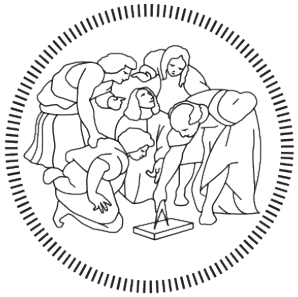
Documentation for ITW project

Jerry Huang

2023-2024



POLITECNICO
MILANO 1863

Contents

1 Introduction **3**
1.1 Pure HTML . . . 3
1.2 JavaScript . . . 4
1.3 Additional notes . . . 4

2 Database design **5**
2.1 Database design . . . 5
2.2 Database diagrams . . . 7

3 HTML version **8**
3.1 Project analysis . . . 8
3.2 Project design . . . 9

4 JavaScript version **21**
4.1 Project analysis . . . 21
4.2 Project design . . . 22

5 Final considerations **33**

List of Figures

1 Image from the teacher's project requirements. . . . 3
2 ER diagram. . . . 7
3 Logical schema. . . . 7
4 Login and register. . . . 12
5 Document access and move. . . . 13
6 Document manager. Folder manager. . . . 14
7 Login sequence. . . . 15
8 Register sequence. . . . 16
9 Index messages sequence. . . . 16
10 Home page sequence. . . . 17
11 Accessing a folder sequence. . . . 17
12 Accessing a document's details sequence. . . . 18
13 Creating a document sequence. . . . 18
14 Creating a folder sequence. . . . 19
15 Moving a document sequence. . . . 20
16 Login and register. . . . 23
17 Home. . . . 24
18 Login sequence. . . . 27
19 Register sequence. . . . 28
20 Home page loading sequence. . . . 28
21 Creating a document sequence. . . . 29
22 Creating a folder sequence. . . . 30
23 Moving a document sequence. . . . 31
24 Deleting a document sequence. . . . 31
25 Deleting a folder sequence. . . . 32
26 Logout sequence. . . . 32

1 Introduction

The project involves the creation of a web application dedicated to document management, developed using technologies such as Java, HTML, and CSS. The website will be available in two variants for client-side scripting: one based on Thymeleaf and the other on JavaScript.
Each version will provide users with specific functionalities based on the technology used. Users will have access to various options for managing, viewing, and organizing documents.
The application provides user login and registration functionality via a public page. It then checks the email and password, ensuring the syntactic validity of both and the uniqueness of the username. Each folder is characterized by a name, creation date, and can contain other folders or documents. Documents include information such as owner, name, creation date, summary, and document type.

1.1 Pure HTML

After the user logs in, a homepage appears displaying a tree of their folders and subfolders. On

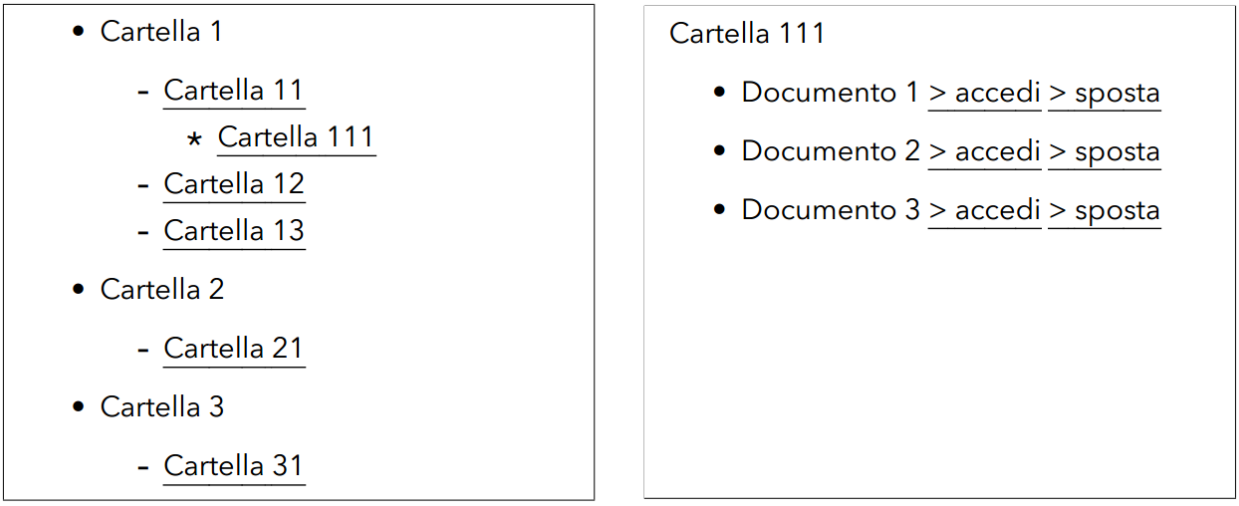


Figure 1: Image from the teacher's project requirements.

the homepage, the user can select a folder and access a contents page, which shows a list of folders and documents within the selected folder. Each document has two links: "access" and "move". If the user clicks on "access", the document page appears showing the details of the selected document. If "move" is clicked, the homepage reappears with a message saying "You are moving document X from folder Y. Choose a destination folder." The folder containing the document is not selectable, and its name is highlighted. Once the user selects the destination folder, the document is moved, and the contents page of the destination folder is displayed. Every page, except the homepage, contains a link to return to the previous page. The user can log out from any page. Finally, a content management page, accessible from the homepage, allows the user to create a top-level folder, a subfolder inside an existing folder, or a document inside a folder.

1.2 JavaScript

The application still supports login and registration, and in addition to the syntax validation of the email, the equality of the "password" and "repeat password" fields is also checked on the client and server side. After the user logs in, the entire application is implemented on a single page. Error messages are displayed within the page for the user to see. The movement of documents or folders occurs via drag and drop. Next to each folder, two labeled buttons appear "add subfolder" and "add document", they respectively trigger a form to input the name of the subfolder or the document details. Finally, a folder called "trash bin" is added; dragging a document into the trash results in its deletion. Before deleting the document, a confirmation window appears. Deleting a folder results in the complete and recursive deletion of all data (documents and subfolders).

1.3 Additional notes

- Since the project involves developing two versions of the same application with only minor modifications, the database remains unchanged as it needs to be accessed by both versions.
- Upon registration, each user is assigned a root folder. All subsequent folders and documents must be created within this root folder. This approach simplifies the database by maintaining a clear, linear tree structure, similar to the organization of a traditional file system on a PC.

2 Database design

2.1 Database design

Data requirements [Entities Attributes Relationships]
The application provides user login and registration functionality via a public page. It then checks the email and password, ensuring the syntactic validity of both and the uniqueness of the username. Each folder is characterized by a name, creation date, and contain other folders or documents. Documents include information such as owner, name, creation date, summary, and document type.

SQL Code

```
CREATE TABLE `document` (
  `DocumentId` int unsigned NOT NULL AUTO_INCREMENT,
  `Owner` int unsigned NOT NULL,
  `Name` varchar(45) NOT NULL,
  `Date` date NOT NULL,
  `Type` varchar(45) NOT NULL,
  `FolderLocation` int unsigned NOT NULL,
  `Summary` varchar(255) NOT NULL,
  PRIMARY KEY (`DocumentId`),
  KEY `FK1_idx` (`FolderLocation`),
  KEY `FK2_idx` (`Owner`),
  CONSTRAINT `FK1` FOREIGN KEY (`FolderLocation`) REFERENCES `folder`
     (`FolderId`) ON DELETE CASCADE ON UPDATE CASCADE,
  CONSTRAINT `FK2` FOREIGN KEY (`Owner`) REFERENCES `user` (`UserId`) ON
     DELETE CASCADE ON UPDATE CASCADE
) ENGINE=InnoDB AUTO_INCREMENT=73 DEFAULT CHARSET=utf8mb4
   COLLATE=utf8mb4_0900_ai_ci

CREATE TABLE `folder` (
  `FolderId` int unsigned NOT NULL AUTO_INCREMENT,
  `Owner` int unsigned NOT NULL,
  `Name` varchar(45) NOT NULL,
  `Date` date NOT NULL,
  `ParentFolder` int unsigned DEFAULT NULL,
  PRIMARY KEY (`FolderId`),
  UNIQUE KEY `Unique_keys` (`Owner`,`Name`,`ParentFolder`),
  KEY `FK3_idx` (`Owner`),
  KEY `FK4_idx` (`ParentFolder`),
  CONSTRAINT `FK3` FOREIGN KEY (`Owner`) REFERENCES `user` (`UserId`) ON
     DELETE CASCADE ON UPDATE CASCADE,
  CONSTRAINT `FK4` FOREIGN KEY (`ParentFolder`) REFERENCES `folder`
     (`FolderId`) ON DELETE CASCADE ON UPDATE CASCADE
) ENGINE=InnoDB AUTO_INCREMENT=199 DEFAULT CHARSET=utf8mb4
   COLLATE=utf8mb4_0900_ai_ci

  CREATE TABLE `user` (
  `UserId` int unsigned NOT NULL AUTO_INCREMENT,
  `Username` varchar(45) NOT NULL,
  `Password` varchar(60) NOT NULL,
  `Email` varchar(45) NOT NULL,
  PRIMARY KEY (`UserId`),
  UNIQUE KEY `Username_UNIQUE` (`Username`)
) ENGINE=InnoDB AUTO_INCREMENT=49 DEFAULT CHARSET=utf8mb4
   COLLATE=utf8mb4_0900_ai_ci
```

Table schema

```
Document(
 DocumentId, //Unique Id of the document
 Owner, //Id of the user who owns the document
 Name, //Name of the document
 Date, //Date of creation
 Type, //Type of document
 FolderLocation //Id of the folder containing the document
)

Folder(
 FolderId, //Unique Id of the folder
 Owner, //Id of the user who owns the folder
 Name, //Name of the folder
 Date, //Date of creation
 ParentFolder //Id of the parent folder containing the folder
)

User(
 UserId, //Unique Id of the user
 Username, //Unique username of the user
 Password, //Hashed text of the password
 Email //Email of the user
)
```

2.2 Database diagrams

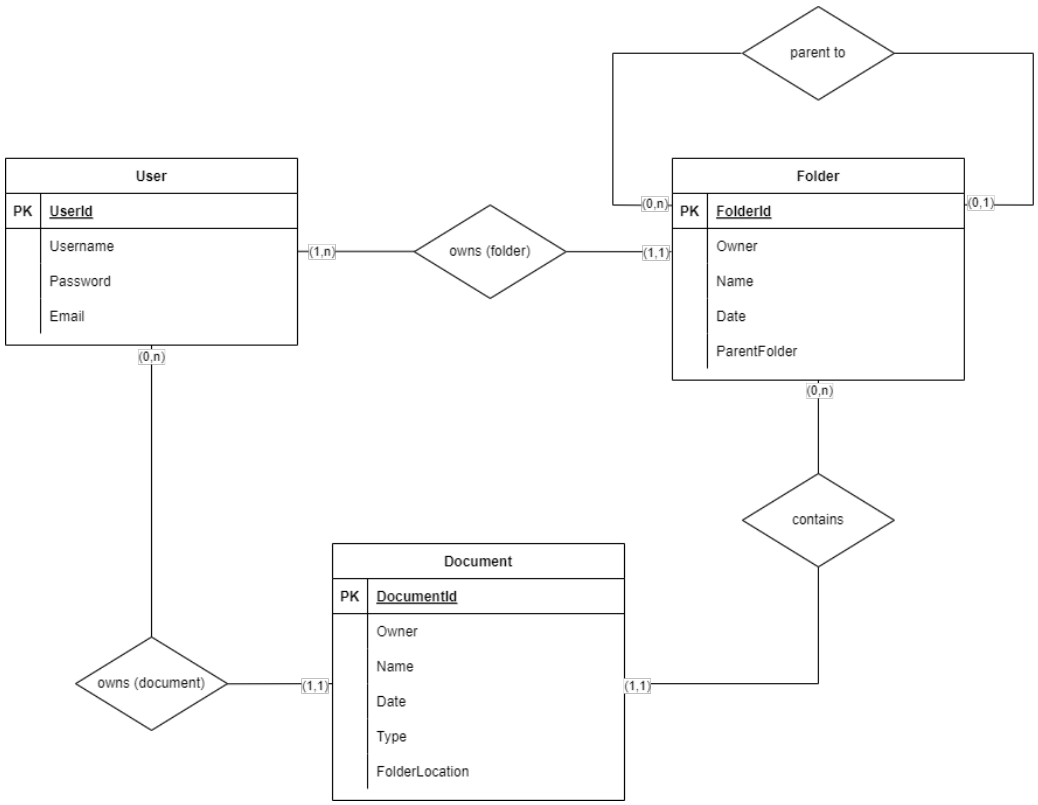


Figure 2: ER diagram.

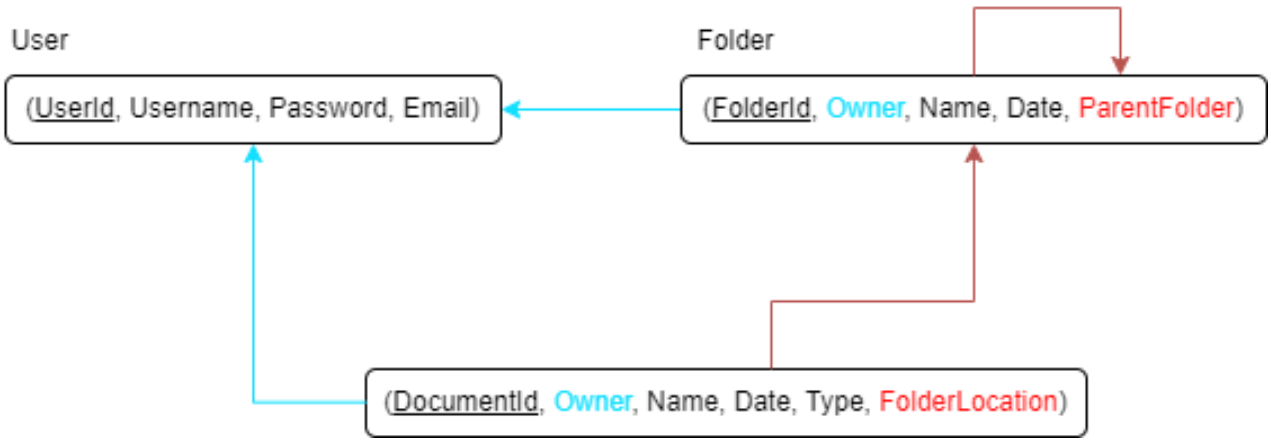


Figure 3: Logical schema.

3 HTML version

3.1 Project analysis

Application requirements [Pages View components Events Actions]
The application provides user login and registration functionality via a public page. It then checks the mail and password, ensuring the syntactic validity of both and the uniqueness of the username. Each folder is characterized by a name, creation date, and can contain other folders or documents. Documents include information such as owner, name, creation date, summary, and document type. After the user logs in, a homepage appears displaying a tree of their folders and subfolders. On the homepage, the user can select a folder and access a contents page, which shows a list of folders and documents within the selected folder. Each document has two links: "access" and "move". If the user "clicks on "access", the document page appears showing the details of the selected document. If "move" is clicked, the homepage reappears with a message saying "You are moving document X from folder Y. Choose a destination folder." The folder containing the document is not selectable, and its name is highlighted. Once the user selects the destination folder, the document is moved, and the contents page of the destination folder is displayed. Every page, except the homepage, contains a link to return to the previous page. The user can log out from any page. Finally, a content management page, accessible from the homepage, allows the user to create a top-level folder, a subfolder inside an existing folder, or a document inside a folder.

3.2 Project design

Components

Beans (Java)

- Document
- Folder
- User

Data Access Objects (Java)

- DocumentDao
  - createDocument
  - checkFolderPath
  - getDocumentsByFolderId
  - getDocumentById
  - moveDocument
- FolderDao
  - getRootFolder
  - setSubFolders
  - createFolder
  - checkFolderPath
  - isFolderAccesible
  - getFolderById
- UserDao
  - getUsernameById
  - loginUser
  - registerUser
  - isUsernameTaken

Views (HTML)

- index (page showing the log in and register forms)
- Home (page showing the tree of folders and documents)
- AvailableFolders (page showing all the folders for document transfer)
- DocumentDetails (page showing all the info on a selected document)
- DocumentManager (page showing form for document creation)
- FolderContent (page showing all documents and folders inside of a folder)
- FolderManager (page showing form for folder creation)

Controllers (Java)

- AvailableFolders (Printing the folders in which the document can be moved to)
- CreateDocument (Creates a document in the selected path)
- CreateFolder (Creates a folder in the selected path)
- DocumentManager (Shows the form for document creation)
- FolderManager (Shows the form for folder creation)
- GetDocumentDetails (Retrieves the document details from the DB and shows them)
- Login (Logs the user in and starts the session tracking)
- Logout (Logs the user out and invalidates the session)
- MoveDocument
- Register
- ShowFolderContent
- ShowHome
- ShowIndex

Page and view components

- index
  - Login form
  - Login anchor
  - Register form
  - Register anchor
- Home
  - Logout anchor
  - Folders tree anchors
  - Document manager anchor
  - Folder manager anchor
- FolderContent
  - Back anchor
  - Logout anchor
  - Subfolders anchors
  - Parent folder anchor
  - Documents anchors
  - Move document anchor
  - Access document anchor
- AvailableFolders
  - Back anchor
  - Logout anchor
  - Folders tree anchors
- DocumentDetails
  - Back anchor
  - Logout anchor
  - Document info text
- DocumentManager
  - Back anchor
  - Logout anchor
  - Document creation form
  - Create anchor
- FolderManager
  - Back anchor
  - Logout anchor
  - Folder creation form
  - Create anchor

Events and actions

- User signing in
  - Check credentials
  - Track session
  - Go to homepage
- User signing up
  - Check credentials
  - Create root folder
  - Go to index
- User goes to homepage
  - Check session
  - Retrieve root folder
  - Print folders tree
- User clicks on a folder in the homepage
  - Check session
  - Retrieve documents in the folder
  - Print documents in the folder
- User clicks on the access option of a document
  - Check session
  - Retrieve document's info
  - Print document's info
- User clicks on the move option of a document
  - Check session
  - Go to AvailableFolders page
- User clicks the destination folder for the moving document
  - Check session
  - Retrieve document's info
  - Check if the destination folder doesn't have document with same name
  - Move document
  - Show the destination folder's content
- User creating a new document
  - Check session
  - Check if the values for the document creation are ok
  - Creates document
  - Go to home
- User creating a new folder
  - Check session
  - Check if the values for the folder creation are ok
  - Creates folder
  - Go to home

Activity diagrams

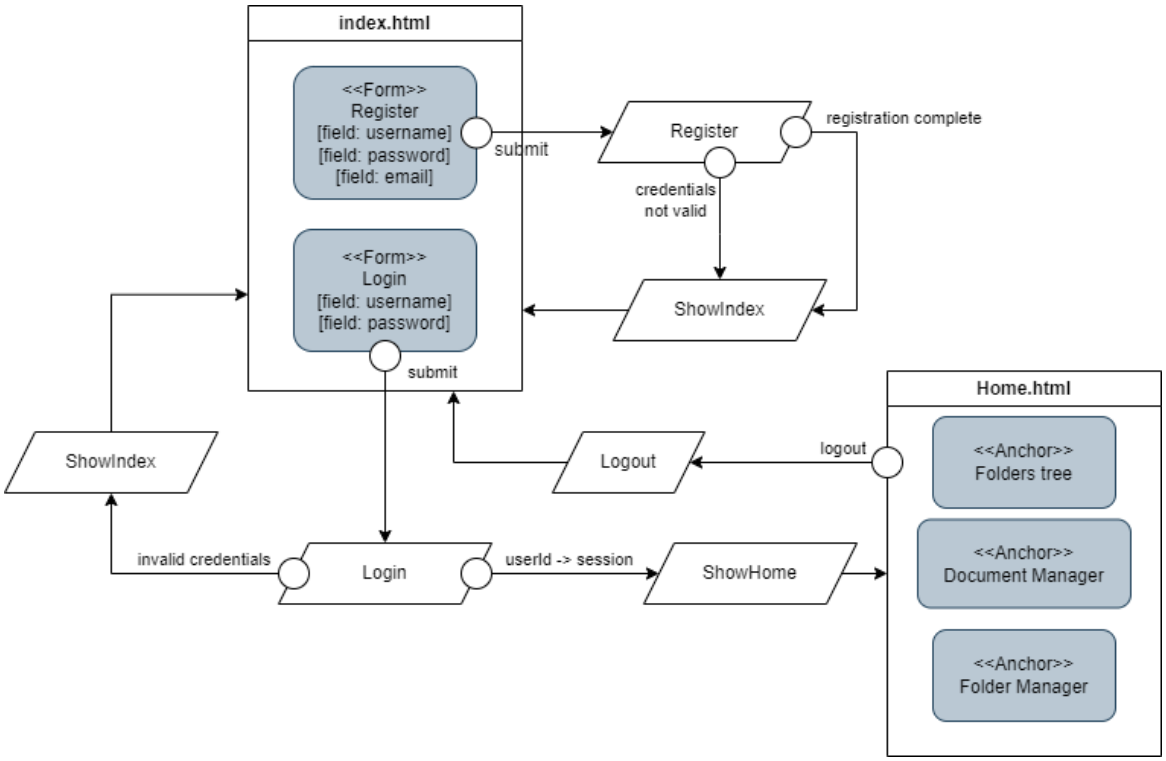


Figure 4: Login and register.

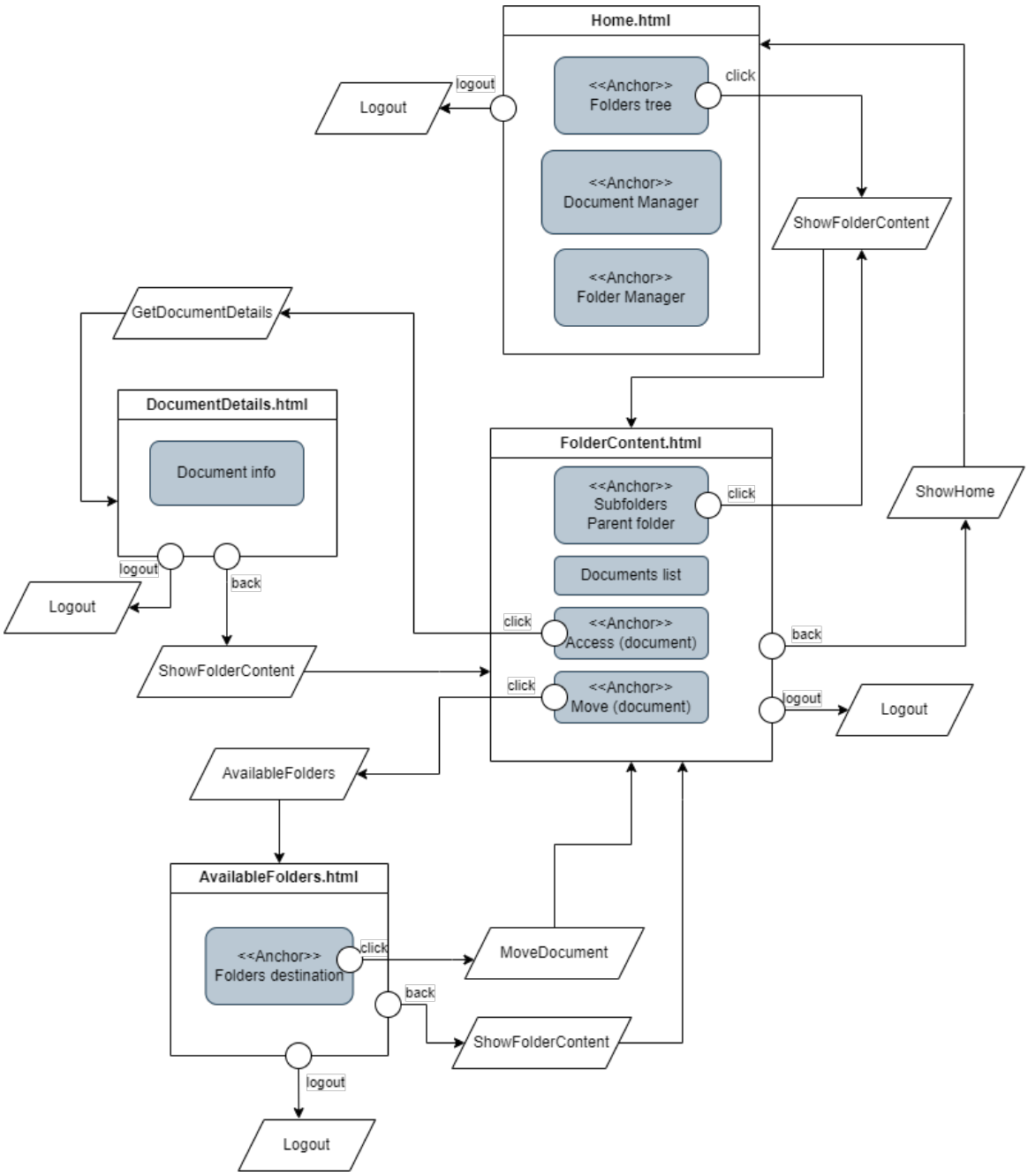


Figure 5: Document access and move.

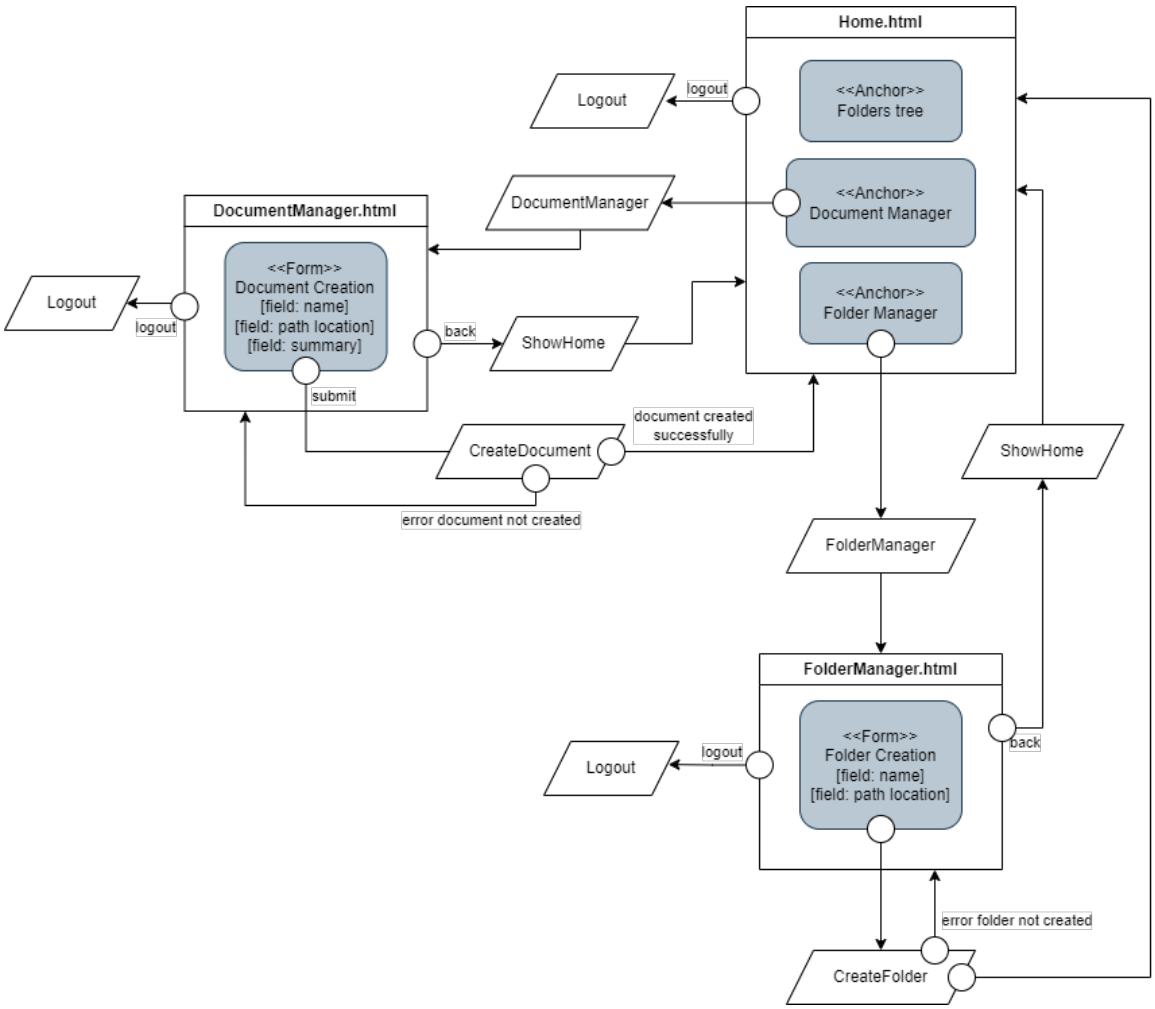


Figure 6: Document manager. Folder manager.

Sequence diagrams
Notes:

- For the sake of keeping the diagram simple to understand, it's assumed that there is no ill-intentioned user, so only some exceptions are shown. Of course everything is checked and a proper error message is sent back if anything goes wrong.
- Operations such as establishing and closing the connection are not shown for the same reason.
- Documents with same name means they have the same name and the same format.
- Folders with same name means they just have the same name.

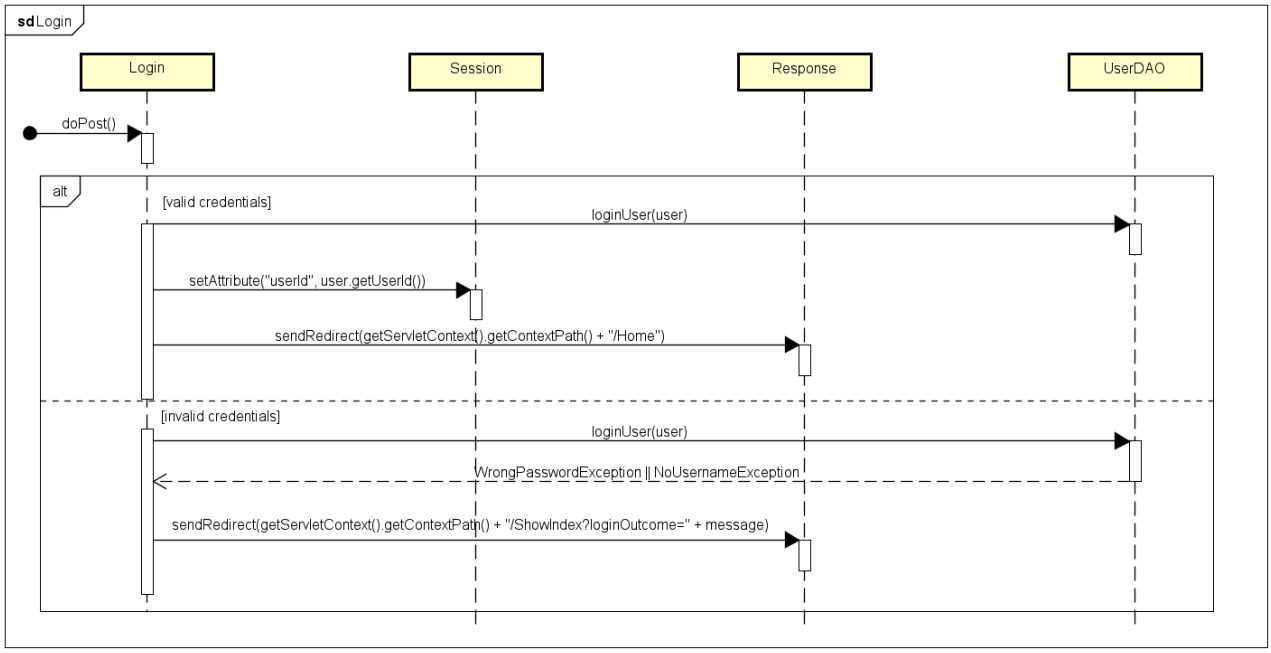


Figure 7: Login sequence.

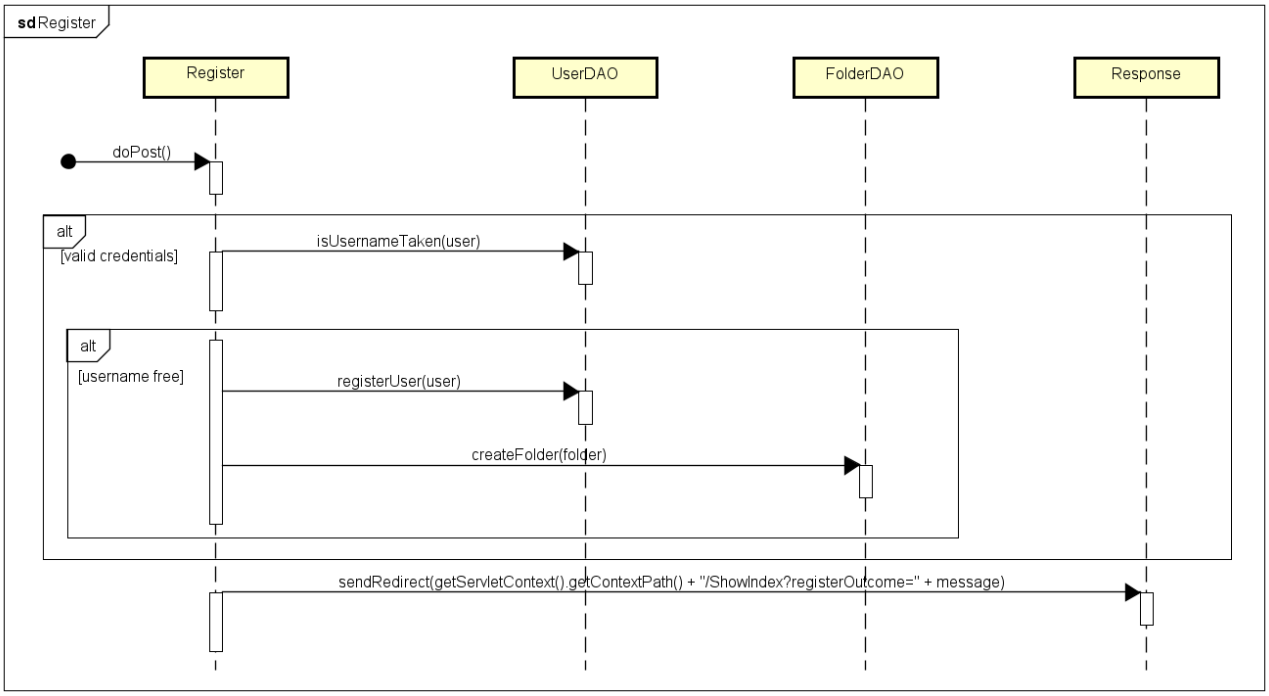


Figure 8: Register sequence.

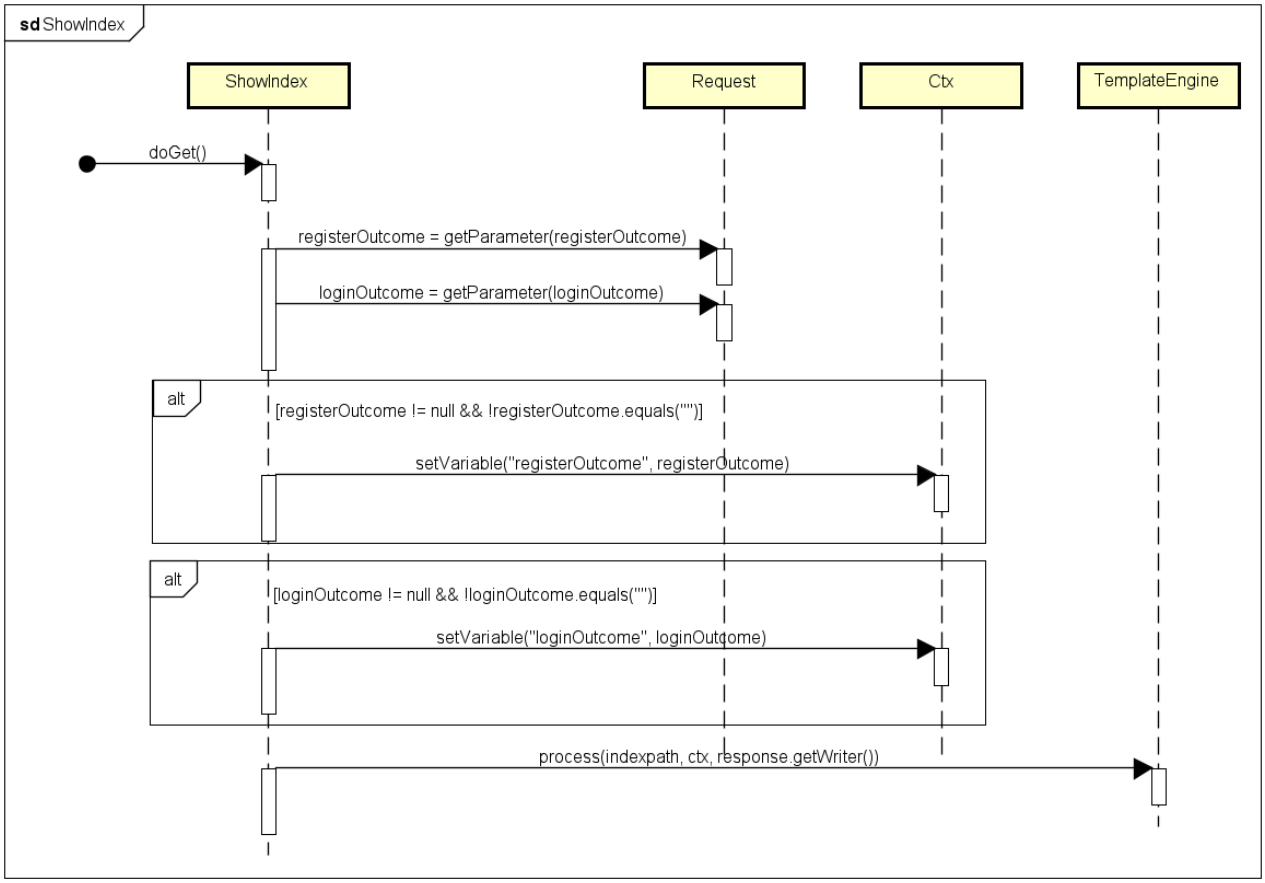


Figure 9: Index messages sequence.

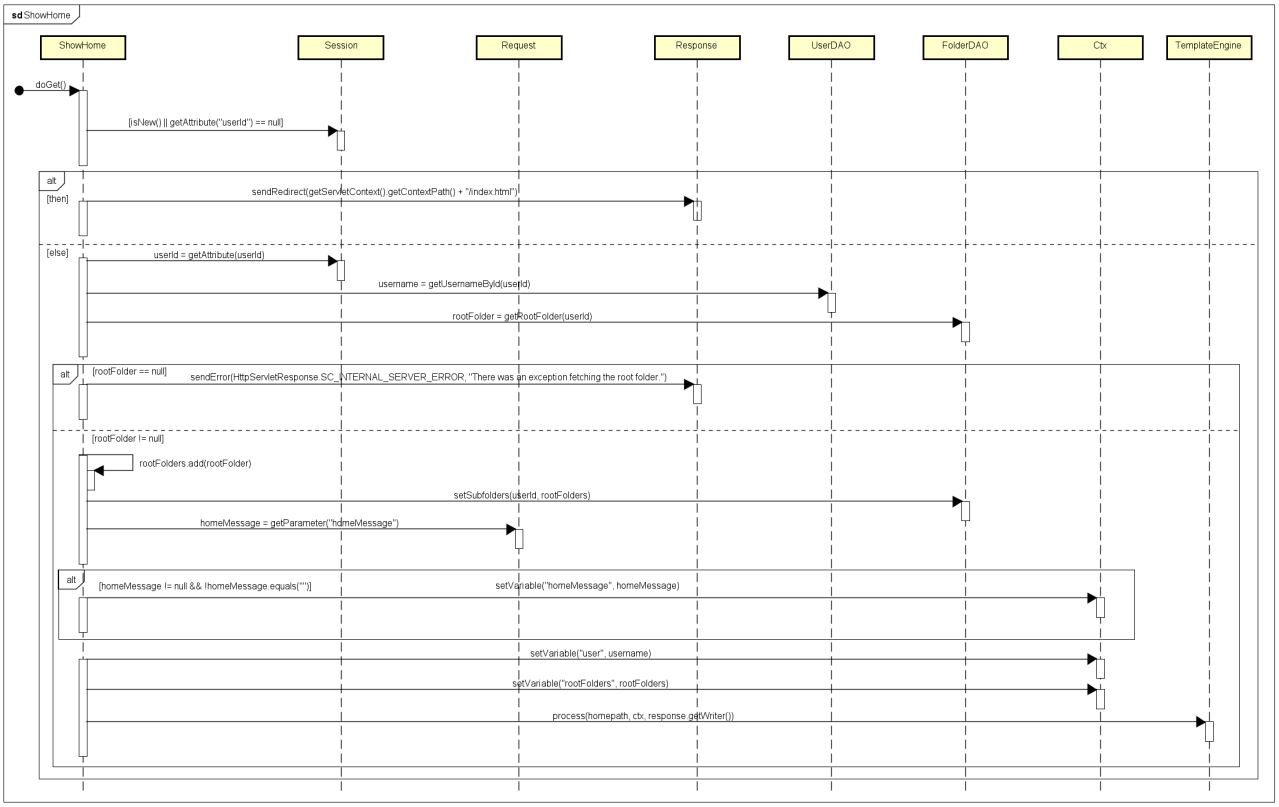


Figure 10: Home page sequence.

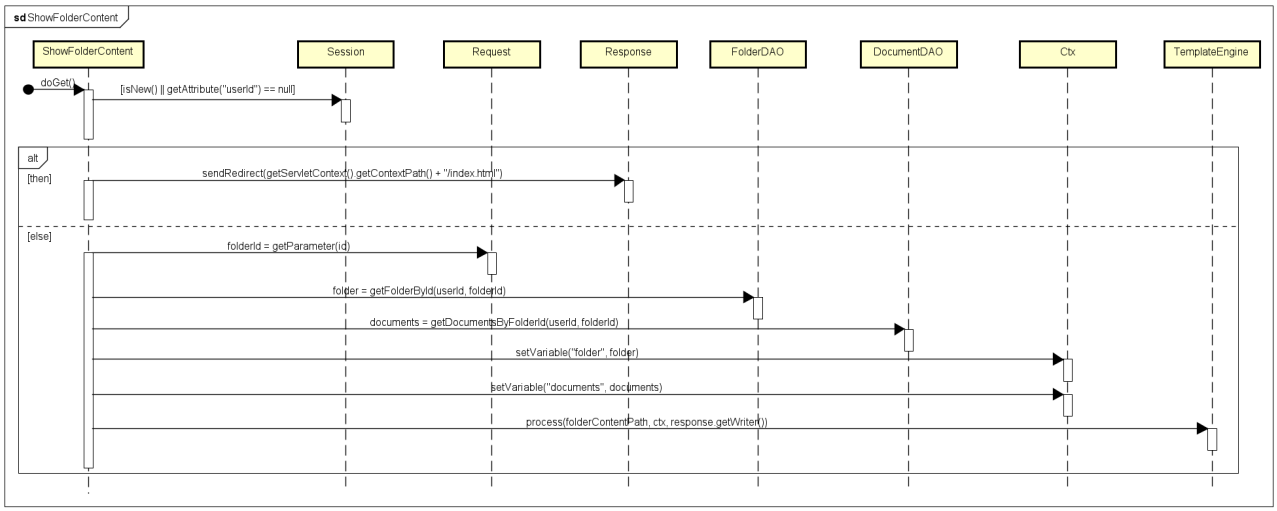


Figure 11: Accessing a folder sequence.

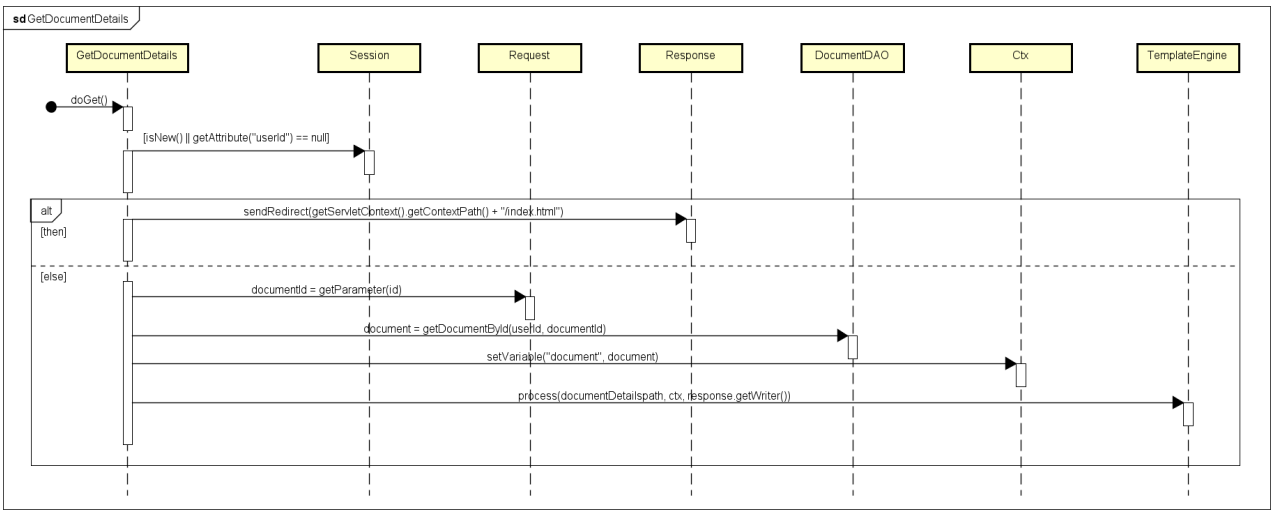


Figure 12: Accessing a document's details sequence.

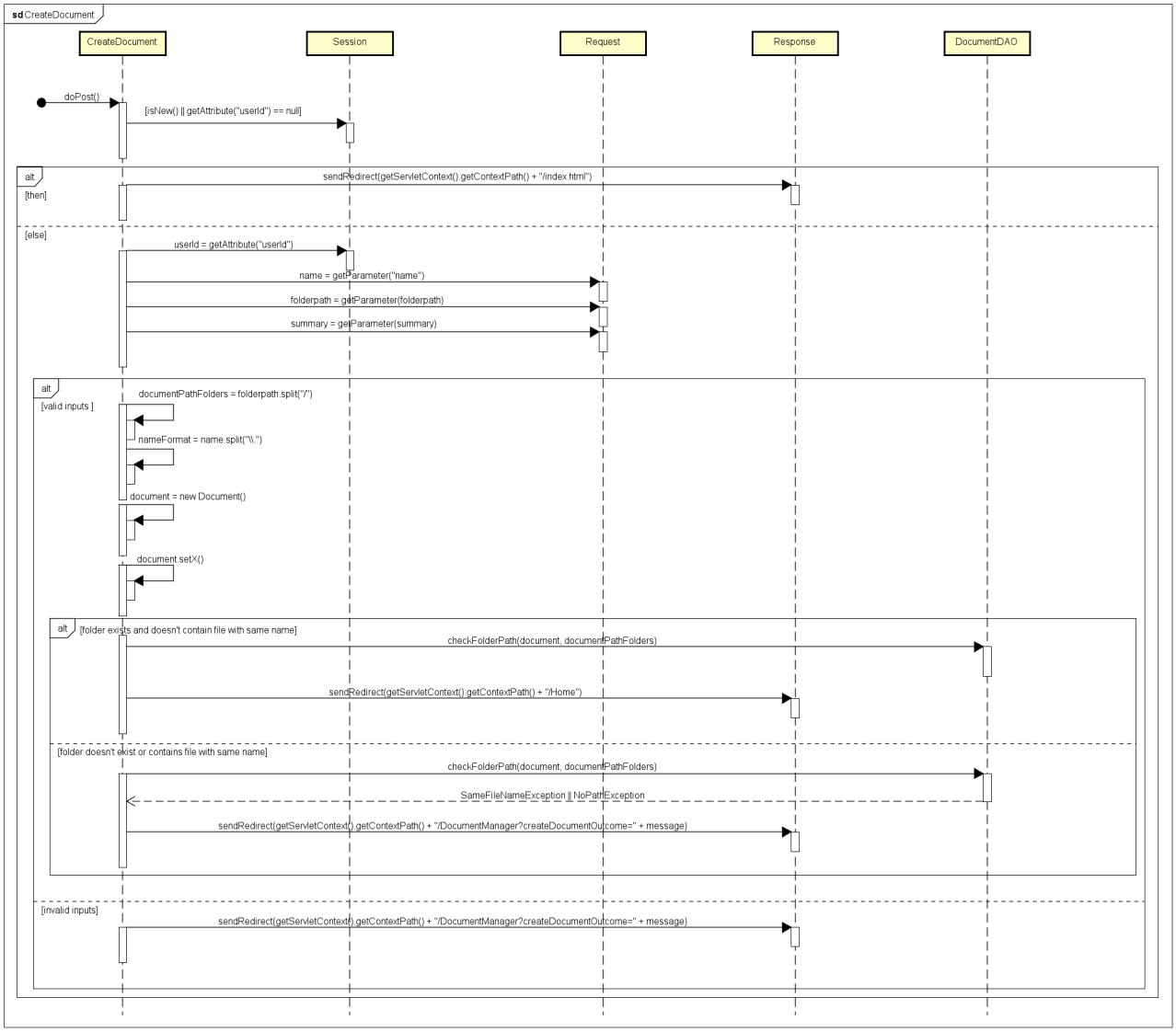


Figure 13: Creating a document sequence.

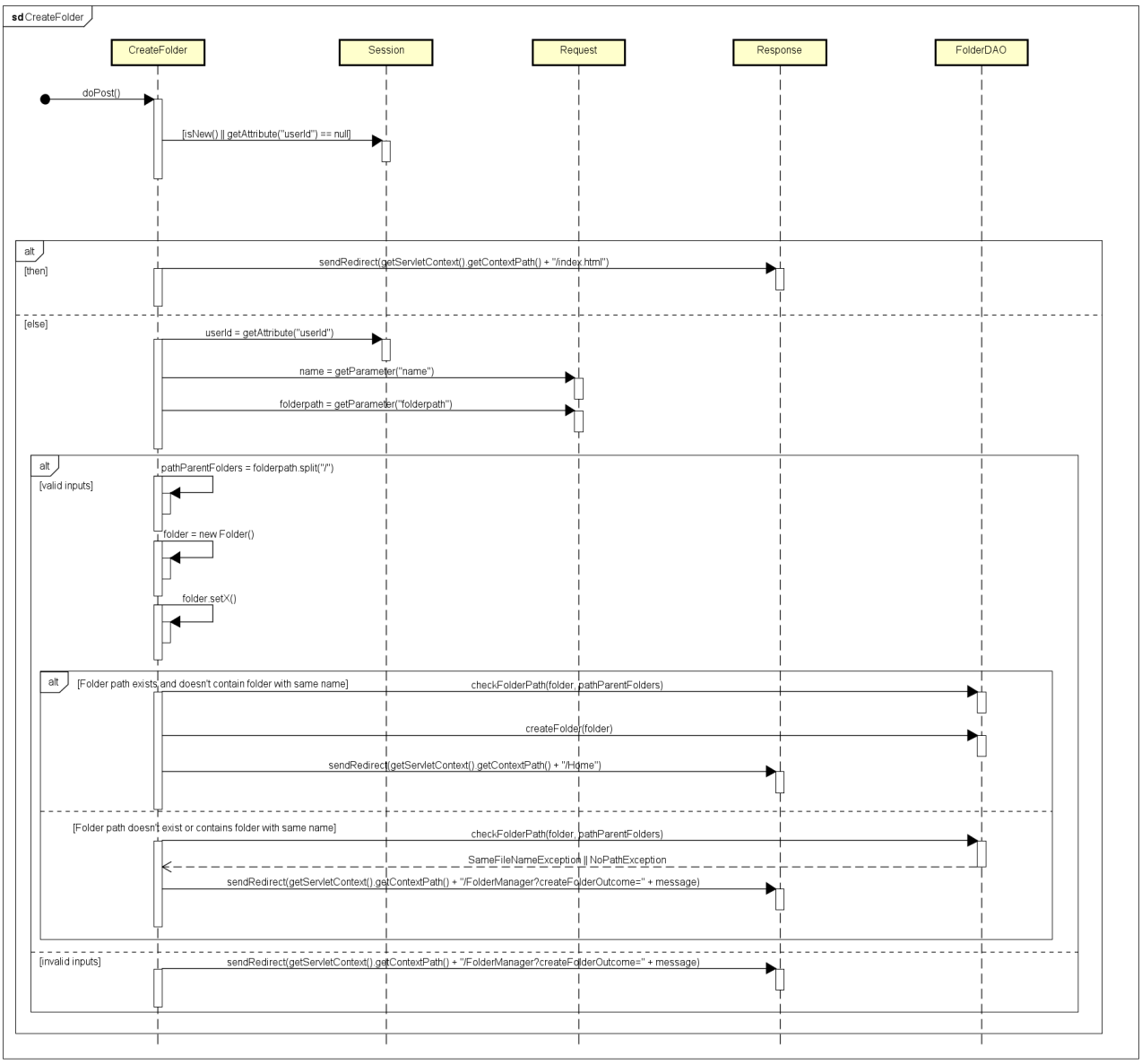


Figure 14: Creating a folder sequence.

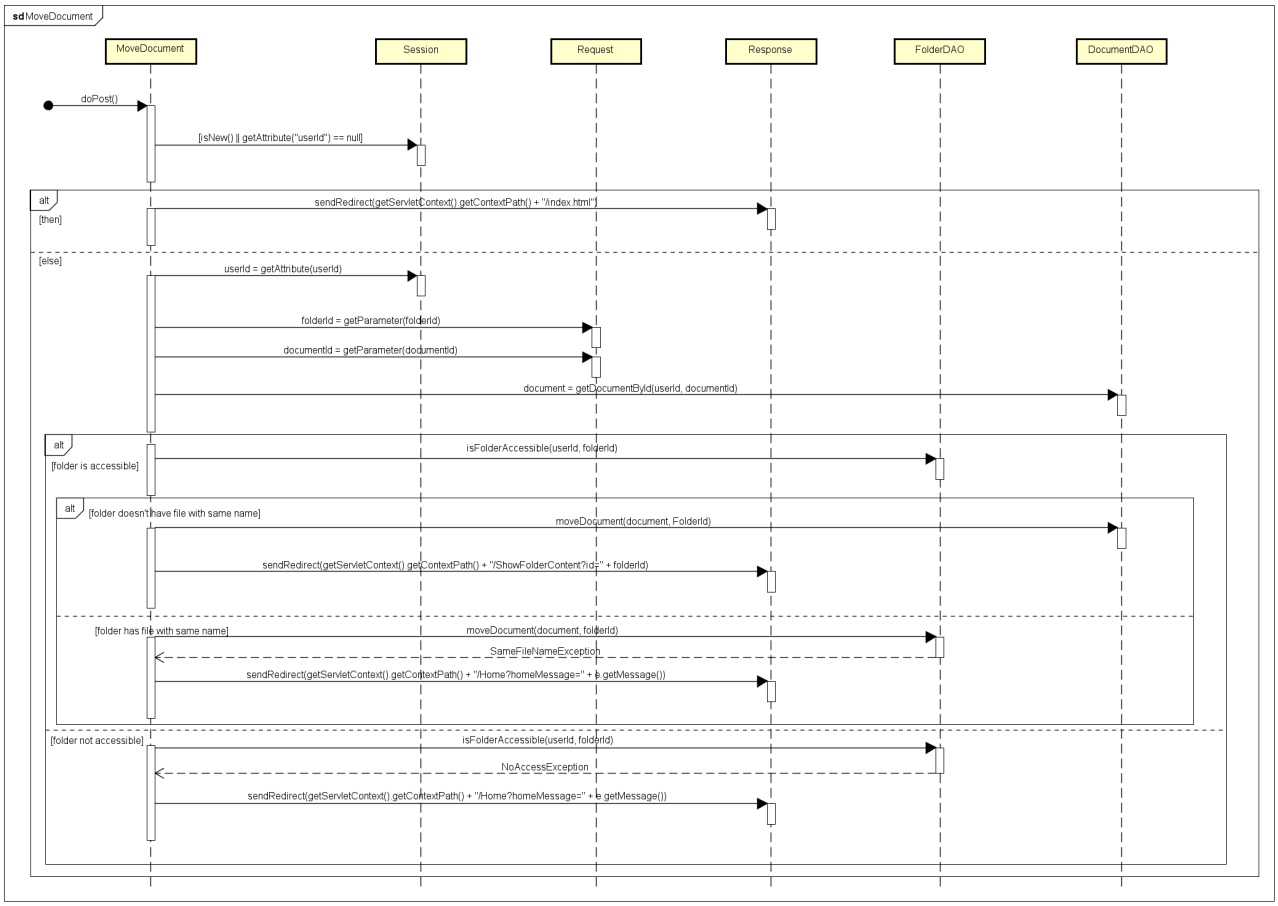


Figure 15: Moving a document sequence.

4 JavaScript version

4.1 Project analysis

Application requirements [Pages View components Events Actions]
The application still supports login and registration via forms, and in addition to the syntax validation of the email, the equality of the "password" and "repeat password" fields is also checked on the client and server side. After the user logs in, the entire application is implemented on a single page. Error messages are displayed within the page for the user to see. The movement of documents or folders occurs via drag and drop. Next to each folder, two labeled buttons appear "add subfolder" and "add document", they respectively trigger a form to input the name of the subfolder or the document details. Finally, a folder called "trash bin" is added; dragging a document into the trash results in its deletion. Before deleting the document, a confirmation window appears. Deleting a folder results in the complete and recursive deletion of all data (documents and subfolders).

Additional notes Beside the login, the application core functions work on a single homepage and the content refreshes without the use of the browser's refresh button (Single Page Architecture). Both the client-side and server-side must check the validity of the user's inputs. The server doesn't expose its core functions, instead the client's side JavaScript has to make use of the function makecall(). All objects returned from the server are JSON.

4.2 Project design

Components

Beans (Java)

- Document
- Folder
- User

Data Access Objects (Java)

- DocumentDao
  - createDocument
  - checkDocumentFolderChildren
  - getDocumentsByFolderId
  - getDocumentById
  - addDocumentsToFolder
  - getDocumentByParameters
  - isDocumentAccessible
  - moveDocument
  - deleteDocument
- FolderDao
  - getRootFolder
  - setSubFolders
  - createFolder
  - checkParentFolderChildren
  - isFolderAccessible
  - getFolderByParameters
  - getFolderById
  - isFolderRoot
  - deleteFolder
- UserDao
  - getUsernameById
  - loginUser
  - registerUser
  - isUsernameTaken

Views (HTML)

- index (page showing the log in and register forms)
- Home (page showing the tree of folders, documents and server messages)

Controllers (Java)

- CheckLogin
- CheckRegister
- CreateDocument
- CreateFolder
- DeleteDocument
- DeleteFolder
- GetFolders
- Logout
- MoveDocument

Note: because of the change in the application's structure, the Folder bean had to be modified. Subfolders and documents parameters have been added to it, so that on homepage's load the entire structure could be saved on the client's side.

Activity diagrams

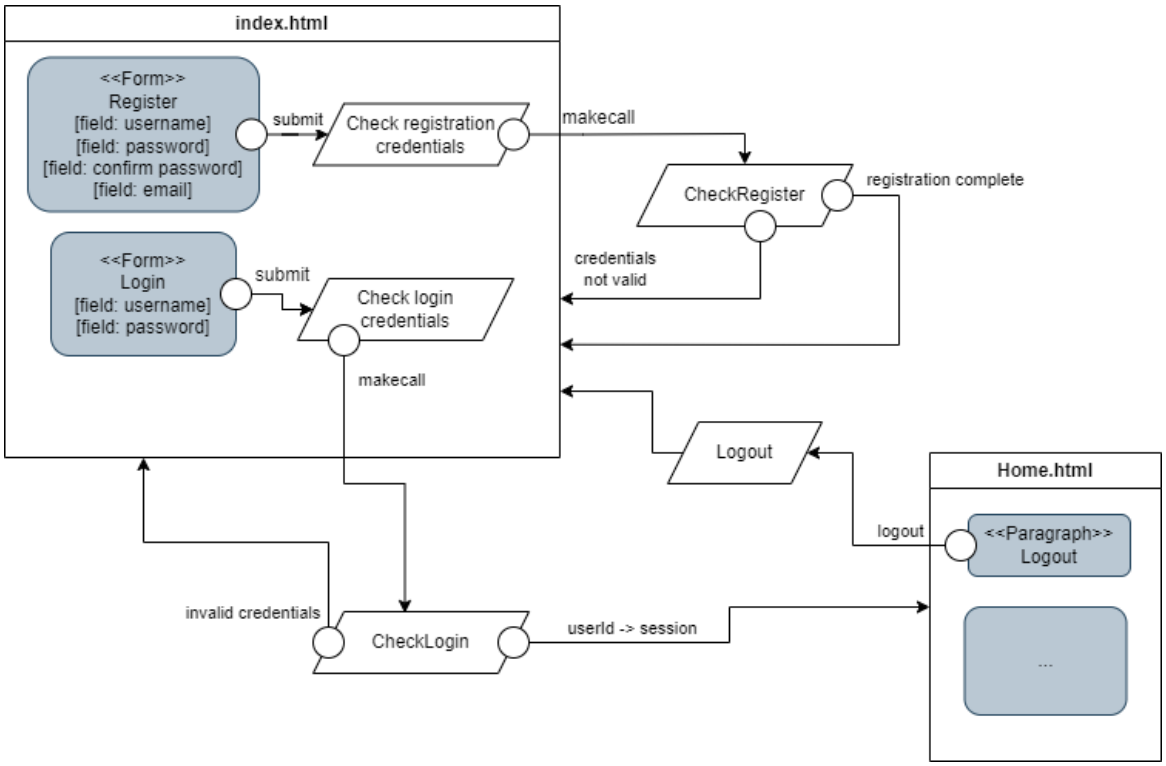


Figure 16: Login and register.

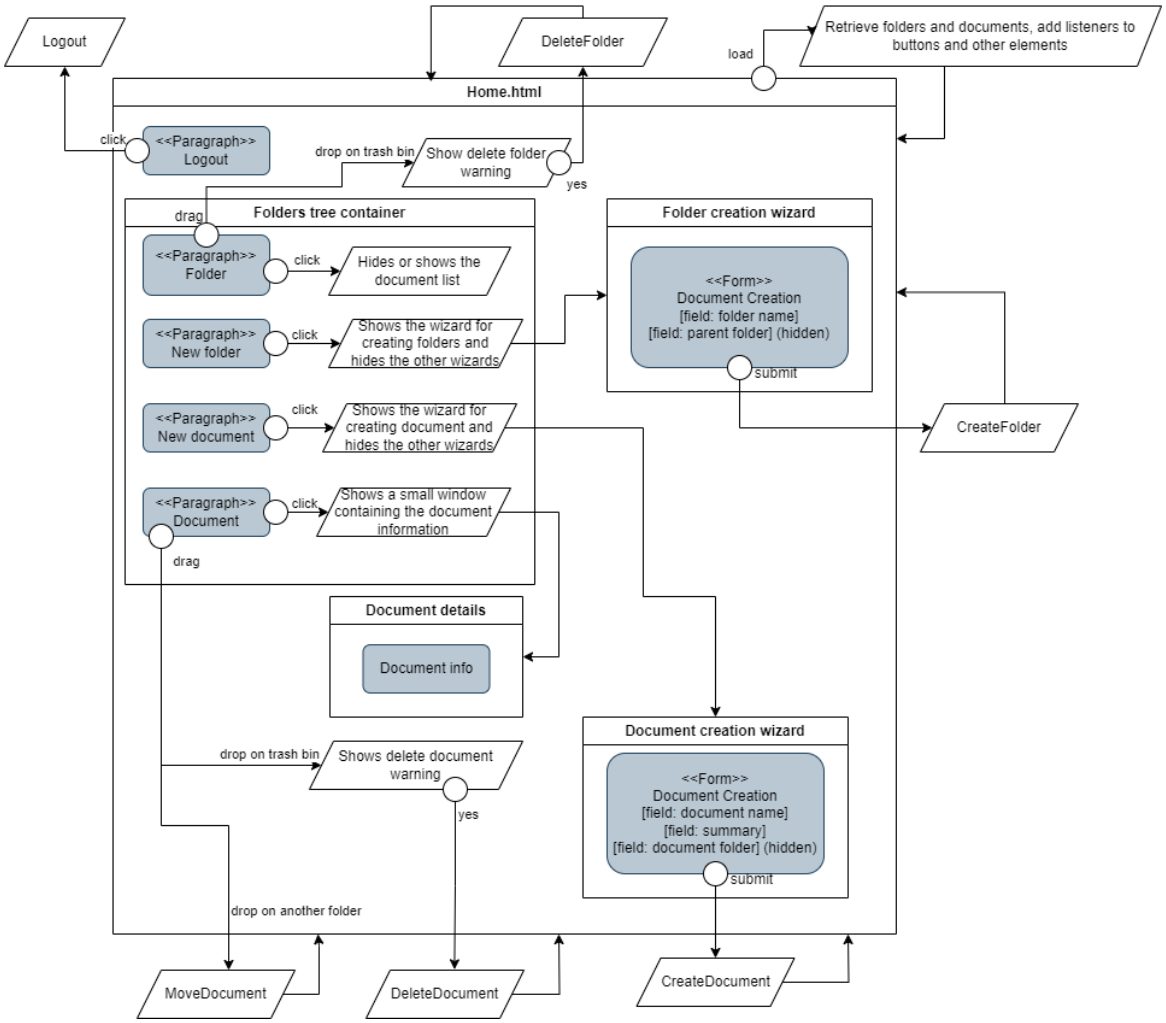


Figure 17: Home.

Events and actions Note: for the sake of keeping it short, most of the checks are omitted, obviously everything is checked and if something is wrong the user gets an error message. If the session has expired the user is kicked out and sent to the index page.

Client side		**Server side**	
Event	**Action**	**Event**	**Action**
index → login form → submit	Check form fields	POST (username, password)	Credentials check
index → register form → submit	Check form fields	POST (username, password, confirmPassword, email)	Credentials check and root folder creation
Home page → load	Updates the page with folders tree (without refreshing the entire page)	GET ()	Retrieve all the folders for the user in a tree structure
Home page → click on a folder	Show the list of documents under the folder		
Home page → click on a document	Show the details of the document		
Home page → click on logout		GET ()	Log out the user and send to index
Home page → click on new folder → submit	Show the wizard → check form fields	POST (folderName, parentFolderId)	Create a new subfolder in the parent folder
Home page → click on new document → submit	Show the wizard → check form fields	POST (documentName, summary, documentFolderId)	Create a new document in the folder
Home page → drag document → drop on a folder		POST (documentId, targetFolderId)	Move the document to the specified folder
Home page → drag document → drop on trash bin → confirm	Show the warning box for document deletion	POST (documentId)	Delete the specified document
Home page → drag folder → drop on trash bin → confirm	Show the warning box for folder deletion	POST (folderId)	Delete the specified folder and the subfolders recursively

Client side		Server side	
Event	**Controller (function)**	**Event**	**Controller (servlet)**
index → login form → submit	makeCall ()	POST (username, password)	CheckLogin
index → register form → submit	makeCall ()	POST (username, password, confirmPassword, email)	CheckRegister
Home page → load	pageOrchestrator.start (), FoldersTree.get (), buildFoldersTree (), makeCall ()	GET ()	GetFolders
Home page → refresh (after adding or removing files)	pageOrchestrator. refresh (), FoldersTree.refresh (), buildFoldersTree ()		
Home page → click on a folder			
Home page → click on a document			
Home page → click on logout	pageOrchestrator ()	GET()	Logout
Home page → click on new folder → submit	subfolderWizard (), makeCall ()	POST (folderName, parentFolderId)	CreateFolder
Home page → click on new document → submit	documentWizard (), makeCall ()	POST (documentName, summary, documentFolderId)	CreateDocument
Home page → drag document → drop on a folder	requestMoveDocument (), makeCall ()	POST (documentId, targetFolderId)	MoveDocument
Home page → drag document → drop on trash bin → confirm	deleteDocumentAlert (), makeCall()	POST (documentId)	DeleteDocument
Home page → drag folder → drop on trash bin → confirm	deleteFolderAlert (), makeCall ()	POST (folderId)	DeleteFolder

Sequence diagrams
Notes:

- For the sake of keeping the diagram simple to understand, it's assumed that there is no ill-intentioned user, so only some exceptions are shown. Of course everything is checked and a proper error message is sent back if anything goes wrong.
- Operations such as establishing and closing the connection are not shown for the same reason.
- Session tracking sequence isn't shown for the same reason, obviously if the user's session expired they get redirected to the index page.
- Documents with same name means they have the same name and the same format.
- Folders with same name means they just have the same name.
- Client side Javascript files are: fileManagementHome.js, loginManagement.js, registerManagement.js, utils.js. This last one contains the function makeCall() to do AJAX requests. Which will not be shown, using instead AJAX POST or AJAX GET for brevity. [C] means it's client-side while [S] is server-side.
- Folder deletion was done with setAutoCommit(false) in order to delete the folder, the documents and the subfolders safely. The application doesn't have to delete the files manually as the DB tables are set with ON DELETE CASCADE.

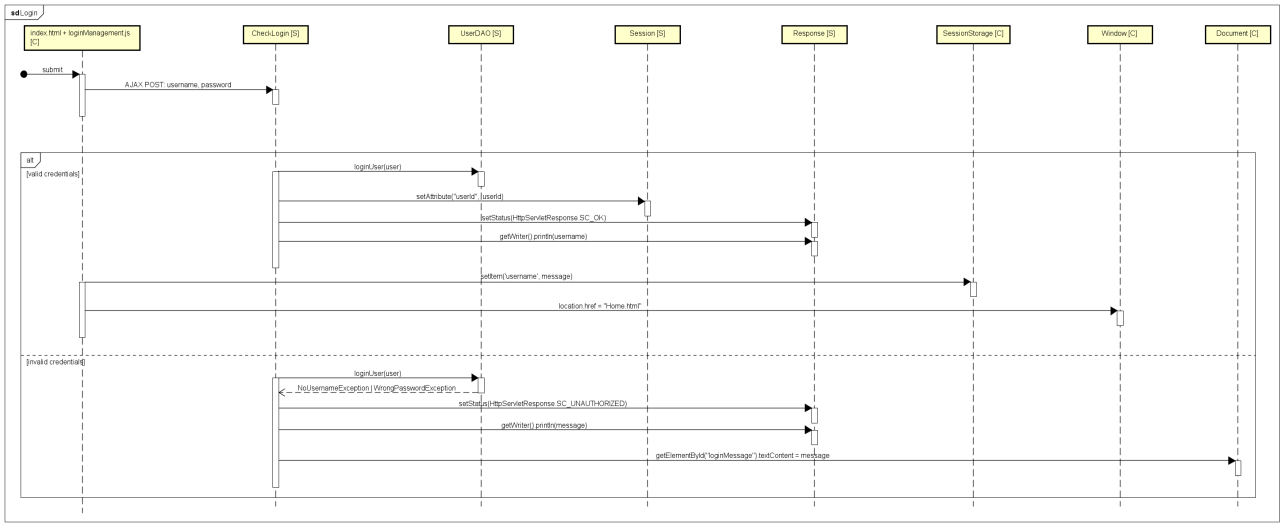


Figure 18: Login sequence.

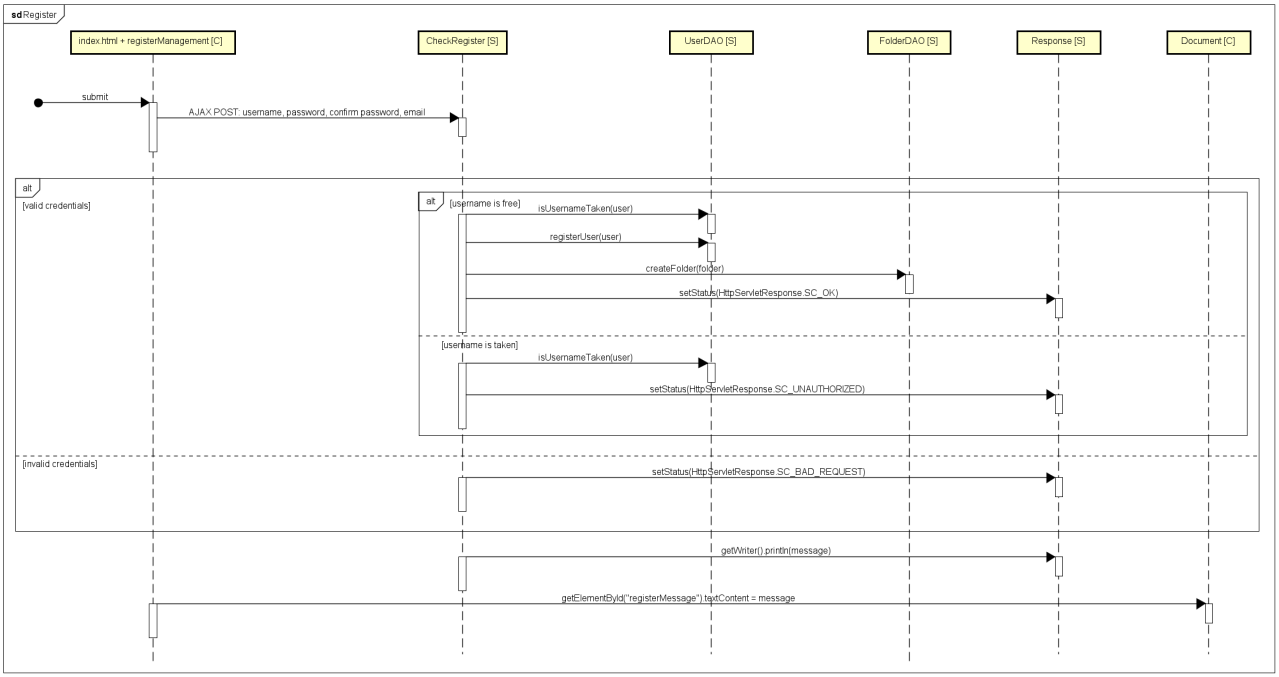


Figure 19: Register sequence.

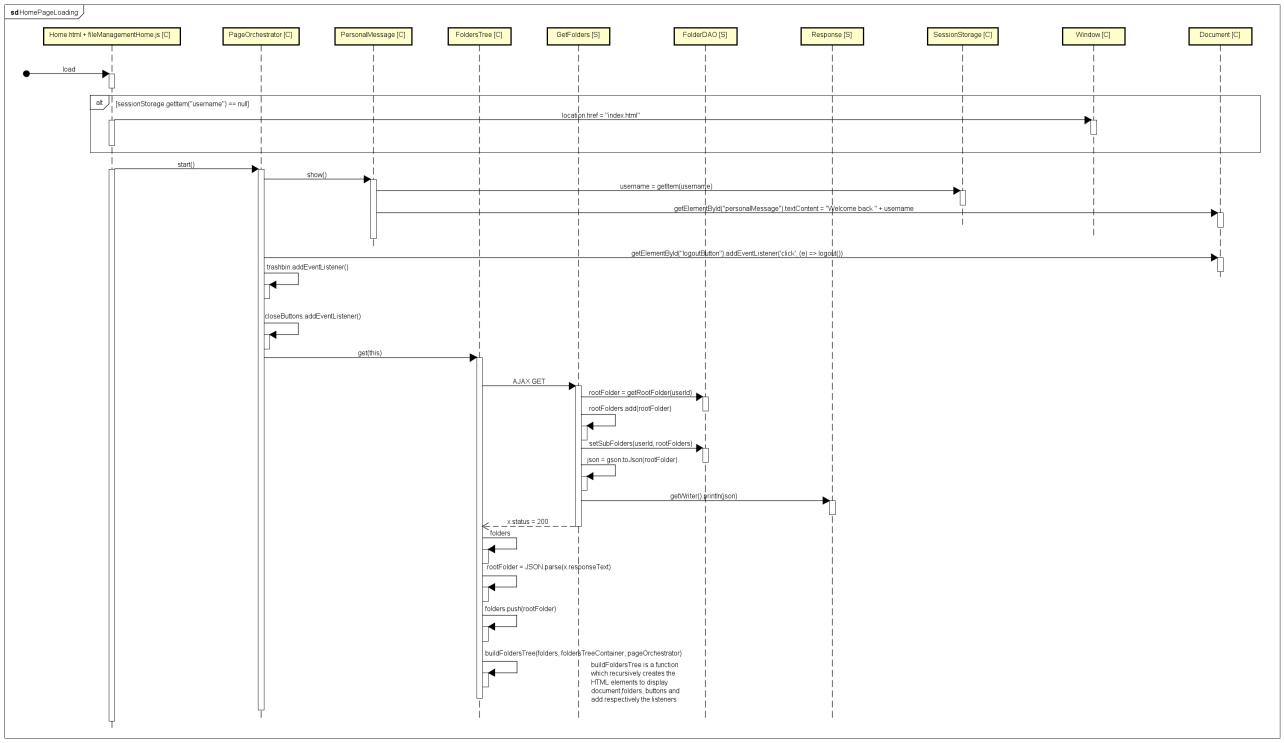


Figure 20: Home page loading sequence.

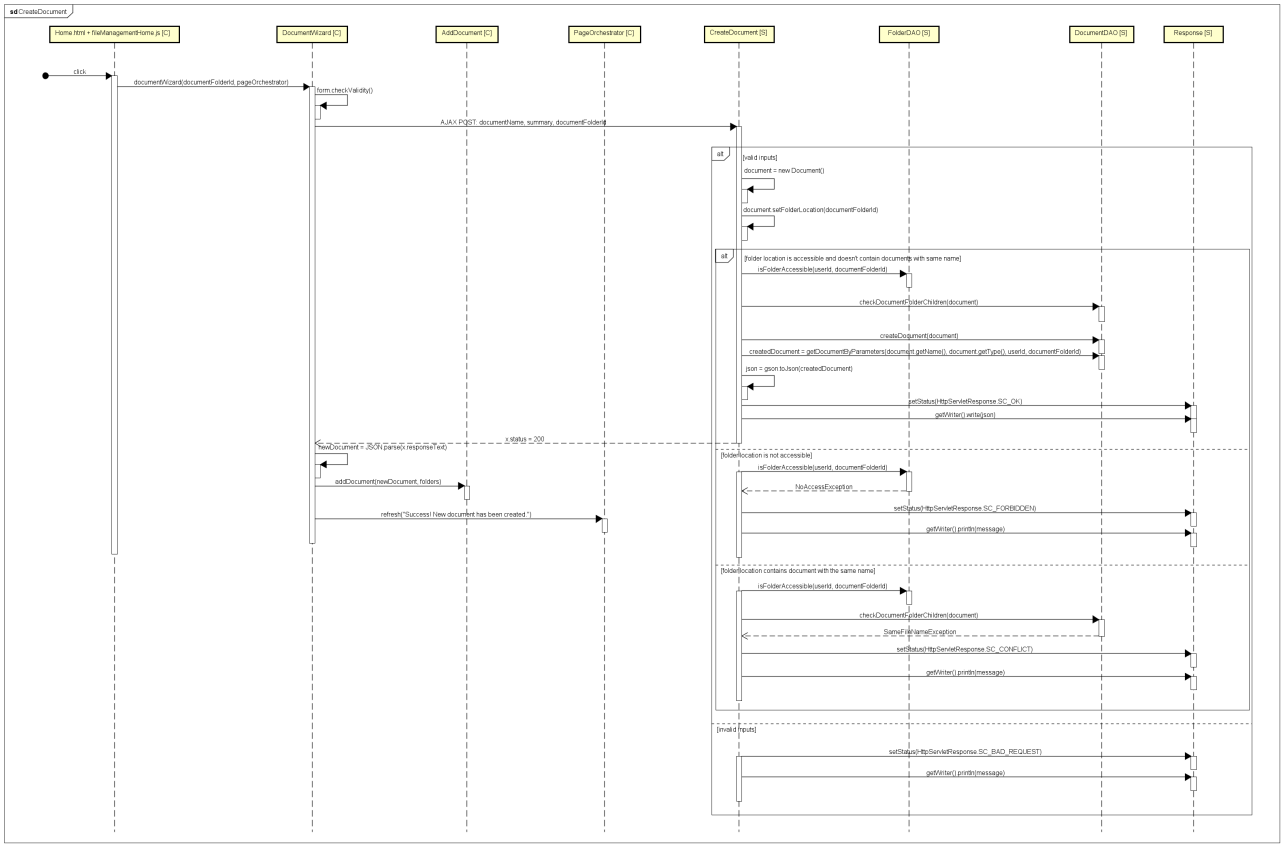


Figure 21: Creating a document sequence.

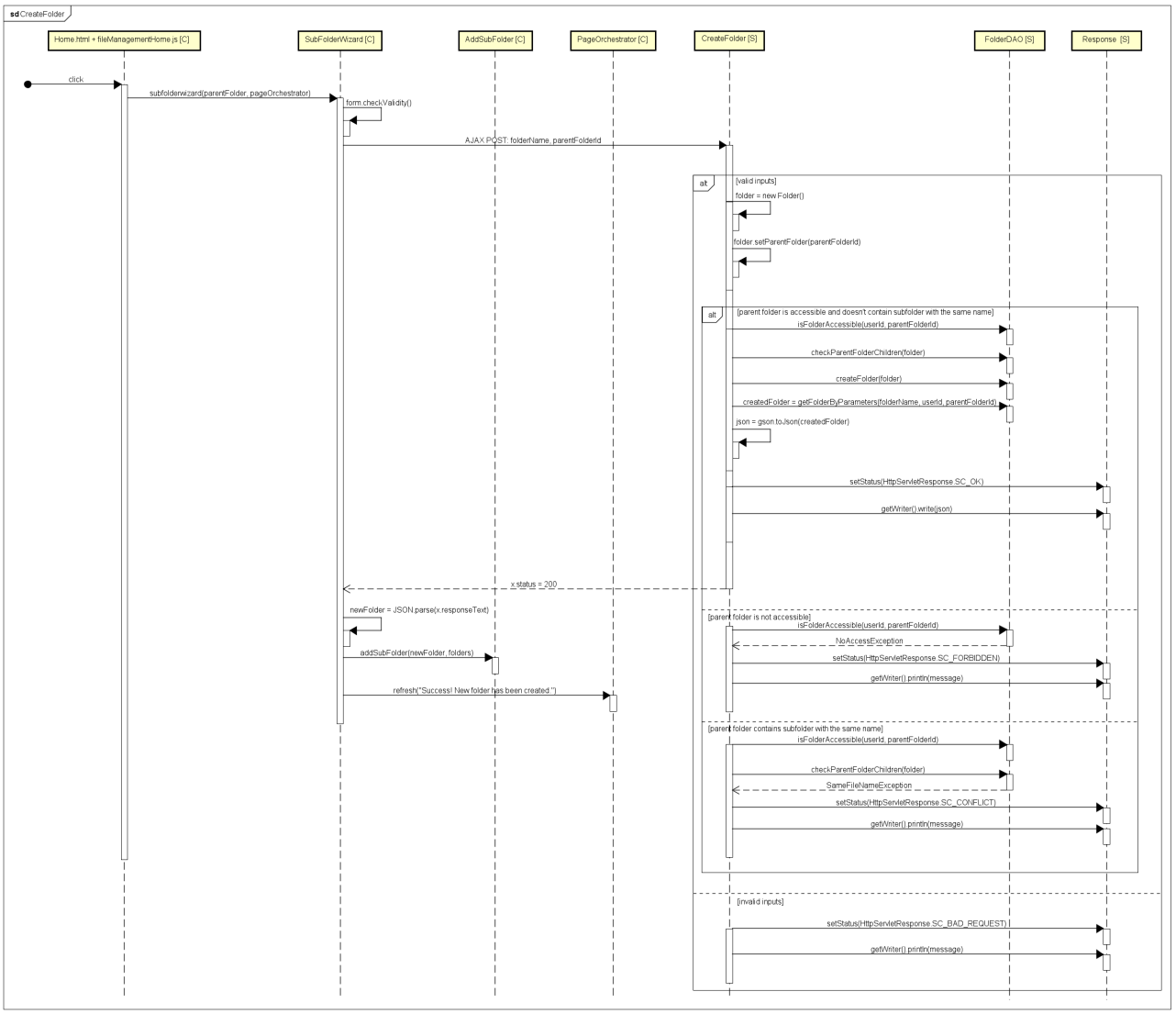


Figure 22: Creating a folder sequence.

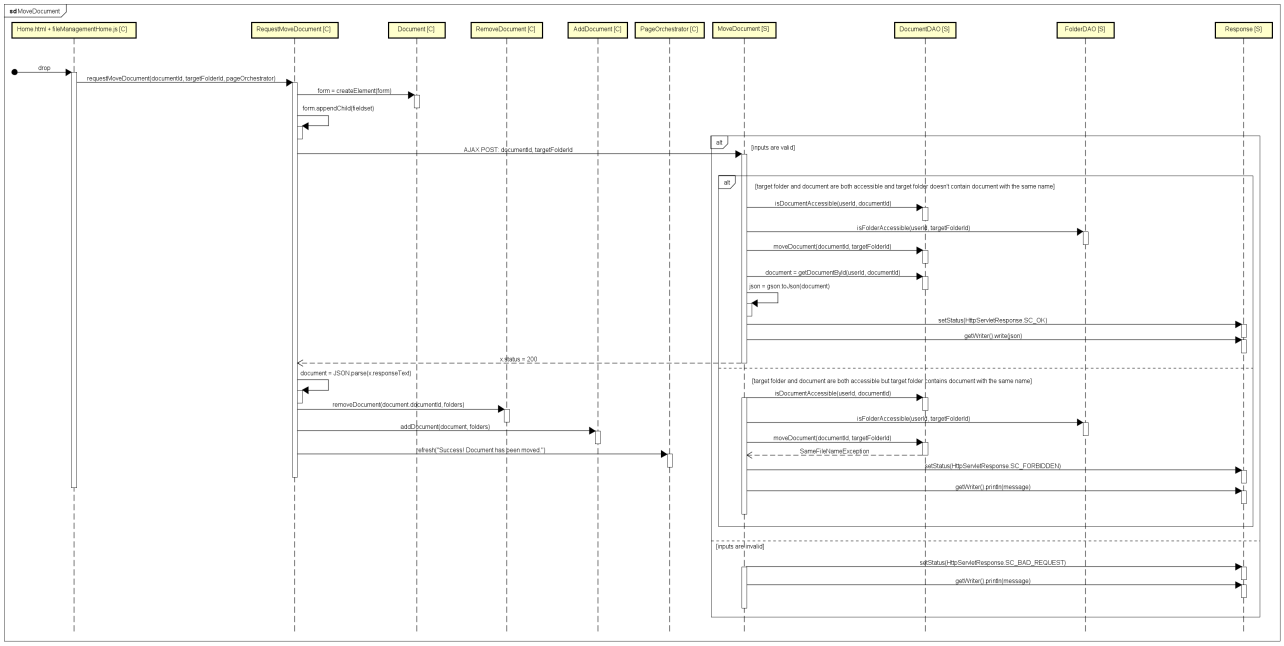


Figure 23: Moving a document sequence.

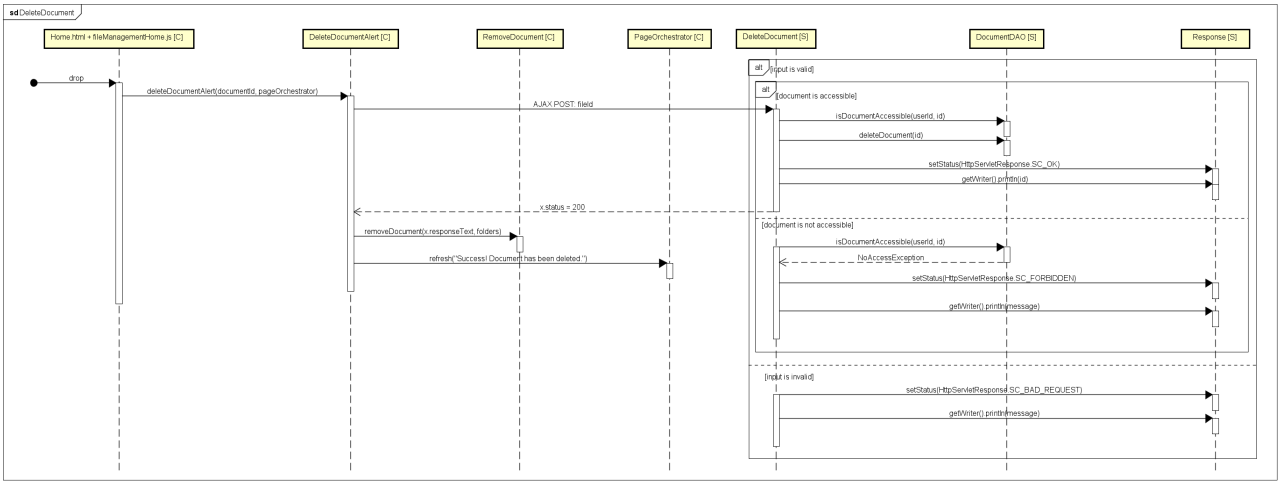


Figure 24: Deleting a document sequence.

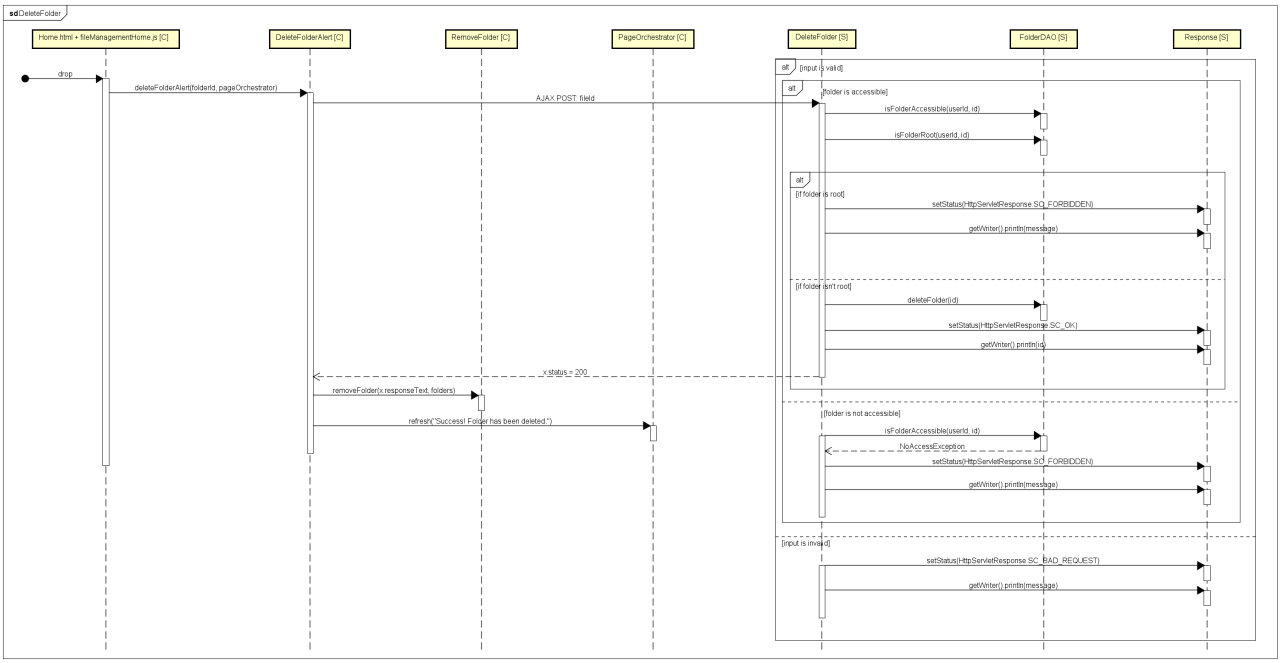


Figure 25: Deleting a folder sequence.

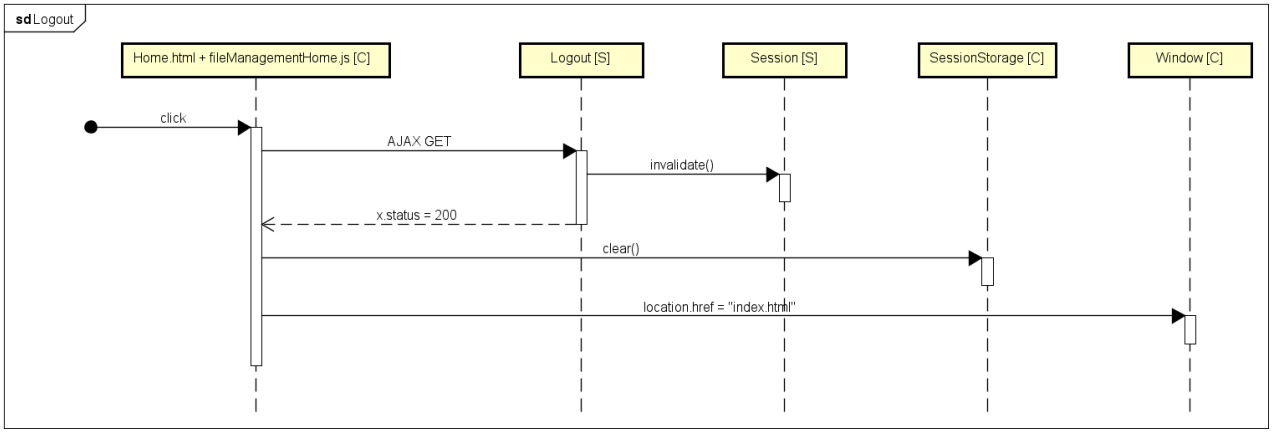


Figure 26: Logout sequence.

5 Final considerations

The application can be considered a success as it fully meets the project requirements of creating a file system website. The implementation leverages multiple technologies, including HTML, CSS, Java, JavaScript, Thymeleaf, and SQL, all of which were seamlessly integrated to develop a complete and functional application.
One of the greatest challenges I encountered during this project was understanding and utilizing JavaScript, a language I had never used before. Fortunately, the resources provided by the instructors, particularly on writing JavaScript and making AJAX requests, were invaluable in overcoming this hurdle. My workflow began with designing the database, which I iterated on two to three times before finalizing the current structure. I am particularly proud of the database design, as I believe it is the most efficient and robust aspect of the project.
The application also has potential for future improvements, such as introducing an admin role with the ability to view all folders and documents across users or implementing the functionality to move folders, which was outside the scope of this project.
This project has been instrumental in solidifying my skills in Java and in designing complex web applications. The experience of integrating diverse technologies and managing data within an SQL environment proved to be both challenging and rewarding. Overcoming these challenges significantly enhanced my problem-solving abilities. The final product is a functional, scalable solution with ample room for further enhancements and features.