect | **TC2** |
| **3.a** | The user should be shown a link to register to use the application | **TC3** |
| **3.b** | Duplicate usernames should not be allowed registration. | **TC4** |
| **3.c** | Error should be shown for duplicate user registration. | **TC4** |
| **4.a** | Authenticated users will be displayed a logout button. | **TC1** |
| **4.b** | The logout will kill the client and server session. | **TC18** |
| **5.a** | The user will see all the documents he is permissible to view. | **TC15, TC16** |
| **5.b** | If the user can create a new document, he will see a create new button. | **TC6** |
| **5.c** | If the user can edit a document edit button will be displayed. | **TC8** |
| **5.d** | If the user can delete a document delete button will be visible. | **TC12, TC14** |
| **5.e** | The owner of a document will be visible next to each document. | **TC15, TC16** |
| **5.f** | If the document is locked the edit button will be disabled. | **TC17** |
| **5.g** | A locked document will have the name of the user who has checked out the document. | **TC17** |
| **6.a** | When the user clicks add he will be redirected to the add page. | **TC6** |
| **6.b** | The user will add the title through an input text box. | **TC7** |
| **6.c** | The user will add the body through a text area. | **TC7** |
| **6.d** | The current user will be defaulted as the owner. | **TC7** |
| **6.e** | The user will click save button to store the document. | **TC7** |
| **6.f** | On save the user will be redirected to the dashboard. | **TC7** |
| **6.g** | The new document will be present in the list on the dashboard. | **TC7** |
| **7.a** | When the user clicks edit, he will be redirected to the edit page. | **TC8** |
| **7.b** | The user can change the title and body of the document. | **TC11** |
| **7.c** | The user can add new users to the shared with list. | **TC11** |
| **7.d** | The user can check the read-only box to make the document read-only for the specific user. | **TC11** |
| **7.e** | The user will click the update button to save. | **TC11** |
| **7.f** | On save the user will be redirected to the dashboard. | **TC11** |
| **7.g** | The changes to the document will be reflected on the dashboard. | **TC11** |
| **8.a** | When the user clicks delete the document will be deleted and the list is updated in the dashboard. | **TC14** |