Module Interface Specification for UnderTree

Team 22, Capstoners
Palanichamy Veerash
Kannammalil Kevin
Qureshi Eesha
Ahmed Faiq

April 5, 2023

1 Revision History

Date	Version	Notes
Jan 13th	1.0	Created MIS Document
Jan 14th	1.1	Assigned Sections
Jan 17th	1.2	Completed Modules
Jan 18th 2022	1.3	Final Changes

2 Symbols, Abbreviations and Acronyms

See SRS Documentation at SRS

Contents

3 Introduction

The following document details the Module Interface Specifications for UnderTree.

4 Notation

The structure of the MIS for modules comes from Hoffman and Strooper (1995), with the addition that template modules have been adapted from Ghezzi et al. (2003). The mathematical notation comes from Chapter 3 of Hoffman and Strooper (1995). For instance, the symbol := is used for a multiple assignment statement and conditional rules follow the form $(c_1 \Rightarrow r_1 | c_2 \Rightarrow r_2 | ... | c_n \Rightarrow r_n)$.

The following table summarizes the primitive data types used by UnderTree.

Data Type	**Notation**	**Description**
character	char	a single symbol or digit
integer	$\mathbb{Z}$	a number without a fractional component in (-∞, ∞)
natural number	$\mathbb{N}$	a number without a fractional component in [1, ∞)
real	$\mathbb{R}$	any number in (-∞, ∞)
boolean	$\mathbb{B}$	a True or False value

The specification of UnderTree uses some derived data types: sequences, strings, and tuples. Sequences are lists filled with elements of the same data type. Strings are sequences of characters. Tuples contain a list of values, potentially of different types. In addition, UnderTree uses functions, which are defined by the data types of their inputs and outputs. Local functions are described by giving their type signature followed by their specification.

5 Module Decomposition

The following table is taken directly from the Module Guide document for this project.

Table 1: Module Hierarchy

Level 1	**Level 2**
Hardware-Hiding Module	
Behaviour-Hiding Module	Project Editing Module
	Editor Module
	Syntax Highlighting Module
	File List Module
	File Toolbar Module
	New File Module
	Upload File Module
	User Cursors Module
	Text Highlighting Module
	File Synchronization Module
	File Services Module
	File Database Interface Module
	PDF Module
	PDF Renderer Module
	PDF Compiler Module
	Chat Module
	Chat Services Module
	Chat Database Interface Module
	Chat Socket Module
	Instructions View Module
	Projects Module
	Project List Module
	Project Deletion Module
	Project Creation Module
	New Project Module
	Import Project Module
	Project Services Module
	Project Database Interface Module
	GitHub Module
	GitHub Services Module
	Authentication Module
	Auth Service Module
	Auth Database Interface Module
Software Decision Module	
	File Data Module
	Chat Data Module
	Project Data Module
	Auth Data Module
	MongoDB Module

6 MIS of PDF Module

6.1 Module

PDF

6.2 Uses

PDFRenderer, FileServices

6.3 Syntax

6.3.1 Exported Constants

N/A

6.3.2 Exported Access Programs

Name	**In**	**Out**	**Exceptions**
init	String, String		
compileLatex			
downloadPDF			

6.4 Semantics

6.4.1 State Variables

projectName: String
fileName: String

6.4.2 Environment Variables

pdfComponent: The browser component that will display the PDF file

errorRenderer: The browser component that will display any error text

fileDownloader: The component responsible for downloading a file in a browser

compileButton: Button that will trigger the compilation of the LaTeX file, specifically calling the **handleCompileClicked()** function when clicked

downloadButton: Button that will download the PDF unto to the user's PC, specifically calling the **downloadPDF()** function when clicked

6.4.3 Assumptions

Upon loading the editor page, the *pdfComponent* displays an empty PDF file.

6.4.4 Access Routine Semantics

init(project, file):

- transition: *projectName*, *fileName* := project, file

  Render *pdfComponent*. Also render *compileButton*, and *downloadButton* and attach onClickListerners on them.

compileLatex():

- transition: *pdfComponent*, *errorRenderer* :=
  PDFRenderer.render((FileServices.compilePDF(*projectName*, *fileName*).error ≡ ””) ⇒
  FileServices.compilePDF(*projectName*, *fileName*).fileData|
  true ⇒ NULL),

  (FileServices.compilePDF(*projectName*, *fileName*)).error ≡ ””) ⇒ NULL
  | true ⇒
  FileServices.compilePDF(*projectName*, bytes(*fileName*).error)

downloadPDF():

- transition: *fileDownloader* := add the file stored in *pdfComponent* to the download queue of the browser in *fileDownloader* regardless of whether the PDF file is empty or not.

6.4.5 Local Functions

N/A

7 MIS of File Services Module

7.1 Module

FileServices

7.2 Uses

AuthService, PDFCompiler, FileDatabaseInterface

7.3 Syntax

7.3.1 Exported Constants

N/A

7.3.2 Exported Access Programs

Name	**In**	**Out**	**Exceptions**
compilePDF	String, String	tuple of (fileData: Sequence of $\mathbb{R}$, error: String)	
renameFileByIndex	String, $\mathbb{N}$, String		
createNewFile	String, String		
uploadFile	String, String, String		

7.4 Semantics

7.4.1 State Variables

N/A

7.4.2 Environment Variables

JWT: JSON Web Token that is passed to the server from the user's client as a cookie

7.4.3 Assumptions

N/A

7.4.4 Access Routine Semantics

compilePDF(projectName, fileName):

- output: out := AuthService.authenticate(JWT, projectName) $\Rightarrow$((PDFCompiler.compile(FileDatabaseInterface.getFile(projectName, fileName)).error $\equiv$ NULL) $\Rightarrow \langle$ PDFCompiler.compile(FileDatabaseInterface.getFile(projectName, fileName)).data, ""$\rangle|$

true $\Rightarrow$
$\langle$"", PDFCompiler.compile(FileDatabaseInterface.getFile(projectName, fileName)).error$\rangle$)
|
AuthService.authenticate(JWT, projectName) $\equiv$ false $\Rightarrow$ $\langle$"", "failed to authenticate"$\rangle$

renameFileByIndex(projectName, currentFileIndex, newName):

- output: out := AuthService.authenticate(JWT, projectName) $\equiv$ true $\Rightarrow$ FileDatabaseInterface.renameFile(projectName, index, fileName)

createNewFile(projectName, fileName):

- output: out := AuthService.authenticate(JWT, projectName) $\equiv$ true $\Rightarrow$ FileDatabaseInterface.createNewFile(projectName, fileName)

uploadFile(projectName, fileName, fileData):

- output: out := AuthService.authenticate(JWT, projectName) $\equiv$ true $\Rightarrow$ FileDatabaseInterface.createNewFile(projectName, fileName) $\Rightarrow$ FileDatabaseInterface.writeToFile(projectName, filename, fileData)

7.4.5 Local Functions

N/A

8 MIS of File Data Module

8.1 Module

FileData

8.2 Uses

N/A

8.3 Syntax

8.3.1 Exported Data Types

File: tuple of (fileName: String, fileType: String, filePath: String, contributors: $\langle$String$\rangle$ of size ?, documentID: string)
FileData: tuple of (projectName: String, owner: String, files: $\langle$File$\rangle$ of size ?) of size 1
ConnectedUser: tuple of (userName: String, profilePictureUrl: String) of size ?

8.3.2 Exported Access Programs

N/A

8.4 Semantics

8.4.1 State Variables

N/A

8.4.2 Environment Variables

N/A

8.4.3 Assumptions

N/A

8.4.4 Access Routine Semantics

N/A

8.4.5 Local Functions

N/A

9 MIS of Chat Module

9.1 Module

Chat

9.2 Uses

ChatServices, ChatSocket

9.3 Syntax

9.3.1 Exported Constants

N/A

9.3.2 Exported Access Programs

Name	**In**	**Out**	**Exceptions**
init	String		
updateChatMessage	String		
sendMessage			
processSocketEvents			

9.4 Semantics

9.4.1 State Variables

projectName: String
connectedUsers: Sequence of (tuple of (userName: String, profilePictureUrl: String))
messages: Sequence of (tuple of (userName: String, message: String))
chatMessage: String

9.4.2 Environment Variables

chatComponent: The overall state of the chat interface.

messageInputField: This input field is responsible for updating the chat message the user is sending, or specifically calls the **updateChatMessage(messageInputField.textValue)** function when the input value is updated.

sendChatButton: This button sends a new chat message from the user, or specifically calls the **sendChatMessage()** function when clicked.

9.4.3 Assumptions

The init() function is automatically called by the rendering library when we first render the module by using the project name passed to it from the url.

9.4.4 Access Routine Semantics

init(project):

- transition: *projectName, connectedUsers, messages* := project, ChatServices.getConnectedUsers(project), ChatServices.getChatMessages(project)
- transition: *chatComponent* := //Described by the following operational spec

  ChatSocket.connect(ChatServices.SERVER_URL||project)

  Render messages, *connectedUsers.userName*, and *connectedUsers.profilePictureUrl*. Also render the *sendChatButton* and *messageInputField* on the user interface. Lastly, attach a onClickListener to the *sendChatButton* and a keyPressListener to *messageInputField*.

updateChatMessage(newMessage):

- transition: *chatMessage* := newMessage

sendMessage():

- transition: *chatComponent* := //Described by the following operational spec

  ChatSocket.emit("newMessage", *chatMessage*)

processSocketEvents():

Data is the JSON fields passed into the socket event into each event and then passed into each conditional. These events can be emitted from other clients or the Chat Services Module.

- transition: *connectUsers* :=
  ChatSocket.on("newUser", data) $\Rightarrow$ *connectedUsers*$||\langle\langle$data.userName, data.profilePictureUrl$\rangle\rangle$ $|$ ChatSocket.on("userRemoved", data) $\Rightarrow$ $\langle$user : tuple of (userName: String, profilePictureUrl: String)$|$user $\in$ *connectedUsers* $\wedge$ user.userName $\neq$ data.userName : $\langle$user.userName, user.profilePictureUrl$\rangle\rangle$
- transition: *messages* :=
  ChatSocket.on("newMessage", data) $\Rightarrow$ *messages*$||\langle\langle$data.userName, data.message$\rangle\rangle$

9.4.5 Local Functions

N/A

10 MIS of Chat Data Module

10.1 Module

ChatData

10.2 Uses

N/A

10.3 Syntax

10.3.1 Exported Data Types

ChatMessage: tuple of (userName: String, message: String) of size ?
ConnectedUser: tuple of (userName: String, profilePictureUrl: String) of size ?

10.3.2 Exported Constants

N/A

10.3.3 Exported Access Programs

N/A

10.4 Semantics

10.4.1 State Variables

N/A

10.4.2 Environment Variables

N/A

10.4.3 Assumptions

N/A

10.4.4 Access Routine Semantics

N/A

10.4.5 Local Functions

N/A

11 MIS of Chat Services Module

11.1 Module

ChatServices

11.2 Uses

ChatData, ChatDatabaseInterface, ChatSocket, AuthService

11.3 Syntax

11.3.1 Exported Constants

SERVER_URL: The url of the main chat socket on the server.

11.3.2 Exported Access Programs

Name	**In**	**Out**	**Exceptions**
getConnectedUsers	String	Sequence of ChatData.ConnectedUser	
getChatMessages	String	Sequence of ChatData.ChatMessage	
processSocketEvents			

11.4 Semantics

11.4.1 State Variables

N/A

11.4.2 Environment Variables

httpServer: The REST API server setup in the backend that processes all requests on "/connectedUsers" to **getConnectedUsers(project)** and "/chatMessages" to **getChatMessages(project)**.

JWT: JSON Web Token that is passed to the server from the user's client as a cookie

11.4.3 Assumptions

httpServer is initialized before any of the functions are called in this module.

11.4.4 Access Routine Semantics

getConnectedUsers(project):

- output: out := AuthService.authenticate(JWT, project) $\equiv$ true $\Rightarrow$ ChatDatabaseInterface.getConnectedUsers(project) | AuthService.authenticate(JWT, project) $\equiv$ false $\Rightarrow$ $\langle\rangle$

getChatMessages(project):

- output: out := AuthService.authenticate(JWT, project) $\equiv$ true $\Rightarrow$ ChatDatabaseInterface.getChatMessages(project) | AuthService.authenticate(JWT, project) $\equiv$ false $\Rightarrow$ $\langle\rangle$

processSocketEvents():

Data is the JSON fields passed into the socket event into each event and then passed into each conditional. These events can be emitted from other clients or the Chat Services Module.

- transition: //Described by the following operational spec

  (ChatSocket.on("connected", data) $\wedge$ AuthService.authenticate(JWT, project) $\equiv$ true)
  $\Rightarrow$ (ChatDatabaseInterface.addUser(data.userName, data.profilePictureUrl, data.project)
  $\wedge$ ChatSocket.emit("newUser", data)) |
  ChatSocket.on("connected", data) $\wedge$ AuthService.authenticate(JWT, project) $\equiv$ false)
  $\Rightarrow$ (ChatSocket.disconnect(data.userName))

  (ChatSocket.on("disconnect", data) $\Rightarrow$ (ChatDatabaseInterface.removeUser(data.userName, data.project) $\wedge$ ChatSocket.emit("userRemoved", data))

  (ChatSocket.on("newMessage", data) $\wedge$ AuthService.authenticate(JWT, project) $\equiv$ true) $\Rightarrow$ (ChatDatabaseInterface.addMessage(data.userName, data.message, data.project)
  |
  (ChatSocket.on("newMessage", data) $\wedge$ AuthService.authenticate(JWT, project) $\equiv$ false) $\Rightarrow$ (ChatSocket.disconnect(data.userName))

11.4.5 Local Functions

N/A

12 MIS of Chat Database Interface Module

12.1 Module

ChatDatabaseInterface

12.2 Uses

ChatData, MongoDB

12.3 Syntax

12.3.1 Exported Constants

N/A

12.3.2 Exported Access Programs

Name	In	Out	Exceptions
getConnectedUsers	String	Sequence of ChatData.ConnectedUser	
getChatMessages	String	Sequence of ChatData.ChatMessage	
addUser	String, String, String		
removeUser	String, String		
addMessage	String, String, String		

12.4 Semantics

12.4.1 State Variables

12.4.2 Environment Variables

N/A

12.4.3 Assumptions

N/A

12.4.4 Access Routine Semantics

getConnectedUsers(projectName):

- output: out := Return the list of users associated with projectName in the chatUsers documents from MongoDB

getChatMessages(projectName):

- output: out := Return all the chat messages in ascending order of time added with projectName in the chat documents from MongoDB

addUser(userName, profilePictureUrl, projectName):

- output: out := Add a new user to chatUsers document with projectName in MongoDB with the following data ChatData.ConnectedUser(userName, profilePicture)

removeUser(userName, projectName):

- output: out := Add a new user to chatUsers document with projectName and ChatData.ConnectedUser.userName *equiv* userName in MongoDB

addMessage(userName, message, projectName):

- output: out := Add a new message to chat document with projectName in MongoDB with the following data ChatData.ChatMessage(userName, message)

12.4.5 Local Functions

N/A

13 MIS of Instructions View Module

13.1 Module

InstructionsView

13.2 Uses

N/A

13.3 Syntax

13.3.1 Exported Constants

N/A

13.3.2 Exported Access Programs

Name	**In**	**Out**	**Exceptions**
init			
openInstructions			
closeInstructions			

13.4 Semantics

13.4.1 State Variables

$isOpen$: $\mathbb{B}$

13.4.2 Environment Variables

$instructionsModal$: The popup UI component for displaying the instructions

$openButton$: Button that will trigger the **openInstructions()** function when clicked

$closeButton$: Button that will trigger the **closeInstructions()** function when clicked

13.4.3 Assumptions

The init function is ran on page load

13.4.4 Access Routine Semantics

init():

- transition: *isOpen* := false

  Render *openButton* and *closeButton*, and attach onClickListerners on them.

openInstructions():

- transition: instructionsModal := *isOpen* $\equiv$ false $\Rightarrow$ instructionModal.render()

closeInstructions():

- transition: instructionsModal := *isOpen* $\equiv$ true $\Rightarrow$ instructionModal.unRender()

13.4.5 Local Functions

N/A

14 MIS of File Database Interface Module

14.1 Module

FileDatabaseInterface

14.2 Uses

MongoDB

14.3 Syntax

14.3.1 Exported Constants

N/A

14.3.2 Exported Access Programs

Name	**In**	**Out**	**Exceptions**
getFile	String, String	String	RecordDoesNotExist
renameFileByIndex	String, $\mathbb{N}$	String	RecordDoesNotExist
createNewFile	String, String	String	
writeToFile	String, String, String	String	RecordDoesNotExist

14.4 Semantics

14.4.1 State Variables

14.4.2 Environment Variables

MongoDB: MongoDB is a database where the projects and files will be stored which can be represented mathematically as a set of projects which has a list of files

14.4.3 Assumptions

N/A

14.4.4 Access Routine Semantics

getFile(projectName, fileName):

- output: out := MongoDB.GetOne(projectName, fileName)
- exception: exc := Throw a RecordDoesNotExist exception if no such record exists in MongoDB

renameFileByIndex(projectName, index, newName):

- transition: ($\exists p | p \in$ MongoDB.$getFiles()$: p.projectName $\equiv$ projectName) $\Rightarrow$ (p.files[index].fileName := newName)
- exception: exc := Throw a RecordDoesNotExist exception if no such record exists in MongoDB

createNewFile(projectName, fileName):

- transition: MongoDB.InsertOne(projectName, fileName)
- exception: N/A

writeToFile(projectName, fileName, fileData):

- transition: $\exists p | p \in$ MongoDB $\wedge$ p.projectName $\equiv$ projectName : ($\exists f | f \in p.files \wedge$ f.fileName $\equiv$ fileName : f.content := fileData)
- exception: exc := Throw a RecordDoesNotExist exception if no such record exists in MongoDB

14.4.5 Local Functions

N/A

15 MIS of Project Editing Module

15.1 Module

ProjectEditor

15.2 Uses

ProjectDetails, PDF, FileList, Editor, Chat

15.3 Syntax

15.3.1 Exported Constants

15.3.2 Exported Access Programs

Name	In	Out	Exceptions
ProjectEditor			-

15.4 Semantics

15.4.1 State Variables

15.4.2 Environment Variables

editor: editor is the area where the current file to be edited will displayed

fileList: fileList is the area where the list of file in the current project will be displayed

projectDetails: projectDetails is the area where the details such as name and collaborators of the file will be displayed

pdf: pdf is the area where the compiled pdf of the current LaTeX file will be shown *chat*: chat is the area where chat between collaborators will be shown

15.4.3 Assumptions

N/A

15.4.4 Access Routine Semantics

ProjectEditor():

- transition: renders *editor*, *fileList*, *projectDetails*, *pdf*, *chat*
- exception: N/A

15.4.5 Local Functions

N/A

16 MIS of File List Module

16.1 Module

FileList

16.2 Uses

FileToolbar, FileServices, ProjectServices

16.3 Syntax

16.3.1 Exported Constants

16.3.2 Exported Access Programs

Name	In	Out	Exceptions
FileList	String		
openFilePressed	$\mathbb{N}$		

16.4 Semantics

16.4.1 State Variables

projectName: String

16.4.2 Environment Variables

fileListArea: is the GUI component that contains the *fileToolbar* and *listArea*

fileToolbar: fileToolbar is a toolbar to do quick actions on file which is implemented by FileToolbar Module

listArea: listArea is a list which renders multiple GUI components into a list

fileButton: fileButton is a gui component that will trigger openFilePressed()

16.4.3 Assumptions

N/A

16.4.4 Access Routine Semantics

FileList(project):

- transition:
  projectName := project

  $\forall i | 0 \leq i \leq$ SIZE : add(ProjectServices.getFilesName(projectName, fileName)[i], i)
  where SIZE is the size of the list returned by ProjectServices.getFilesName(projectName, fileName)

  render *fileToolbar* and *listArea* in fileList Area

- exception: N/A

16.4.5 Local Functions

add(fileName):

- transition: add a new *fileButton* with the name *filename* to *listArea* where when *file-Button* is pressed, it will call openFilePressed(fileName)

17 MIS of File Toolbar Module

17.1 Module

FileToolbar

17.2 Uses

NewFile, UploadFile, DeleteFile, FileServices

17.3 Syntax

17.3.1 Exported Constants

17.3.2 Exported Access Programs

Name	In	Out	Exceptions
FileToolbar	string, $\mathbb{N}$		
newFilePressed			
uploadFilePressed			
deleteFilePressed			
renameFilePressed			RenameFailed

17.4 Semantics

17.4.1 State Variables

projectName: String
currentFileIndex: $\mathbb{N}$

17.4.2 Environment Variables

newFileButton: is a button that will trigger newFilePressed() when it is pressed

uploadFileButton: is a button that will trigger uploadFilePressed() when it is pressed

deleteFileButton: is a button that will trigger deleteFilePressed() when it is pressed

renameFileButton: is a button that will trigger renameFilePressed() when it is pressed

newFileModal: is a modal implemented by NewFile module

uploadFileModal: is a modal implemented by UploadFile module

renameFileInputField: it is an input field used to type a new name for the file

17.4.3 Assumptions

N/A

17.4.4 Access Routine Semantics

FileToolbar(project, index):

- transition:
  projectName := project
  currentFileIndex := index
  render *newFileButton* and *uploadFileButton*
- exception: N/A

newFilePressed():

- transition: render *newFileModal* and make it visible
- exception: N/A

uploadFilePressed():

- transition: render *uploadFileModal* and make it visible
- exception: N/A

deleteFilePressed():

- transition: renders *deleteFileModal* modal onto the screen using the current file index as a parameter: DeleteFile(currentFileIndex)
- exception: N/A

renameFilePressed():

- transition: renders *renameFileInputField* on top of the button of the current file selected in the list of files. Once the new name is typed and *Enter* key is pressed, the following steps will happen:

  FileServices.renameFileInProjectByIndex(projectName, currentFileIndex, newName) where *newName* is the name typed in *renameFileInputField*
- exception: RenameFailed

17.4.5 Local Functions

N/A

18 MIS of New File Module

# This module is a GUI component

18.1 Module

NewFile

18.2 Uses

FileServices

18.3 Syntax

18.3.1 Exported Constants

18.3.2 Exported Access Programs

Name	In	Out	Exceptions
NewFile	String		-
confirmButtonPressed			FileNotCreated
cancelButtonPressed			-

18.4 Semantics

18.4.1 State Variables

projectName: String

18.4.2 Environment Variables

fileNameInputField: input field where the name of the file will be inputted

confirmButton: button that will trigger confirmButtonPressed when it is pressed

cancelButton: button that will trigger cancelButtonPressed when it is pressed

18.4.3 Assumptions

N/A

18.4.4 Access Routine Semantics

NewFile(project):

- transition:
  projectName := project
  renders *fileNameInputField*, *confirmButton*, *cancelButton*
- exception: N/A

confirmButtonPressed():

- transition: FileServices.createNewFile(projectName)
- exception: FileNotCreated

cancelButtonPressed():

- transition: closes this modal
- exception: N/A

18.4.5 Local Functions

N/A

19 MIS of Upload File Module

# This module is a GUI component

19.1 Module

UploadFile

19.2 Uses

FileServices

19.3 Syntax

19.3.1 Exported Constants

19.3.2 Exported Access Programs

Name	In	Out	Exceptions
UploadFile	String		-
confirmButtonPressed			FileNotCreated
cancelButtonPressed			-
onChange			-

19.4 Semantics

19.4.1 State Variables

projectName: String
file: File # File is web representation of a file

19.4.2 Environment Variables

fileInput: file uploader GUI component that opens the window that lets you choose your local file and will trigger onChange(event) once file is selected

confirmButton: button that will trigger confirmButtonPressed when it is pressed

cancelButton: button that will trigger cancelButtonPressed when it is pressed

19.4.3 Assumptions

N/A

19.4.4 Access Routine Semantics

UploadFile(project):

- transition:
  projectName := project
  renders *fileInput*, *confirmButton*, *cancelButton*
- exception: N/A

confirmButtonPressed():

- transition: FileServices.uploadFile(projectName, file)
- exception: FileNotCreated

cancelButtonPressed():

- transition: closes this modal
- exception: N/A

onChange(event):

- transition: file := event.target.files[0]
- exception: N/A

19.4.5 Local Functions

N/A

20 MIS of Editor File Module

# This module is a GUI component

20.1 Module

Editor

20.2 Uses

UserCursor, TextHighlighting, SyntaxHighlighting, FileSynchronization

20.3 Syntax

20.3.1 Exported Constants

20.3.2 Exported Access Programs

Name	**In**	**Out**	**Exceptions**
init	string, setCurrentText: (string $\rightarrow$ void)		-

20.4 Semantics

20.4.1 State Variables

modified: $\mathbb{B}$
documentID: string

20.4.2 Environment Variables

codeMirror: codeMirror is the editor component that is implemented by CodeMirror library
localStorage: localStorage is storage used by the browser which Undertree will use to store data such as username
websocketProvider: websocketProvider is a web socket used by the YJS library to synchronize file content between the collaborators

20.4.3 Assumptions

Editor.init() is called during the initial render

20.4.4 Access Routine Semantics

init(currentFilePath, setCurrentText):

- transition: *webSocketProvider* is assigned a new instance of a socket created using *currentFilePath*. Registers UserCursor, SyntaxHighlighting modules with *codeMirror*. Binds *codeMirror* to *webSocketProvider* so that the editor is synchronized using YJS's websocket. *codeMirror* also gets binded to event listener that will call *setCurrentText*(t) and *addUserToModified*$()$ whenever the editor is updated, where t is the text in the editor. Lastly, documentID := currentFilePath
- exception: N/A

20.4.5 Local Functions

addUserToModified():

- transition: $modified = true \Rightarrow modified := true$ and make a API call to the server to add *localStorage.username* to the list of the contributor for the file with file path equals to *documentID*.

21 MIS of Projects Module

21.1 Module

Projects

21.2 Uses

ProjectList, ProjectCreation

21.3 Syntax

21.3.1 Exported Constants

21.3.2 Exported Access Programs

Name	In	Out	Exceptions
createProjectButtonPressed			-

21.4 Semantics

21.4.1 State Variables

21.4.2 Environment Variables

createProjectButton: a button that leads to the project creation screen implemented in the projectCreation module, triggers the createProjectButtonPressed() function

projectList: a GUI component implemented in the projectList module

21.4.3 Assumptions

N/A

21.4.4 Access Routine Semantics

createProjectButtonPressed():

- transition: triggers ProjectCreation.ProjectCreation()
- exception: N/A

21.4.5 Local Functions

N/A

22 MIS of Project List Module

22.1 Module

ProjectList

22.2 Uses

ProjectServices, ProjectDeletion

22.3 Syntax

22.3.1 Exported Constants

22.3.2 Exported Access Programs

Name	In	Out	Exceptions
ProjectList			-
openButtonPressed			-
deleteButtonPressed			-

22.4 Semantics

22.4.1 State Variables

selectedProject: String

22.4.2 Environment Variables

projectList: projectList is the area where the list of projects is displayed

projectLabel: project is a block in the projectList for an individual project being displayed

openButton: openButton is a button next to a projectLabel, clicking it triggers openButtonPressed()

deleteButton: deleteButton is a button next to a projectLabel, clicking it triggers deleteButtonPressed()

22.4.3 Assumptions

N/A

22.4.4 Access Routine Semantics

openButtonPressed():

- transition: triggers ProjectEditing.ProjectEditor()
- exception: N/A

deleteButtonPressed():

- transition: triggers ProjectDeletion.ProjectDeletion()
- exception: N/A

ProjectList():

- transition: renders ProjectList module
- exception: N/A

22.4.5 Local Functions

N/A

23 MIS of Project Deletion Module

23.1 Module

ProjectDeletion

23.2 Uses

ProjectServices

23.3 Syntax

23.3.1 Exported Constants

23.3.2 Exported Access Programs

Name	In	Out	Exceptions
ProjectDeletion			-
confirmButtonPressed			-
cancelButtonPressed			-

23.4 Semantics

23.4.1 State Variables

projectName: String
ownerName: String

23.4.2 Environment Variables

confirmButton: confirmButton is the button that will appear in the modal to confirm delete action. It will trigger the confirmButtonPressed() function.

confirmActionMessage: confirmActionMessage is a text message that will ask the user if they are sure they want to delete the selected project

cancelButton: cancelButton is the button that will appear in the modal to abort deletion and return, it will trigger the cancelButtonPressed() function

successMessage: a message showing that the deletion was successful

23.4.3 Assumptions

N/A

23.4.4 Access Routine Semantics

ProjectDeletion():

- transition: renders ProjectDeletion module
- exception: N/A

confirmButtonPressed():

- transition: triggers ProjectServices.deleteProject(projectName, ownerName), renders *successMessage*, closes *confirmActionMessage*
- exception: N/A

cancelButtonPressed():

- transition: closes *confirmActionMessage*
- exception: N/A

23.4.5 Local Functions

N/A

24 MIS of Project Creation Module

24.1 Module

ProjectCreation

24.2 Uses

NewProject, ImportProject

24.3 Syntax

24.3.1 Exported Constants

24.3.2 Exported Access Programs

Name	In	Out	Exceptions
ProjectCreation			-
newButtonPressed			-
importButtonPressed			-

24.4 Semantics

24.4.1 State Variables

24.4.2 Environment Variables

createNewButton: button that will allow user to create a project from scratch, triggers newButtonPressed()

createFromImportButton: button that will allow user to import a project, triggers importButtonPressed()

24.4.3 Assumptions

N/A

24.4.4 Access Routine Semantics

newButtonPressed():

- transition: triggers NewProject.NewProject()
- exception: N/A

importButtonPressed():

- transition: triggers ImportProject.ImportProject()
- exception: N/A

ProjectCreation():

- transition: renders ProjectCreation module
- exception: N/A

24.4.5 Local Functions

N/A

25 MIS of New Project Module

25.1 Module

NewProject

25.2 Uses

ProjectServices

25.3 Syntax

25.3.1 Exported Constants

25.3.2 Exported Access Programs

Name	In	Out	Exceptions
NewProject			-
createButtonPressed			InvalidInput

25.4 Semantics

25.4.1 State Variables

projectName: String

ownerName: String

collaborators: Set of Strings

creationDate: String

25.4.2 Environment Variables

projectForm: Form area on page that contains input fields *projectNameField*: Text input field where user will enter the desired project name

collaboratorsField: Text input field where user will list the desired collaborators

creationDateTag: Text feild with auto-populated date

createButton: Button that will submit the project form content, triggers createButton-Pressed()

25.4.3 Assumptions

N/A

25.4.4 Access Routine Semantics

NewProject():

- transition: renders NewProject module
- exception: N/A

createButtonPressed():

- transition: triggers ProjectServices.addProject(projectName, projectOwner, creationDate, collaborators, []), closes NewProject module
- exception: exc := Throw InvalidInputError if any of the input fields contain forbidden or null characters

25.4.5 Local Functions

N/A

26 MIS of Import Project Module

26.1 Module

ImportProject

26.2 Uses

ProjectServices

26.3 Syntax

26.3.1 Exported Constants

26.3.2 Exported Access Programs

Name	In	Out	Exceptions
ImportProject			-
createButtonPressed			InvalidInput
selectProjectButtonPressed			-

26.4 Semantics

26.4.1 State Variables

projectName: String

ownerName: String

collaborators: Set of Strings

creationDate: String

26.4.2 Environment Variables

projectList: Area on page that displays a list of possible projects to import from *selectProjectButton*: Button that triggers selectProjectButtonPressed() *projectDetails*: Form area on page that contains input fields *projectNameField*, *collaboratorsField*, *creationDateTag*, and *createButton*

projectNameField: Text input field where user will enter the desired project name

collaboratorsField: Text input field where user will list the desired collaborators *creationDateTag*: Text feild with auto-populated date

createButton: Button that will submit the project form content, triggers createButton-Pressed()

26.4.3 Assumptions

N/A

26.4.4 Access Routine Semantics

ImportProject():

- transition: renders ImportProject module
- exception: N/A

selectProjectButtonPressed():

- transition: triggers ProjectServices.getProject(projectName, ownerName), renders *projectDetails*
- transition: projectName := ProjectServices.getProject().projectName
- transition: collaborators := ProjectServices.getProject().collaborators
- transition: creationDate := ProjectServices.getProject().date
- exception: N/A

createButtonPressed():

- transition: triggers ProjectServices.addProject(projectName, projectOwner, creationDate, collaborators, []), closes ImportProject module
- exception: exc := Throw InvalidInputError if any of the input fields contain forbidden or null characters

26.4.5 Local Functions

N/A

27 MIS of Project Database Interface Module

27.1 Module

ProjectDatabaseInterface

27.2 Uses

ProjectData

27.3 Syntax

27.3.1 Exported Constants

27.3.2 Exported Access Programs

Name	In	Out	Exceptions
getProject	String, String, String	Sequence of Strings	RecordDoesNotExist
addProject	String, String, String[], String[]		InvalidInput
deleteProject	String, String		RecordDoesNotExist
editProjectDetail	String, String, String, String		InvalidInput, RecordDoesNotExist

27.4 Semantics

27.4.1 State Variables

27.4.2 Environment Variables

projectDirectory: The storage on the server where project details are stored

27.4.3 Assumptions

N/A

27.4.4 Access Routine Semantics

getProject(projectName, projectOwner):

- output: out := Return the project associated with the project name and owner name from MongoDB if it exists
- exception: exc := Throw a RecordDoesNotExist exception if no such record exists in MongoDB

addProject(projectName, projectOwner, date, collaborators[], files[]):

- transition: Insert a record for a project into MongoDB with the given name, owner, date, collaborators, and files
- exception: exc := Throw a InvalidInput exception if any of the supplied parameters contain forbidden characters or are null

deleteProject(projectName, projectOwner):

- transition: Remove the record for the project associated with the project name and owner name from MongoDB if it exists
- exception: exc := Throw a RecordDoesNotExist exception if no such record exists in MongoDB

editProjectDetail(projectName, owner, key, newValue):

- transition: Update the given key with the given newValue for a record with the given projectName and owner
- exception: exc := Throw a InvalidInput exception if any of the supplied parameters contain forbidden characters or are null
- exception: exc := Throw a RecordDoesNotExist exception if no such record exists in MongoDB

27.4.5 Local Functions

N/A

28 MIS of Project Services Module

28.1 Module

ProjectServices

28.2 Uses

ProjectDatabaseInterface, ProjectData, AuthService

28.3 Syntax

28.3.1 Exported Constants

28.3.2 Exported Access Programs

Name	In	Out	Exceptions
deleteProject	String, String		-
addProject	String, String, String, String[], String[]		-
getProject	String, String	Sequence of Strings	-
editProjectDetail	String, String, String, String		-

28.4 Semantics

28.4.1 State Variables

28.4.2 Environment Variables

JWT: JSON Web Token that is passed to the server from the user's client as a cookie

28.4.3 Assumptions

AuthService.authenticate(JWT, project) will be called and all functions will only run if AuthService.authenticate(jwt, project) returns true.

28.4.4 Access Routine Semantics

getProject(projectName, projectOwner):

- output: out := Return ProjectDatabaseInterface.deleteProject(projectName, projectOwner)
- exception: N/A

addProject(projectName, projectOwner, date, collaborators[], files[]):

- transition: trigger ProjectDatabaseInterface.addProject(projectName, projectOwner, date, collaborators[], files[])

- exception: N/A

deleteProject(projectName, projectOwner):

- transition: returns ProjectDatabaseInterface.deleteProject(projectName, projectOwner)
- exception: N/A

editProjectDetail(projectName, owner, key, newValue):

- transition : returns ProjectDatabaseInterface.editProjectDetail(projectName, owner, key, newValue)
- exception: N/A

28.4.5 Local Functions

N/A

29 MIS of GitHub Module

29.1 Module

GitHub

29.2 Uses

GitHubServices

29.3 Syntax

29.3.1 Exported Constants

N/A

29.3.2 Exported Access Programs

Name	In	Out	Exceptions
viewLog			
commitChanges	Map of String		
pushChanges			

29.4 Semantics

29.4.1 State Variables

logReqData: Map of String
logData: Seq of String
changesSelected: Map of String

29.4.2 Environment Variables

viewLogButton: is a button that will trigger viewLog() when it is pressed

selectLines: is a button that allows user to highlight blocks of text for changes they want to commit which is then stored in changesSelected

commitChangesButton: is a button that will trigger commitChanges() when it is pressed

pushChangesButton: is a button that will trigger pushChanges() when it is pressed

29.4.3 Assumptions

You can only click the *pushChangesButton* if you've committed previously. The UnderTree user data is cached and can be retrieved from a browser cookie.

29.4.4 Access Routine Semantics

viewLog(): calls GitHubServices.retrieveLog(logReqData) and passes in the user that clicked it along with the necessary information in logReqData. The data is then retrieved from the backend and updates the log view.

commitChanges(): calls GitHubServices.createCommit(data) and passes in *changesSelected* along with other the necessary information in data.

pushChanges(): calls GitHubServices.pushCommit(data) and passes in the user that clicked it along with the necessary information in data.

29.4.5 Local Functions

N/A

30 MIS of GitHub Services Module

30.1 Module

GitHubServices

30.2 Uses

ProjectServices, FileServices

30.3 Syntax

30.3.1 Exported Constants

N/A

30.3.2 Exported Access Programs

Name	In	Out	Exceptions
retrieveLog	String		
createCommit	String		
pushCommit	String		

30.4 Semantics

30.4.1 State Variables

N/A

30.4.2 Environment Variables

N/A

30.4.3 Assumptions

N/A

30.4.4 Access Routine Semantics

retrieveLog(data): Extracts the user data from the parameter and calls AuthService.checkAuth(user, log) and validates that the user is authorized to make this operation. Then it obtains the project id from the data object, based on that user object, It runs a GitHub API to retrieve the logs and then returns the logs.

createCommit(data): Extracts the user data from the parameter and calls AuthService.checkAuth(user, commit) and validates that the user is authorized to make this operation. Then it obtains the necessary information from the data object, like user id, project id and file content. It then gets the HEAD commit by calling getHEADCommit(), and the tree that the HEAD commit points to by calling getTree(). Then it creates a new tree with the new content and creates a new commit. This commit is then stored in Project Data for later when the user wants to push it.

pushCommit(data): Extracts the user data from the parameter and calls AuthService.checkAuth(user, push) and validates that the user is authorized to make this operation. Then it obtains the latest commit from Project Data and then pushes it to GitHub using the API. It will use the SHA from the commit to update the reference, effectively moving the HEAD reference to the latest commit.

30.4.5 Local Functions

getHEADCommit(): Obtains the commit that HEAD points to using the GitHub API and returns it.

getTree(): Obtains the tree that HEAD commit refers to using the GitHub API and returns it.

31 MIS of Authentication Module

31.1 Module

Authentication

31.2 Uses

AuthService

31.3 Syntax

31.3.1 Exported Constants

N/A

31.3.2 Exported Access Programs

Name	In	Out	Exceptions
loginUser	String		
logoutUser	String		
openLogin			
closeLogin			

31.4 Semantics

31.4.1 State Variables

openLogin: $\mathbb{B}$

31.4.2 Environment Variables

loginButton: is a button that will trigger **openLogin()** when it is pressed
loginModal: The popup UI component for displaying the login form, it renders based on the value of **openLogin()**
submitLogin: is a button that will trigger **loginUser()** when it is pressed and if successful, triggers **closeLogin()**
logoutButton: is a button that will trigger **logoutUser()** when it is pressed

31.4.3 Assumptions

The user's auth data will be cached on browser which can be retrieved as well.

31.4.4 Access Routine Semantics

loginUser(userData): Calls the AuthService.loginAuth(userData) and passes along the login details that the user entered.

logoutUser(userData): Calls AuthService.logoutAuth(userData) passing along the userData saved on browser.

openLogin(): Assigns openLogin value to True, which opens the login modal.

closeLogin(): Assigns openLogin value to False, which closes the login modal.

31.4.5 Local Functions

N/A

32 MIS of Auth Service Module

32.1 Module

AuthService

32.2 Uses

AuthDatabaseInterface, AuthData

32.3 Syntax

32.3.1 Exported Constants

N/A

32.3.2 Exported Access Programs

Name	In	Out	Exceptions
loginAuth	String		
logoutAuth	String		
checkAuth	String, String	$\mathbb{B}$	
authenticate	String, String	$\mathbb{B}$	

32.4 Semantics

32.4.1 State Variables

N/A

32.4.2 Environment Variables

N/A

32.4.3 Assumptions

N/A

32.4.4 Access Routine Semantics

loginAuth(userData): Extracts the code needed to authenticate with the GitHub API, and then uses that to receive the tokens from GitHub which will be stored to make GitHub operations on behalf of the user by calling AuthDatabaseInterface.saveToken(token).

logoutAuth(userData): Retrieves the access token of the user and then communicates with the GitHub API to delete it to log the user out of the system.

checkAuth(userData, operation): Uses the access token to determine the user's roles and if they are authorized to perform the GitHub operation that is requested and then returns a boolean based on if it accepts or rejects the request.

authenticate(jwt, projectName): Validates the JWT token and that the user logged in to the browser has access to the project. It returns a boolean based on the answer.

32.4.5 Local Functions

N/A

33 MIS of Auth Database Interface Module

33.1 Module

AuthDatabaseInterface

33.2 Uses

AuthData

33.3 Syntax

33.3.1 Exported Constants

N/A

33.3.2 Exported Access Programs

Name	**In**	**Out**	**Exceptions**
saveToken	String		
tokenExists	String	$\mathbb{B}$	

33.4 Semantics

33.4.1 State Variables

N/A

33.4.2 Environment Variables

N/A

33.4.3 Assumptions

N/A

33.4.4 Access Routine Semantics

saveToken(token): Receives the token and saves it in the MongoDB database based on the type of token.

tokenExists(token): Checks to see if the token exists in the database to validate several use cases like the user is logged in.

33.4.5 Local Functions

N/A

34 MIS of Auth Data Module

34.1 Module

AuthData

34.2 Uses

N/A

34.3 Syntax

34.3.1 Exported Data Types

UserData: tuple of (userName: String, token: String)

34.3.2 Exported Constants

N/A

34.3.3 Exported Access Programs

N/A

34.4 Semantics

34.4.1 State Variables

N/A

34.4.2 Environment Variables

N/A

34.4.3 Assumptions

N/A

34.4.4 Access Routine Semantics

N/A

34.4.5 Local Functions

N/A

References

Carlo Ghezzi, Mehdi Jazayeri, and Dino Mandrioli. *Fundamentals of Software Engineering*. Prentice Hall, Upper Saddle River, NJ, USA, 2nd edition, 2003.

Daniel M. Hoffman and Paul A. Strooper. *Software Design, Automated Testing, and Maintenance: A Practical Approach*. International Thomson Computer Press, New York, NY, USA, 1995. URL http://citeseer.ist.psu.edu/428727.html.

35 Appendix

[Extra information if required —SS]