Akdeniz University

Department of Computer Engineering

Software Engineering Project

Webitor : Web Text Editor

Software Requirements Specification

Mustafa Arınmış, Yusuf Ramazan Tanrıkulu

Team Leader: Muhammed Fatih Özdil
Product Owner: Mustafa Arınmış

Instructor: Prof. Ümit Deniz ULUŞAR

13.04.2023

This report is submitted to the Department of Computer Engineering of Akdeniz University of the Software Engineering course CSE332.

Abbreviations	
IP	Internet Protocol
JWT	JSON Web Token
SRS	Software Requirement Specification
UI	User Interface
API	Application Programming Interface
REST	Representational State Transfer

Contents

1 Introduction

1.1 Purpose

The purpose of this software is to provide a code editing tool that allows users to store and edit their code across multiple devices, such as tablets and phones. The software is intended to improve accessibility and convenience for programmers, by allowing them to work on their code from anywhere and on any device. This document will outline the specific requirements and functionality that will be included in the software.

1.2 Product Scope

The "Webitor" is a cloud-based editor application designed to allow users to easily create, edit, and manage their documents across multiple platforms. The software is accessible through web browsers, as well as Android and iOS mobile applications. The main goal of the Webitor is to provide users with a simple and accessible tool for storing and editing text content in the cloud and synchronizing it across multiple devices.

All users will have the ability to create, edit, and save their documents to the cloud-based database, which can be accessed from any device with an internet connection. It focuses on providing a reliable and easy-to-use tool for basic editing and management.

Overall, The main objective of the software is to provide an easy-to-use tool for users to create, edit, and manage their text files across multiple platforms.,

1.3 References

This project is the second edition of Webitor, which was a UI Design course term project written for web browsers without any web framework. It does not contain a backend so the first version is a kind of mock-up with HTML, CSS and Javascript technologies.

Live https://webitorr.netlify.app/
Source: https://github.com/arinmis/webitor
Documentation: https://github.com/arinmis/webitor/blob/master/docs/final.pdf
Presentation: https://www.youtube.com/watch?v=LfCKm28LS4c&ab_channel=MustafaArinmis

Example references products:

- https://codeshare.io/
- https://www.overleaf.com/

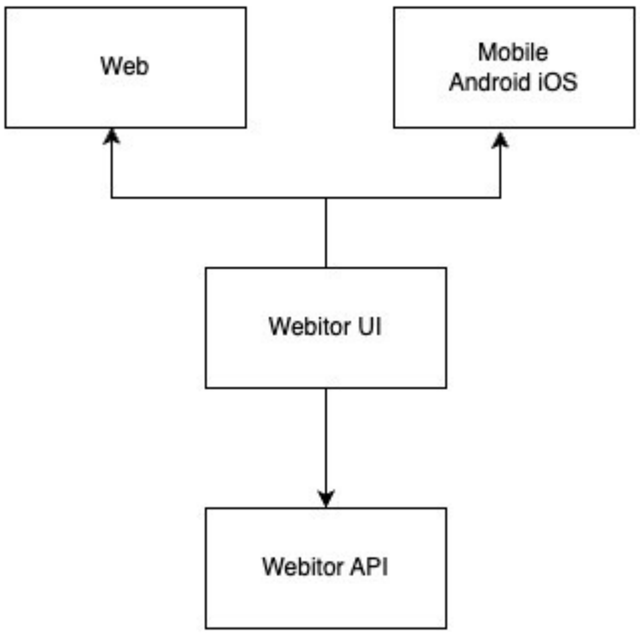
2 Overall Description

2.1 Product Perspective

Webitor consists of two kinds of clients in general. An authorized user can access the same files and folders consistently with our backend. Our server will have a REST API that is accessed from both web and mobile applications.

The client has restrictions in terms of a single file size, project size, and amount of files that can be created. This restriction considers both capability of the client’s side computation capability and cloud storage cost so users have limits which are presented in Table 1, section 2.3 User Types and Characteristics.

The system will interface with external systems such as cloud storage providers, to enable users to save their work and access it from multiple devices. A simple diagram that shows the major components of the overall system, subsystem interconnections, and external interfaces can be helpful in understanding the architecture of the system.



Figure

2.2 Product Functions

- Users will be members of the system and will open an account for themselves.
- It will be allowed to create folders and files using the file explorer on the left.
- In the opened file, it is aimed that they can write code quickly with line numbers or take notes quickly.
- The written data is automatically saved in the database and it is aimed that they can be accessed from other devices.
- Users can download the files they have written to their own locales if they wish, they can upload files from their own locales and edit them.

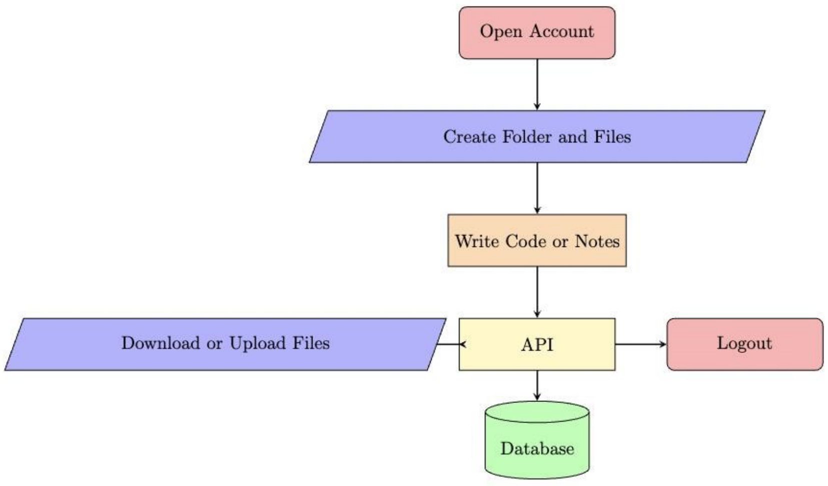


Figure 1

2.3 User Types and Characteristics

There are two different account types in the system. The first account type is basic, the second account type is premium.

There are file size limitations for basic accounts.

Examine the table1 below.

For basic accounts, a code highlight feature is offered for a language they choose.

The file sizes available for premium accounts have been increased. Highlight feature available in all languages

	Basic	**Premium**
Single file size	100KB	1MB
Project size	1MB	5MB
File amount	10	50

Table 1

2.4 Operating Environment

	Basic	**Premium**
Web	Yes	Yes
Android	Yes	Yes
iOS	No	Yes
Desktop (MacOS, Linux, Windows)	No	No

Table 2

2.5 Design and Implementation Constraints

Technical Constraints

1. **File size and project size restrictions:** The client application should be designed to enforce restrictions on the size of a single file, project size, and the number of files that can be created to consider the capability of the client's side computation and cloud storage costs.
2. **Internet Connection:** The written texts are not kept on the client's device and are saved directly to the database. That's why an internet connection is essential.
3. **Synchronization:** The application should synchronize file content among different user client types for the same users, like Web Browser and Android.
4. **Platform:** The software must be accessible through web browsers, Android and iOS mobile applications, which means that the application must be designed to be compatible with all the major web browsers such as Chrome, Firefox, Safari, Edge, and Opera. We do not support IE. Additionally, it should be designed to be compatible with the latest mobile operating systems such as Android and iOS.
5. **Technologies:** We work on a lot of technologies such as TypeScript, HTML, CSS and the team should know these technologies.
6. **Scalability**: The system must be scalable enough to handle a large number of users.

Implementation Constraints

1. **Usability and user experience:** The application must be easy to use, intuitive, and provide a good user experience, with minimal user training required.
2. **Performance and scalability:** The application must be designed to have fast response times and minimal latency, even as the user base grows.
3. **Authorization**: We handle security using JWT) for blocking unauthorized access.
4. **Encryption:** Data passed between the client and server will be encrypted with SSL. HTTPS will be used instead of HTTP.

2.6 User Documentation

Documentation will be attached to the repository as a markdown and video tutorial.

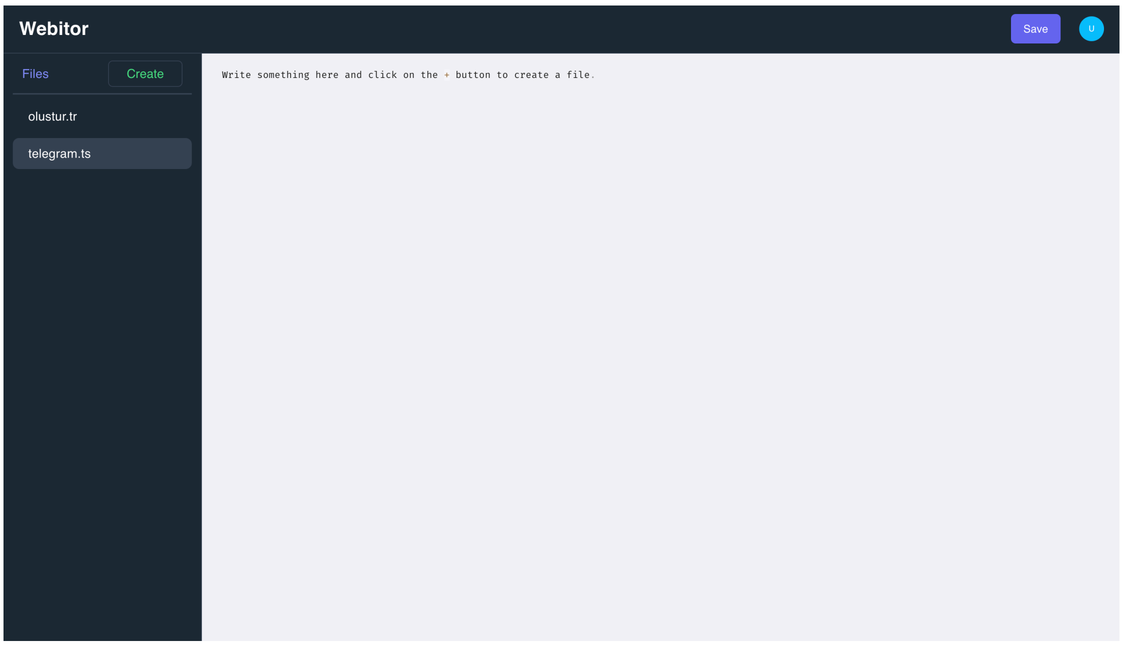
2.7 Assumptions and Dependencies

Client computers and mobile phones should have enough computation power that enables them to parse file content and highlight it on the client side. Network bandwidth might also be critical to saving edited files on time.

3 External Interface Requirements

3.1 User Interfaces

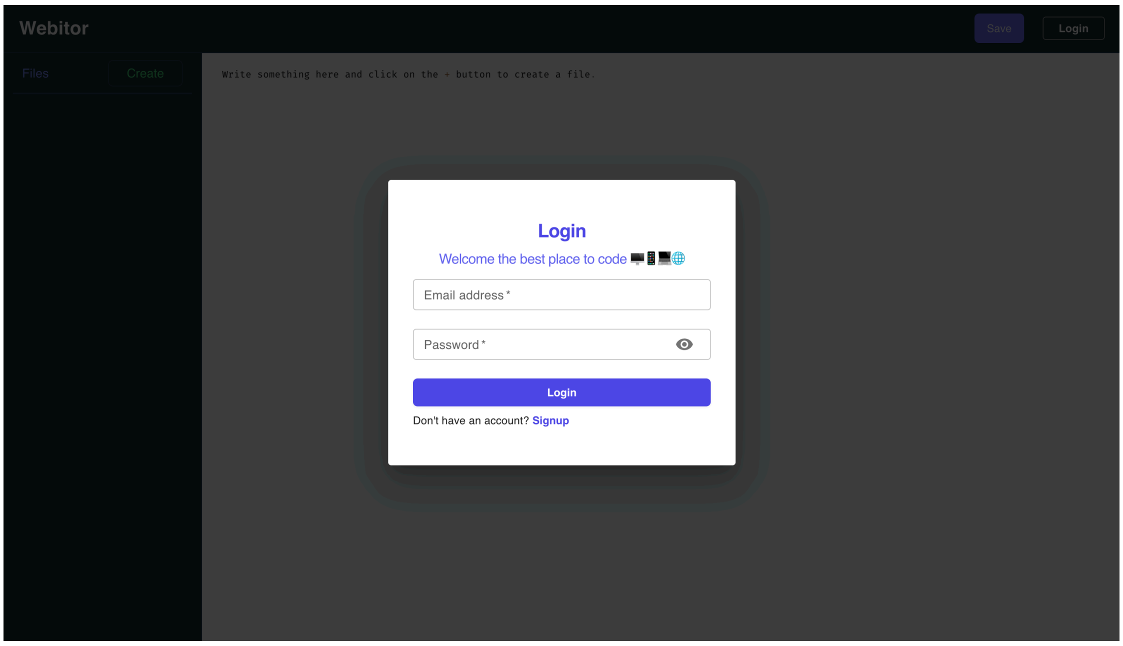
Main View



Write something here and click on the + button to create a file.

Figure 1

Login Screen



Write something here and click on the + button to create a file.

Login

Welcome the best place to code 🖥️📱💻🌐

Email address *

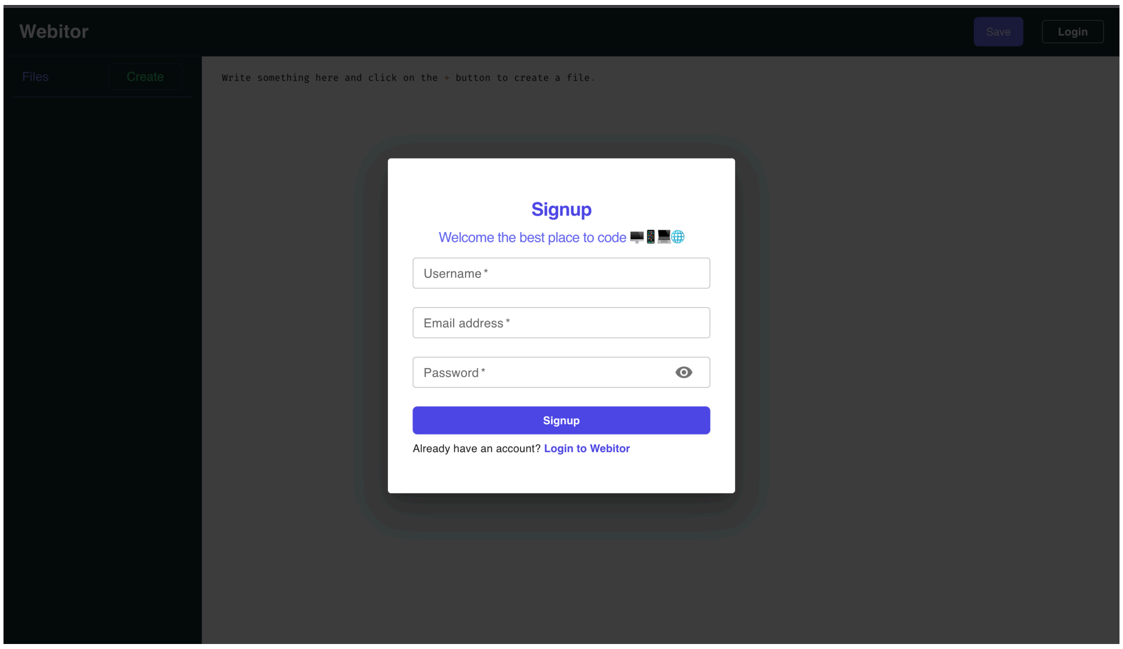
Password *

Login

Don't have an account? Signup

Figure 2

Signup Screen



Write something here and click on the + button to create a file.

Signup

Welcome the best place to code 🖥️📱💻🌐

Username *

Email address *

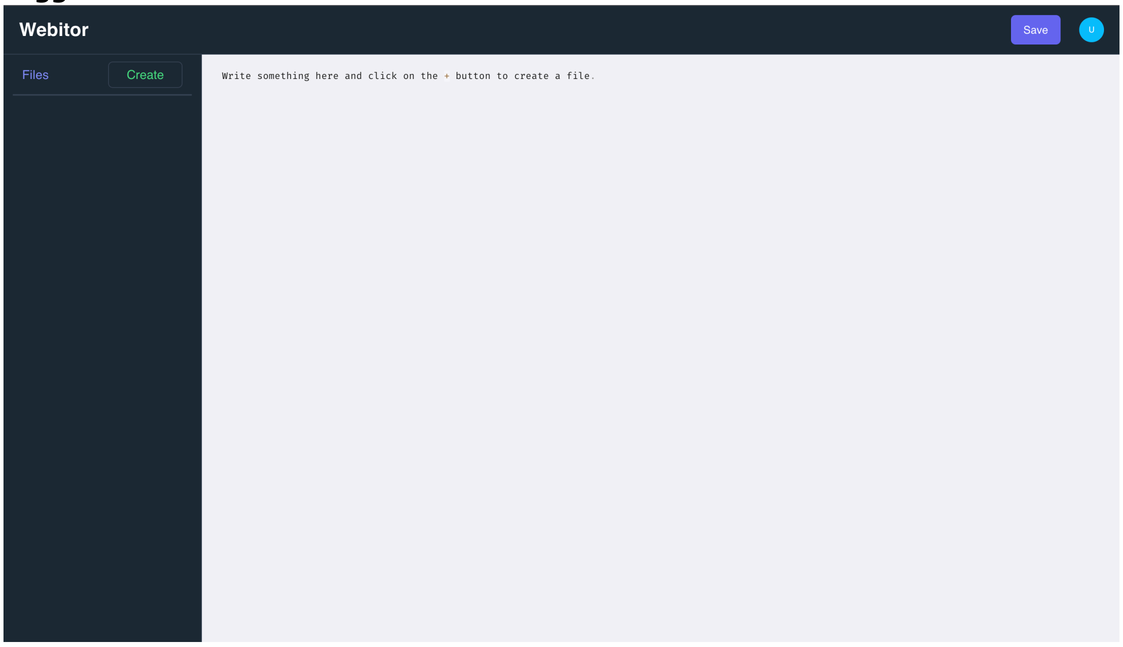
Password *

Signup

Already have an account? Login to Webitor

Figure 3

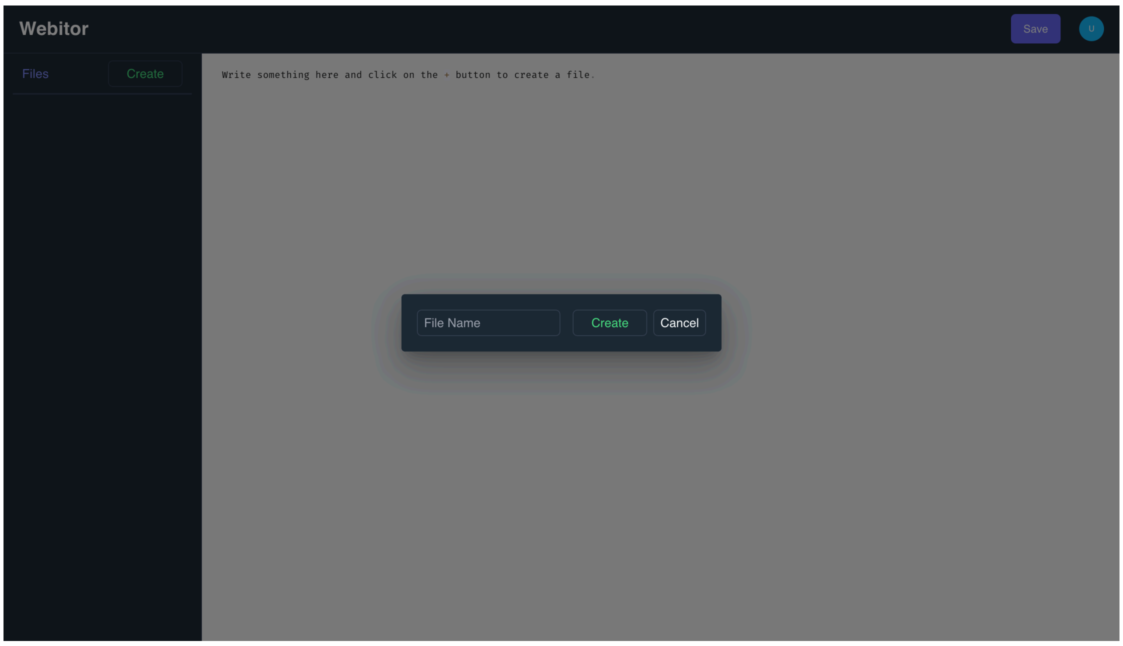
Logged In Screen



Write something here and click on the + button to create a file.

Figure 4

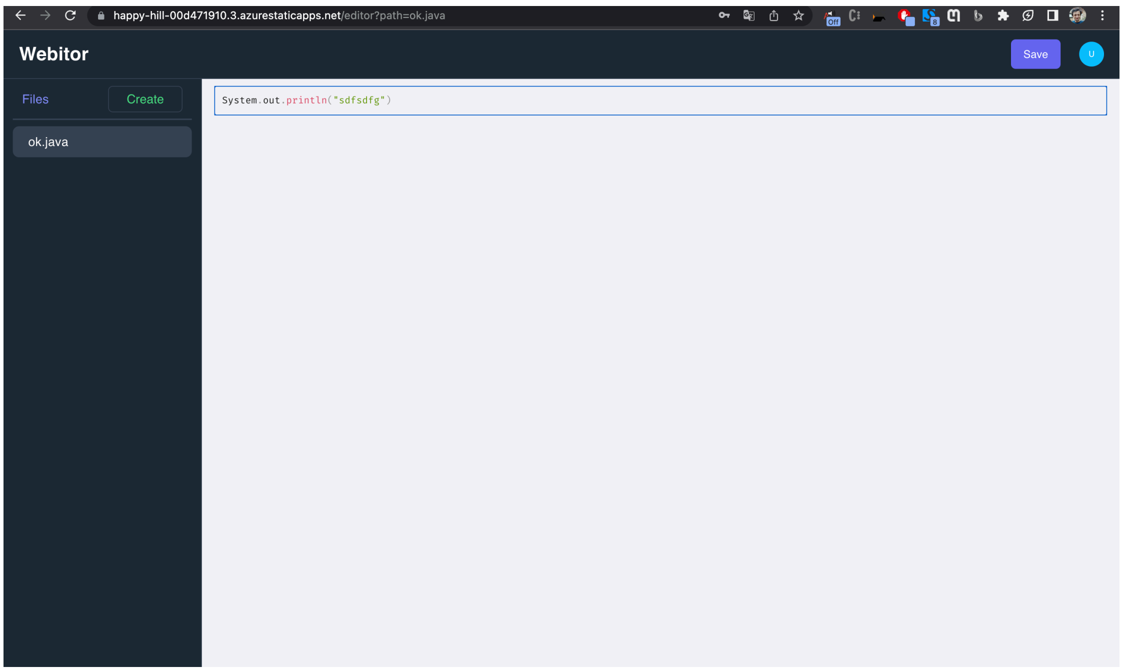
Create File Screen



Write something here and click on the + button to create a file.

Figure 5

Select File


happy-hill-00d471910.3.azurestaticapps.net/editor?path=ok.java
Webitor
Save
U
Files
Create
ok.java

```
System.out.println("sdfsdfg")
```

Figure 6

In summary, there will be 3 main component:

- Login/signup:
- file CRUD:
  - File manager
  - Text editing panel
  - Useful features(dark mode, font type, font size)
- Settings:
  - User credentials update

3.2 Hardware Interfaces

Webitor will require hardware interfaces that support web browsers, Android, and iOS mobile applications and are capable of transmitting and receiving text-based files securely using HTTPS protocol.

3.3 Software Interfaces

Frontend

1. Typescript - v5.0.4: A superset of JavaScript that provides additional static typing to improve code reliability and maintainability.
2. React - v18.2.0: A popular JavaScript library for building user interfaces.
3. Next.js - v13.3.1 : A framework built on top of React that enables server-side rendering and provides a more streamlined development experience.
4. Material-UI - v5.12.0: A React-based component library that provides pre-built UI components.
5. Styled Components - v5.3.9: A CSS-in-JS library that allows developers to write CSS in their JavaScript code.
6. TanStack Query - v4.29.3: A server-side state management library. Toss out that granular state management, manual refetching, and endless bowls of async-spaghetti code. TanStack

Query gives you declarative, always-up-to-date auto-managed queries and mutations that directly improve both your developer and user experiences.

7. Redux-Toolkit v1.9.3: A client-side state management library.

Mobile

1. Typescript - v5.0.4: A superset of JavaScript that provides additional static typing to improve code reliability and maintainability.
2. React Native - v0.71.6: A mobile development framework that allows the building native mobile apps for both Android and iOS platforms using JavaScript and React.
3. Styled Components - v5.3.9: A library for styling React components using CSS-in-JS approach.
4. React-Native-cli - v2.0.1: A command-line interface for building and managing React Native projects.
5. React Native Elements - v4.0: A library of reusable UI components for React Native apps.
6. React Navigation v3.2.0: A library for implementing navigation and routing in React Native apps.
7. TanStack Query - v4.29.3: A library for fetching and managing most of the server-side state in React apps.
8. Redux-Toolkit v1.9.3: A library for managing most of the client-side state in React apps using Redux.

Backend

1. ASP.NET Core - v6.0: A cross-platform framework for building modern, cloud-based web applications and microservices.
2. C# - v10.0: A modern, object-oriented programming language used for building applications that run on .NET.
3. Entity Framework Core - v6.0.0 : A lightweight and extensible Object-Relational Mapping (ORM) framework that enables developers to work with databases using .NET objects.
4. MediatR - v12.0.1 : Defines an object that encapsulates how a set of objects interact.
5. RESTful API: An architectural style for building web services that uses HTTP protocols to create a scalable and flexible interface between different systems.
6. Microsoft.AspNetCore.Authentication.JwtBearer - v6.0.0 : A JSON-based open standard used for creating access tokens that authenticate the identity of users in web applications.
7. Microsoft.EntityFrameworkCore.InMemory - v6.0.6 : designed to be a general-purpose database for testing and is not designed to mimic a relational database.
8. Swagger/OpenAPI - v6.5.0: A standard for defining and documenting RESTful APIs that provides a user-friendly interface for interacting with web services.

Data items or messages going in and out of the system:

- **User data**: Data items coming into the system include user authentication information and user-generated data such as documents. API requests from the front end and mobile interfaces are used to retrieve and manipulate user data. The purpose of these messages is to enable users to interact with their files and folders within the Webitor environment.
- **Communications**: Communications between Webitor components are typically performed using HTTP protocols. Data sharing between software components is important for Webitor to function properly, but the specific mechanism for data sharing is not mentioned in the given information.

3.4 Communications Interfaces

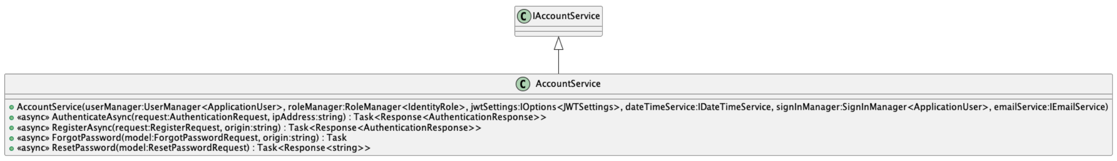
Webitor uses HTTPS as the communication protocol, ensuring secure data transfer rates and synchronization mechanisms for accessing and managing documents across multiple platforms.

4 System Features / Requirements

This section outlines the key features and requirements of the Webitor system

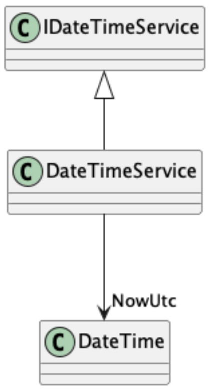
You can see UML diagrams for our system features.

Account Service

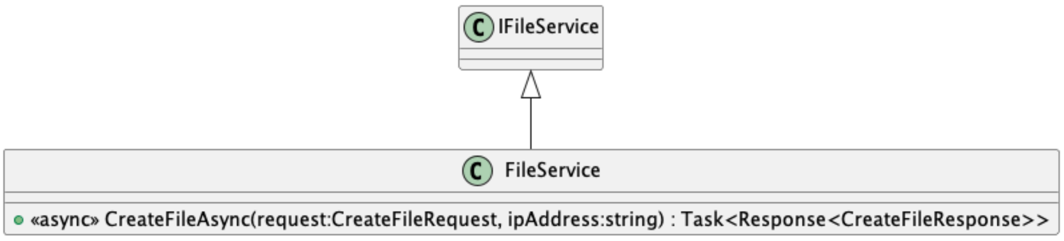


IAccountService
AccountService
AccountService(userManager:UserManager<ApplicationUser>, roleManager:RoleManager<IdentityRole>, jwtSettings:IOptions<JWTSettings>, dateTimeService:IDateTimeService, signInManager:SignInManager<ApplicationUser>, emailService:IEmailService)
«async» AuthenticateAsync(request:AuthenticationRequest, ipAddress:string) : Task<Response<AuthenticationResponse>>
«async» RegisterAsync(request:RegisterRequest, origin:string) : Task<Response<AuthenticationResponse>>
«async» ForgotPassword(model:ForgotPasswordRequest, origin:string) : Task
«async» ResetPassword(model:ResetPasswordRequest) : Task<Response<string>>

Date Service

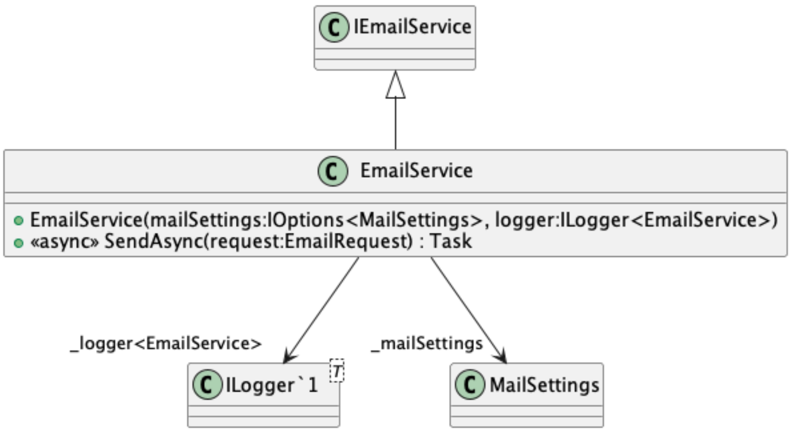


File Service



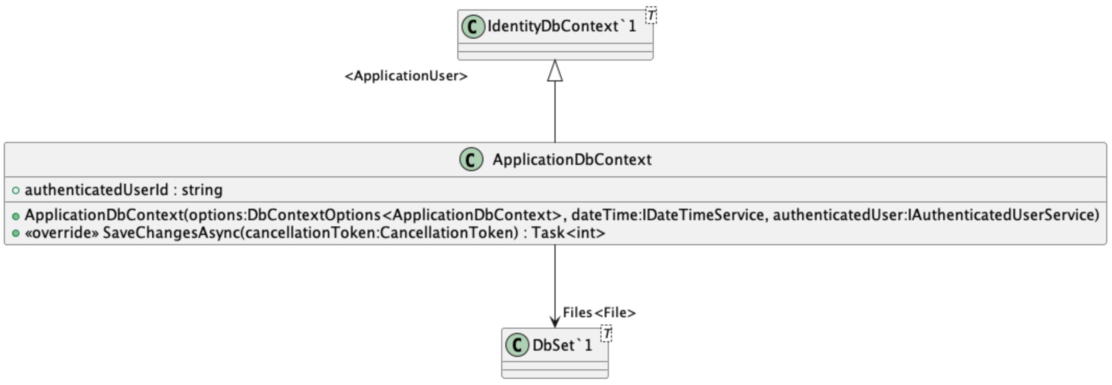
IFileService
FileService
«async» CreateFileAsync(request:CreateFileRequest, ipAddress:string) : Task<Response<CreateFileResponse>>

Email Service

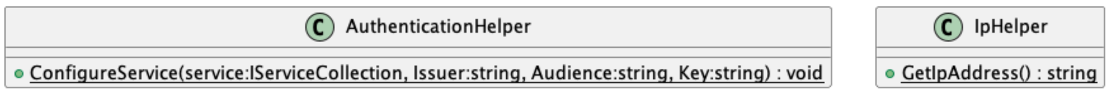


IEmailService
EmailService
EmailService(mailSettings:IOptions<MailSettings>, logger:ILogger<EmailService>)
«async» SendAsync(request:EmailRequest) : Task
_logger<EmailService>
_mailSettings
ILogger`1
T
MailSettings

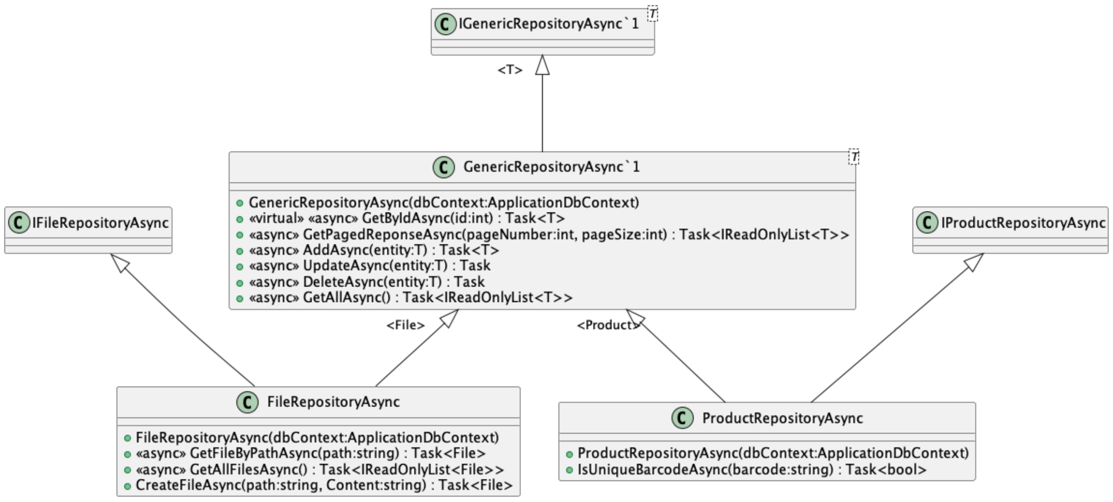
DB Context


IdentityDbContext`1
T
<ApplicationUser>
ApplicationDbContext
authenticatedUserId : string
ApplicationDbContext(options:DbContextOptions<ApplicationDbContext>, dateTime:IDateTimeService, authenticatedUser:IAuthenticatedUserService)
«override» SaveChangesAsync(cancellationToken:CancellationToken) : Task<int>
Files<File>
DbSet`1
T

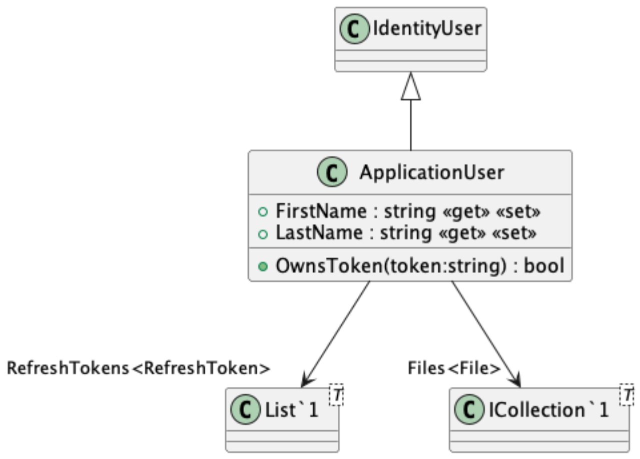
Authentication Helper


AuthenticationHelper
ConfigureService(service:IServiceCollection, Issuer:string, Audience:string, Key:string) : void
IpHelper
GetIpAddress() : string

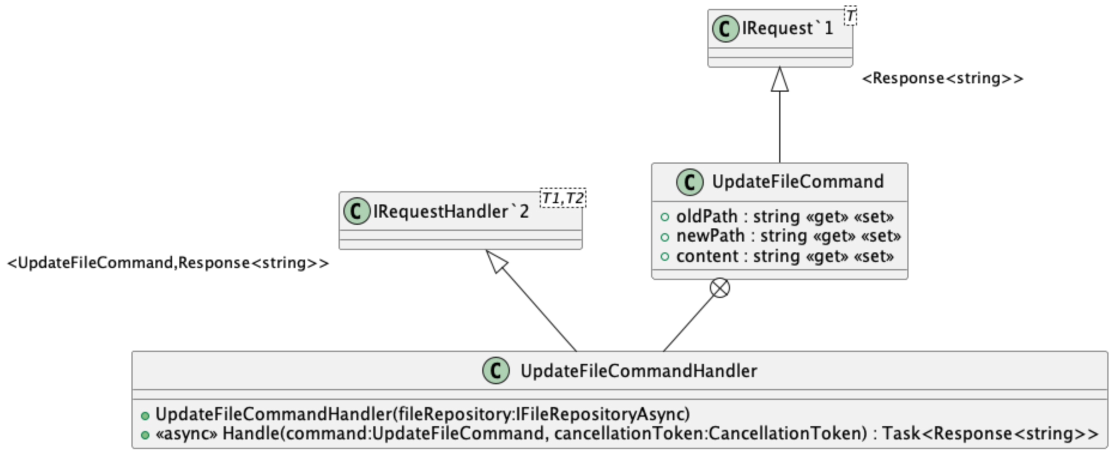
Repositories


IGenericRepositoryAsync`1
T
<T>
GenericRepositoryAsync`1
T
GenericRepositoryAsync(dbContext:ApplicationDbContext)
«virtual» «async» GetByIdAsync(id:int) : Task<T>
«async» GetPagedReponseAsync(pageNumber:int, pageSize:int) : Task<IReadOnlyList<T>>
«async» AddAsync(entity:T) : Task<T>
«async» UpdateAsync(entity:T) : Task
«async» DeleteAsync(entity:T) : Task
«async» GetAllAsync() : Task<IReadOnlyList<T>>
IFileRepositoryAsync
IProductRepositoryAsync
<File>
<Product>
FileRepositoryAsync
FileRepositoryAsync(dbContext:ApplicationDbContext)
«async» GetFileByPathAsync(path:string) : Task<File>
«async» GetAllFilesAsync() : Task<IReadOnlyList<File>>
CreateFileAsync(path:string, Content:string) : Task<File>
ProductRepositoryAsync
ProductRepositoryAsync(dbContext:ApplicationDbContext)
IsUniqueBarcodeAsync(barcode:string) : Task<bool>

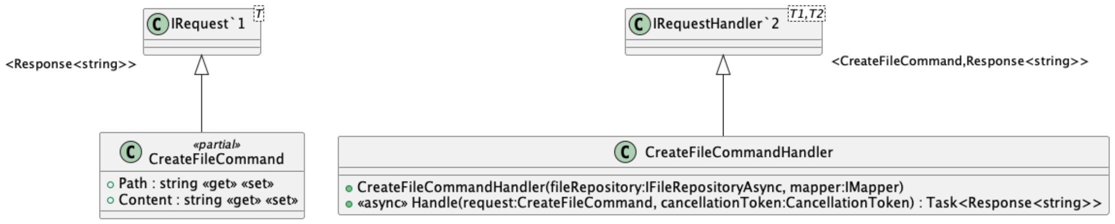
User


IdentityUser
ApplicationUser
FirstName : string «get» «set»
LastName : string «get» «set»
OwnsToken(token:string) : bool
RefreshTokens<RefreshToken>
Files<File>
List`1
T
ICollection`1
T

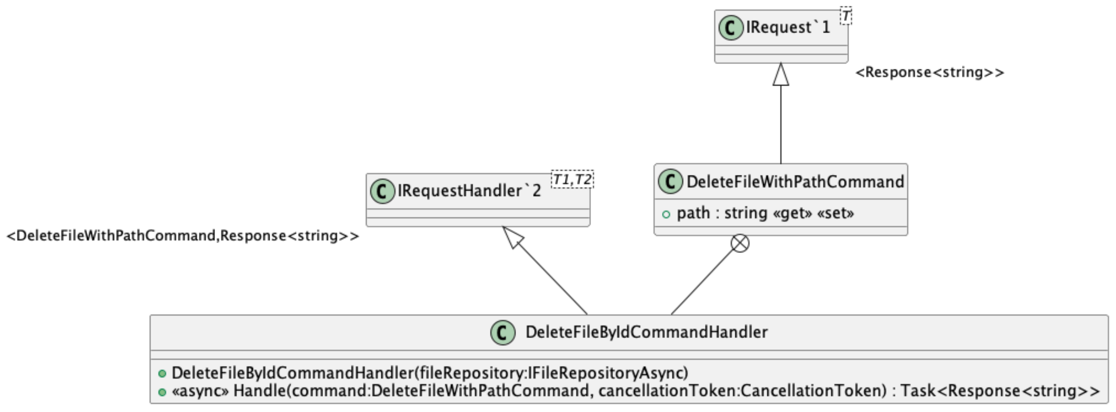
Update File Command


IRequest`1
T
<Response<string>>
IRequestHandler`2
T1,T2
<UpdateFileCommand,Response<string>>
UpdateFileCommand
oldPath : string «get» «set»
newPath : string «get» «set»
content : string «get» «set»
UpdateFileCommandHandler
UpdateFileCommandHandler(fileRepository:IFileRepositoryAsync)
«async» Handle(command:UpdateFileCommand, cancellationToken:CancellationToken) : Task<Response<string>>

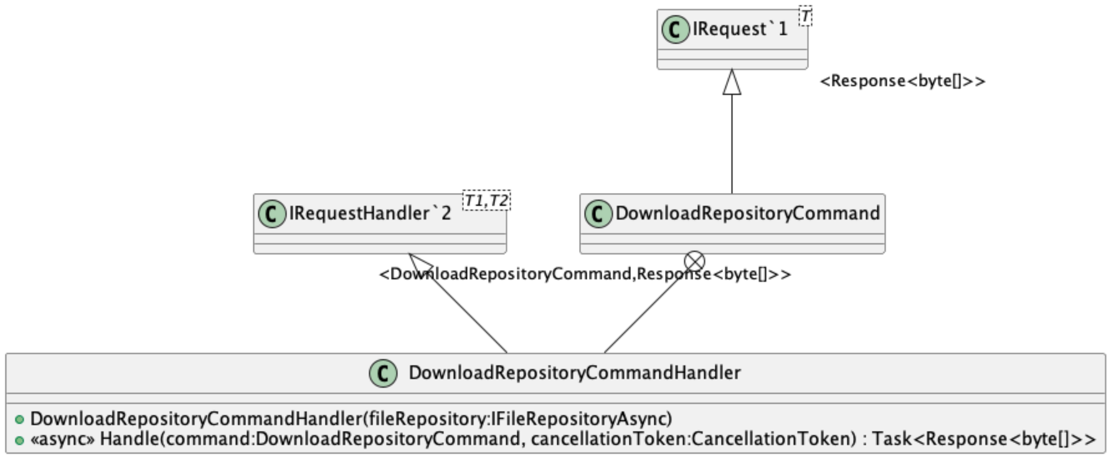
Create File Command


IRequest`1
T
<Response<string>>
«partial»
CreateFileCommand
Path : string «get» «set»
Content : string «get» «set»
IRequestHandler`2
T1,T2
<CreateFileCommand,Response<string>>
CreateFileCommandHandler
CreateFileCommandHandler(fileRepository:IFileRepositoryAsync, mapper:IMapper)
«async» Handle(request:CreateFileCommand, cancellationToken:CancellationToken) : Task<Response<string>>

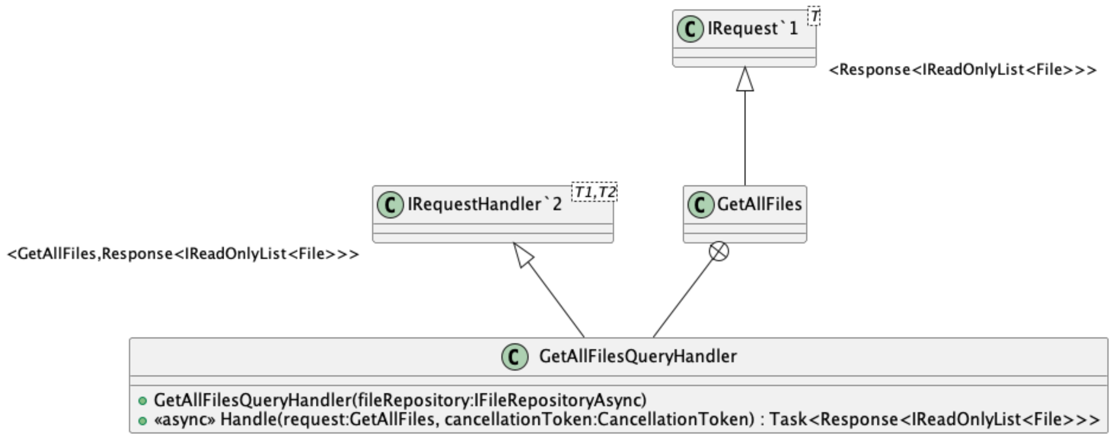
Delete File With Path Command


IRequest`1
T
<Response<string>>
IRequestHandler`2
T1,T2
DeleteFileWithPathCommand
path : string «get» «set»
<DeleteFileWithPathCommand,Response<string>>
DeleteFileByIdCommandHandler
DeleteFileByIdCommandHandler(fileRepository:IFileRepositoryAsync)
«async» Handle(command:DeleteFileWithPathCommand, cancellationToken:CancellationToken) : Task<Response<string>>

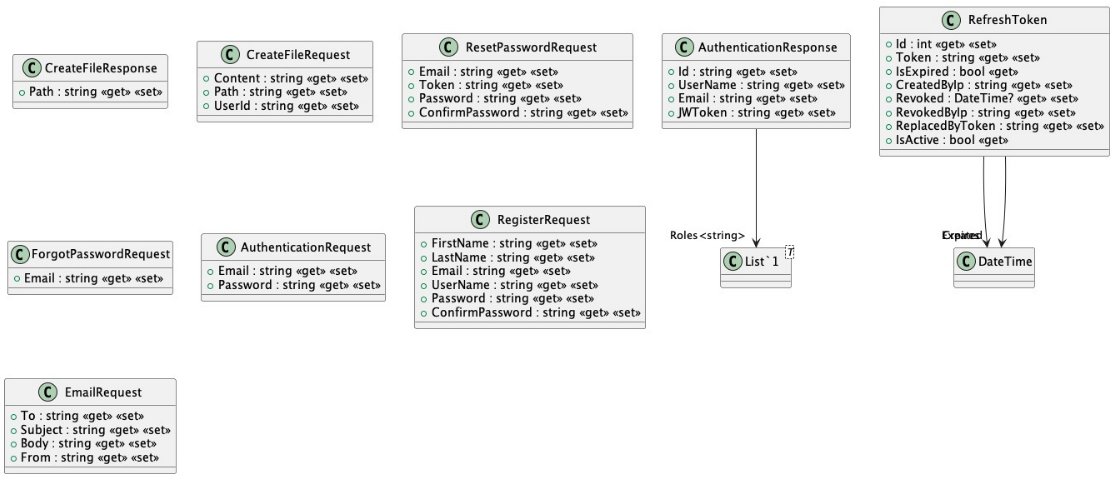
Download Repository Command


IRequest`1
T
<Response<byte[]>>
IRequestHandler`2
T1,T2
DownloadRepositoryCommand
<DownloadRepositoryCommand,Response<byte[]>>
DownloadRepositoryCommandHandler
DownloadRepositoryCommandHandler(fileRepository:IFileRepositoryAsync)
«async» Handle(command:DownloadRepositoryCommand, cancellationToken:CancellationToken) : Task<Response<byte[]>>

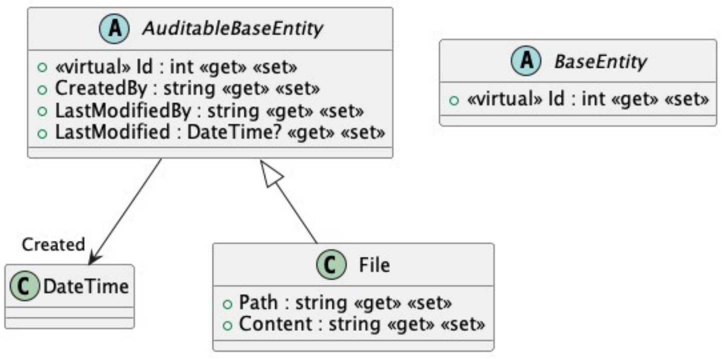
Get All Files Query


IRequest`1
T
<Response<IReadOnlyList<File>>>
IRequestHandler`2
T1,T2
GetAllFiles
<GetAllFiles,Response<IReadOnlyList<File>>>
GetAllFilesQueryHandler
GetAllFilesQueryHandler(fileRepository:IFileRepositoryAsync)
«async» Handle(request:GetAllFiles, cancellationToken:CancellationToken) : Task<Response<IReadOnlyList<File>>>

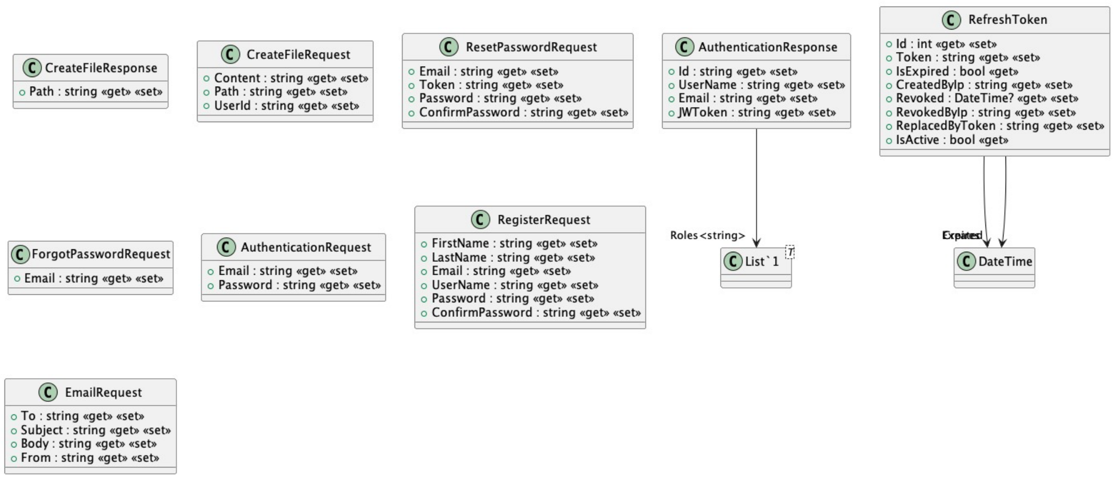
DTO


CreateFileResponse
Path : string «get» «set»
CreateFileRequest
Content : string «get» «set»
Path : string «get» «set»
UserId : string «get» «set»
ResetPasswordRequest
Email : string «get» «set»
Token : string «get» «set»
Password : string «get» «set»
ConfirmPassword : string «get» «set»
AuthenticationResponse
Id : string «get» «set»
UserName : string «get» «set»
Email : string «get» «set»
JWToken : string «get» «set»
RefreshToken
Id : int «get» «set»
Token : string «get» «set»
IsExpired : bool «get»
CreatedByIp : string «get» «set»
Revoked : DateTime? «get» «set»
RevokedByIp : string «get» «set»
ReplacedByToken : string «get» «set»
IsActive : bool «get»
ForgotPasswordRequest
Email : string «get» «set»
AuthenticationRequest
Email : string «get» «set»
Password : string «get» «set»
RegisterRequest
FirstName : string «get» «set»
LastName : string «get» «set»
Email : string «get» «set»
UserName : string «get» «set»
Password : string «get» «set»
ConfirmPassword : string «get» «set»
Roles <string>
List`1
T
Expires
Created
DateTime
EmailRequest
To : string «get» «set»
Subject : string «get» «set»
Body : string «get» «set»
From : string «get» «set»

Entities

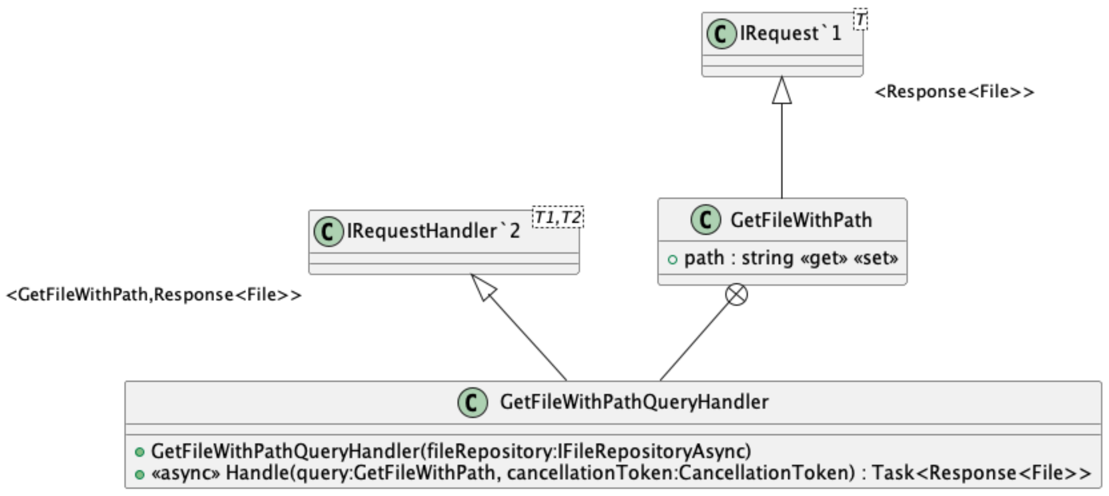

AuditableBaseEntity
«virtual» Id : int «get» «set»
CreatedBy : string «get» «set»
LastModifiedBy : string «get» «set»
LastModified : DateTime? «get» «set»
BaseEntity
«virtual» Id : int «get» «set»
Created
DateTime
File
Path : string «get» «set»
Content : string «get» «set»

Entities

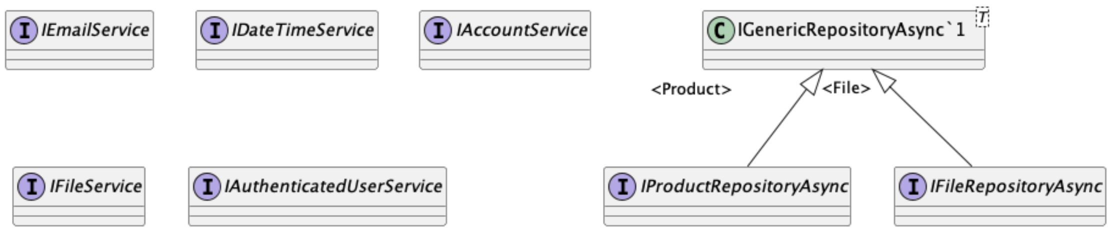


CreateFileResponse
Path : string «get» «set»
CreateFileRequest
Content : string «get» «set»
Path : string «get» «set»
UserId : string «get» «set»
ResetPasswordRequest
Email : string «get» «set»
Token : string «get» «set»
Password : string «get» «set»
ConfirmPassword : string «get» «set»
AuthenticationResponse
Id : string «get» «set»
UserName : string «get» «set»
Email : string «get» «set»
JWToken : string «get» «set»
RefreshToken
Id : int «get» «set»
Token : string «get» «set»
IsExpired : bool «get»
CreatedByIp : string «get» «set»
Revoked : DateTime? «get» «set»
RevokedByIp : string «get» «set»
ReplacedByToken : string «get» «set»
IsActive : bool «get»
ForgotPasswordRequest
Email : string «get» «set»
AuthenticationRequest
Email : string «get» «set»
Password : string «get» «set»
RegisterRequest
FirstName : string «get» «set»
LastName : string «get» «set»
Email : string «get» «set»
UserName : string «get» «set»
Password : string «get» «set»
ConfirmPassword : string «get» «set»
Roles <string>
List`1
T
Expires
Created
DateTime
EmailRequest
To : string «get» «set»
Subject : string «get» «set»
Body : string «get» «set»
From : string «get» «set»

Get File With Path Query


IRequest`1
T
<Response<File>>
IRequestHandler`2
T1,T2
GetFileWithPath
path : string «get» «set»
<GetFileWithPath,Response<File>>
GetFileWithPathQueryHandler
GetFileWithPathQueryHandler(fileRepository:IFileRepositoryAsync)
«async» Handle(query:GetFileWithPath, cancellationToken:CancellationToken) : Task<Response<File>>

Interfaces



4.1 User Class 1 - Basic User

4.1.1 System Feature: File Management

4.1.1.1. Description and Priority

Allow basic users to create folders, files inside it with any extension. Basic users can upload and download files, with a single file size restriction of 100kb and a project file size restriction of 1mb. Basic users are limited to a maximum of 10 files. This feature is of high priority as it is the core functionality of the application. 4.1.1.2. Stimulus/Response Sequences

- User creates a new folder/file.
- System adds the folder/file to the user's storage limit and displays it on the file explorer.
- User uploads a file.
- System stores the file in the user's storage limit and displays it on the file explorer.
- User downloads a file.
- System retrieves the file from the user's storage limit and downloads it to the user's device.

4.1.1.3. Functional Requirements

FR1: Restrict single file size to 100kb for basic users.
FR2: Restrict project file size to 1mb for basic users. FR3:
Limit basic users to a maximum of 10 files.

4.2 User Class 2 - Premium User

4.2.1 System Feature: File Management

4.2.1.1. Description and Priority

Allow premium users to create folders, files inside in it with any extension. Premium users can upload and download files, with a single file size restriction of 1mb and a project file size restriction of 5mb. Premium users can have up to 50 files. This feature is of high priority as it is the core functionality of the application.

4.2.1.2. Stimulus/Response Sequences

- User creates a new folder/file.
- System adds the folder/file to the user's storage limit and displays it on the file explorer.
- User uploads a file.
- System stores the file in the user's storage limit and displays it on the file explorer.
- User downloads a file.
- System retrieves the file from the user's storage limit and downloads it to the user's device.

4.2.1.3. Functional Requirements

FR1: Restrict single file size to 1mb for premium users.
FR2: Restrict project file size to 5mb for premium users.
FR3: Allow premium users to have up to 50 files.

4.3 User Class 3 - Basic User/Premium User

4.3.1 System Feature: User Authentication and Authorization

4.3.1.1. Description and Priority

Implement user authentication and authorization to ensure that only authorized users can access their own files and folders. This feature is of high priority as it ensures the security of the application.

4.3.1.2. Stimulus/Response Sequences

- Users enter their credentials.
- System verifies the credentials and grants access to the user's files and folders.
- User attempts to access another user's files and folders.
- System denies access and displays an error message.

4.3.1.3. Functional Requirements

FR1: Implement user authentication through measures such as passwords and multi-factor authentication.
FR2: Implement user authorization to ensure that users can only access their own files and folders.

4.3.2 System Feature: Text Editor

4.3.2.1. Description and Priority

Provide a text editor for users to write, compile, and execute code. This feature is of high priority as it is essential for editing and running code.

4.3.2.2. Stimulus/Response Sequences

- User opens the text editor.
- System displays the text editor interface.
- User writes code.

- System highlights syntax and provides auto-completion suggestions.
- User compiles and executes code.
- System displays output and errors.

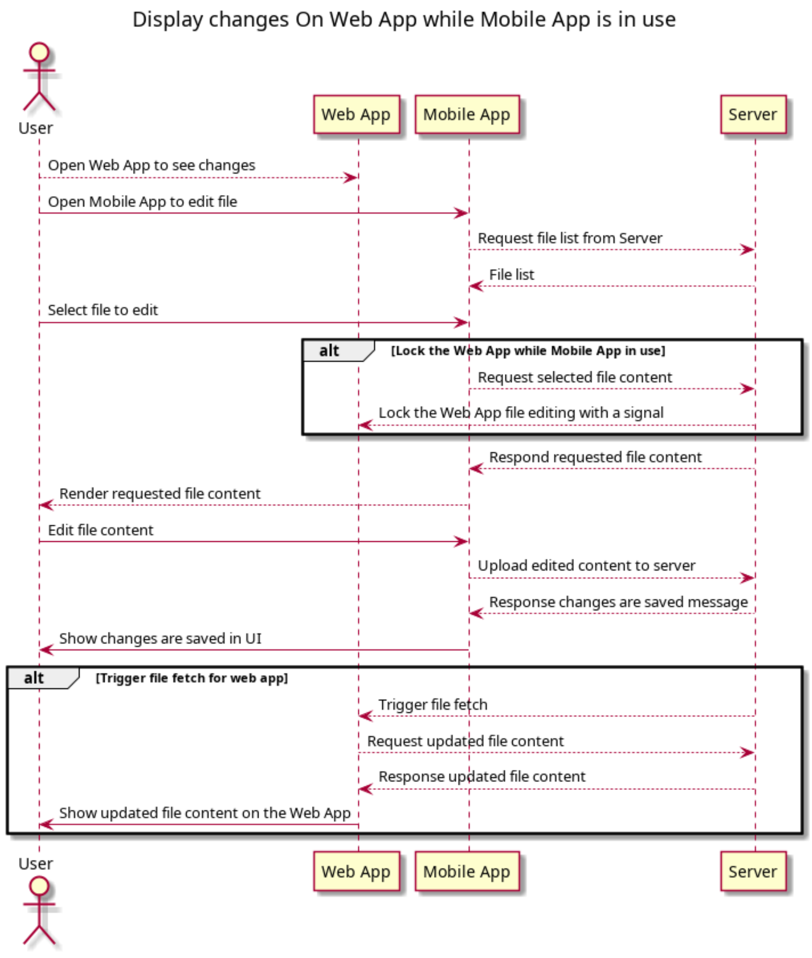
4.3.2.3. Functional Requirements

FR1: Provide a text editor with syntax highlighting and auto-completion suggestions. FR2: Allow users to compile and execute code directly from the text editor. FR3: Display output and errors in the text editor interface.

4.3.3 System Feature: Real-time Synchronization

4.3.3.1. Description and Priority

Provide real-time synchronization of code changes between multiple platform(Mobile and Web App). This feature is of high priority as it prevent data inconsistency because of out dated client.


Display changes On Web App while Mobile App is in use
User
Web App
Mobile App
Server
Open Web App to see changes
Open Mobile App to edit file
Request file list from Server
File list
Select file to edit
alt
[Lock the Web App while Mobile App in use]
Request selected file content
Lock the Web App file editing with a signal
Respond requested file content
Render requested file content
Edit file content
Upload edited content to server
Response changes are saved message
Show changes are saved in UI
alt
[Trigger file fetch for web app]
Trigger file fetch
Request updated file content
Response updated file content
Show updated file content on the Web App
User
Web App
Mobile App
Server

4.3.3.2. Stimulus/Response Sequences

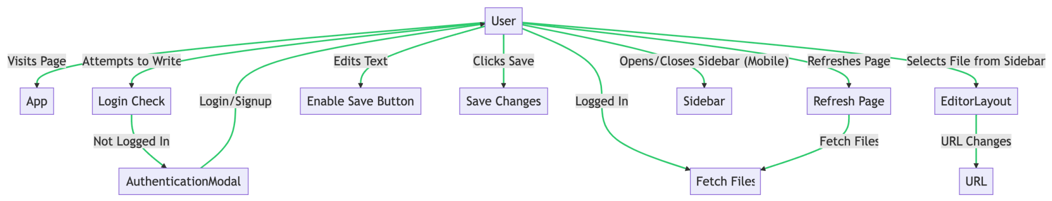
- User A makes a change to the code.
- System sends the change to the server.
- System sends the change to User B's interface.
- User B's interface updates with the change automatically.

4.3.3.3. Functional Requirements

FR1: Provide real-time synchronization of code changes between multiple users. FR2: Update all users' interfaces with changes automatically.

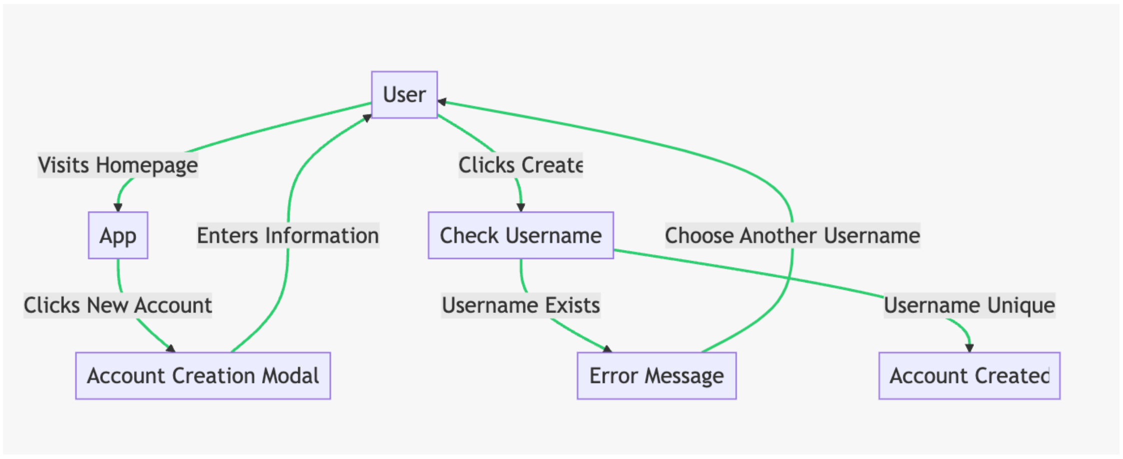
5 Use Cases

General Use Case Diagram



5.1 Creating a new Account

The purpose of this use case is to describe the procedure of creating an account in the system **Pre-conditions:**
None
Post-conditions:
• An account is created for the user **Basic Flow:**
1. The user is in the homepage.
2. The user clicks on the 'new account' link and is taken to the account creation modal.
3. The user enters all their information and clicks the 'create' button.
4. If the username already exists, an error message appears to the user and asks the user to choose another username. Otherwise, a confirmation appears letting the user know that the account has been created.



5.2 Deleting an Account

The purpose of this use case is to describe the procedure of deleting an account in the system **Pre-conditions:**
• User must have an account • User must be logged in to the system
Post-conditions:
• An account is deleted for the user **Basic Flow:**

1. The user is in homepage to display files
2. The user clicks on the 'settings' link and is taken to the settings page
3. The user clicks on the 'delete account' button
4. The user is warned by an alert to confirm the operation
5. If the user confirms the operation, the account will be deleted and the user will be routed to the login page

5.3 Create File

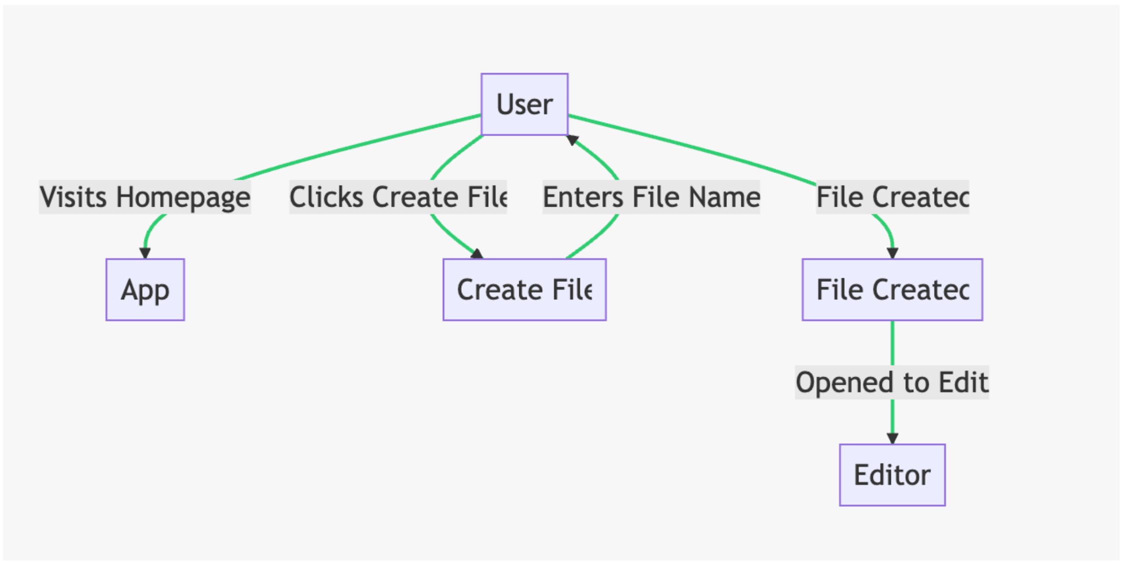
The purpose of this use case is to describe the procedure of creating a file in the system **Pre-conditions:**

• User must have an account • User must be logged in to the system

Post-conditions:

• A new file will be created to edit **Basic Flow:**

1. The user is in homepage to access file manager
2. The user clicks on the 'create file' button and enters the file name with its file extension
3. The newly created file will be opened to edit



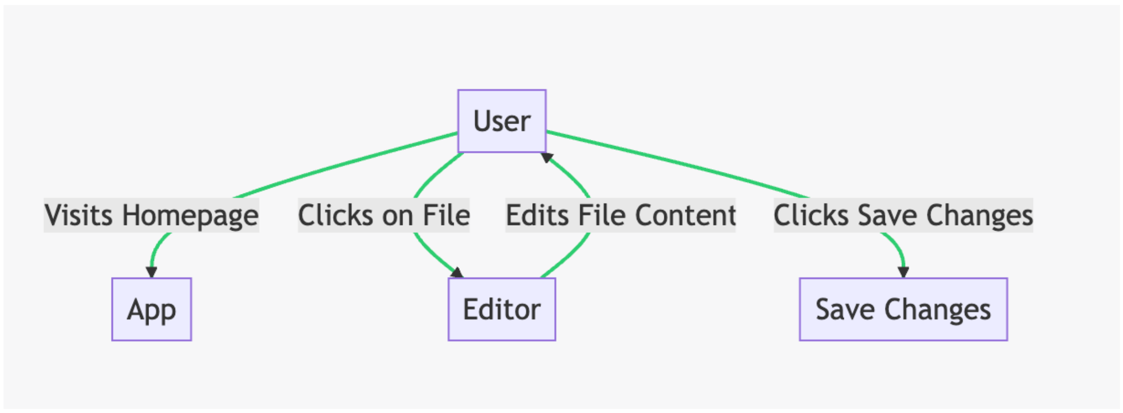
In this diagram, the user visits the homepage, clicks on the 'create file' button, enters the file name with its file extension, and the file is created and opened for editing.

5.4 Edit File

The purpose of this use case is to describe the procedure of editing a file in the system **Pre-conditions:**

• User must have an account
• User must be logged in to the system
• Target file must be created **Post-conditions:**
• The target file will be edited **Basic Flow:**

1. The user is in homepage to access file manager
2. The user hovers and clicks on the target file to edit
3. The user edits file content as desired
4. 'Save Changes' button will be clicked in order to save changes



In this diagram, the user visits the homepage, clicks on a file to edit, edits the file content, and clicks 'Save Changes' to save the changes.

5.5 Change Font Family

The purpose of this use case is to describe the procedure of changing font-family of the system **Pre-conditions:**
• User must have an account • User must logged in to the system **Post-conditions:**
• The font family of the system will be changed **Basic Flow:**
1. The user is in homepage
2. The user clicks on to 'settings' button and is routed to 'settings' page
3. The user selects the target font family from the listed drop-down menu and clicks on it

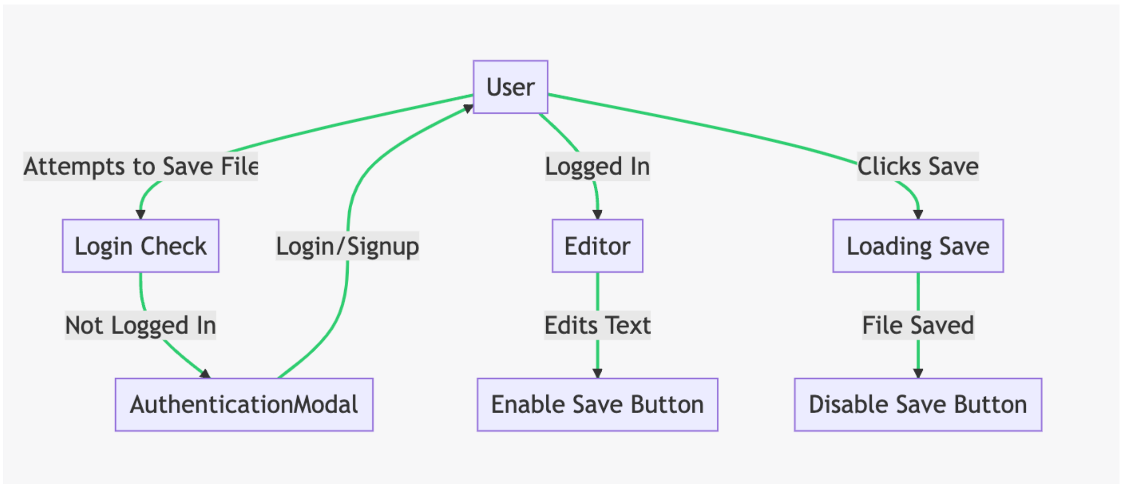
5.6 Change Color Mode

The purpose of this use case is to describe the procedure of switching the color mode of the system between dark and light **Pre-conditions:**
• User must have an account • User must logged in to the system **Post-conditions:**
• The color of the system will be changed **Basic Flow:**
1. The user is in homepage
2. The user clicks to switch color mode button

5.7 Upload File

The purpose of this use case is to describe the procedure of uploading a file in the system to edit **Pre-conditions:**
• User must have an account
• User must be logged in to the system
• Target file size must exceed a certain limit(see 2.3) according to user type **Post-conditions:**
• The target file will be uploaded to the system **Basic Flow:**
1. The user is in homepage to access file manager
2. The user hovers and clicks the 'upload' button
3. The user surfs and finds the target file on his/her file system
4. If the limit does not exceed the users plan, the file will be uploaded to the system and will be available to edit

5.8 Save File



The purpose of this use case is to describe the procedure of saving a file in the system.

Pre-conditions: • User must have an account • User must be logged in to the system • Target file must be opened for editing

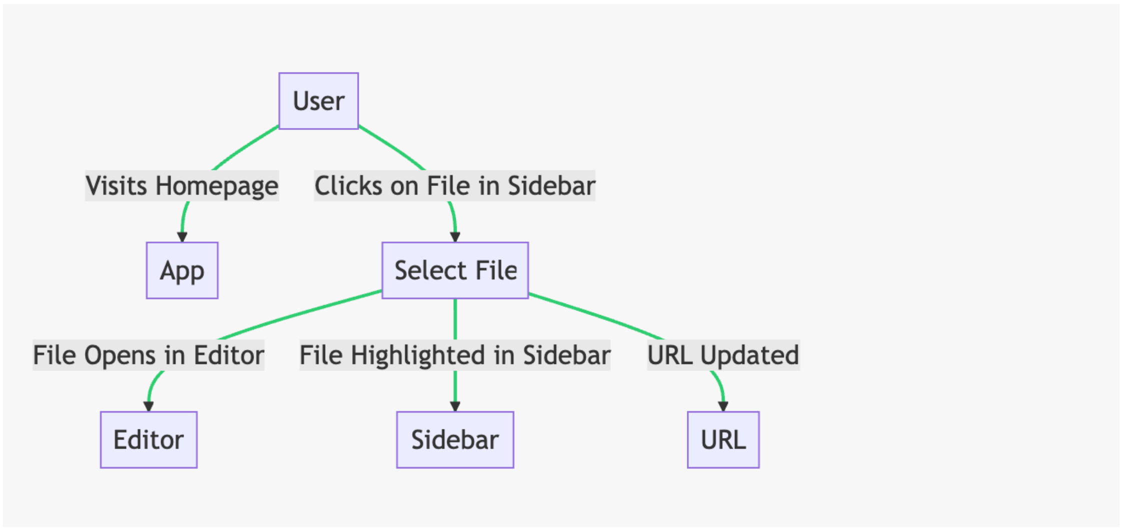
Post-conditions: • The changes to the target file will be saved

Basic Flow:

1. The user is in the editor with a file open
2. The user makes changes to the file content
3. The 'Save' button becomes enabled
4. The user clicks the 'Save' button
5. A loading indicator is displayed while the file is being saved
6. Once the file is saved, the 'Save' button is disabled again

This use case ensures that only logged-in users can save changes to a file, and that the 'Save' button is only active when there are changes to save. After the file is saved, the 'Save' button is disabled until more changes are made.

5.9 Select File



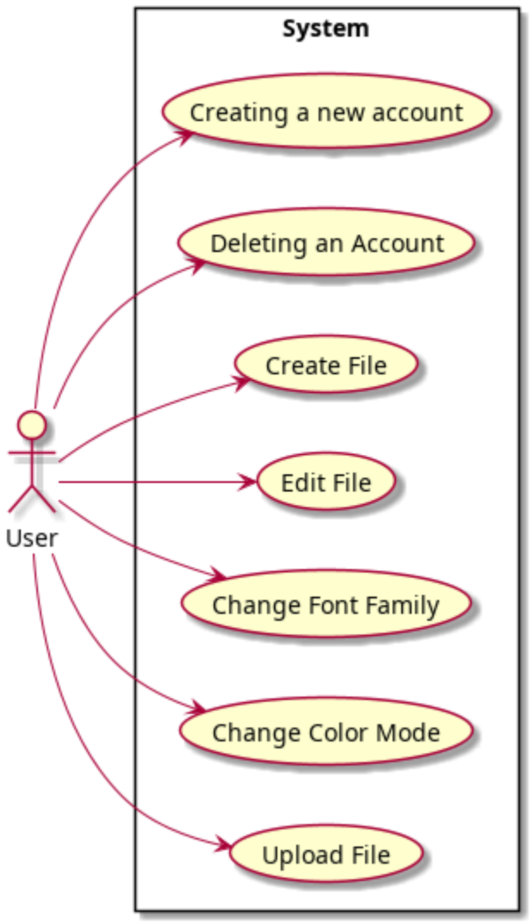
The purpose of this use case is to describe the procedure of selecting a file in the system.

Pre-conditions:
• User must have an account
• User must be logged in to the system

• Files must be available in the sidebar
Post-conditions:
• The selected file will be opened in the editor
• The URL will be updated to reflect the selected file
• The selected file will be highlighted in the sidebar
Basic Flow:

1. The user is on the homepage with the file manager accessible.
2. The user clicks on a file in the sidebar.
3. The selected file is highlighted in the sidebar and the URL is updated.
4. The content of the selected file is displayed in the editor for viewing or editing.

This use case ensures that only logged-in users can select files to view or edit. The selected file is highlighted in the sidebar and the URL is updated to reflect the selected file. The content of the selected file is displayed in the editor.



6 Nonfunctional System Requirements

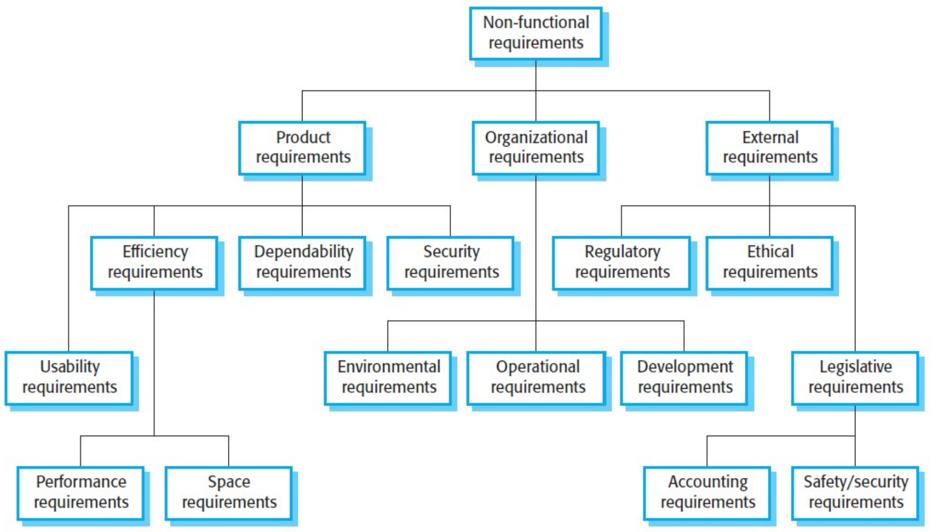


Figure 5 An example figure

6.1 Performance Requirements

Total project size is restricted up to at most 5 MB and downloading a project may depend on users network bandwidth but our application should satisfy rendering speed constraints after client downloaded project.

ID: NF1
TAG: File display time for application(both web and mobile)
GIST: The speed of the file content render
SCALE: Response time between file switching
METER: Switch time between two 1 MB text files
MUST: No more than 1 second 100% of the time
WISH: No more than 100 milliseconds 100% of the time

6.2 Safety Requirements

Connection loss may cause consistency problems between which data the user wants to store and which data actually stored on the server side. Since there won't be any client-side storage for our application, if the user exits the application before all the changes are saved, there will be uncommitted changes. In order to prevent this problem our application will warn the user if there is no saved data on the client side.

6.3 Security Requirements

In order for the user to access his/her data, he/she needs an access token. Our application will use Json-Web-Token to handle authorization. Also switching data between the client and server side will be HTTPS protocol which is encrypted with SSL.

6.4 Software Quality Attributes

Webitor will be responsive and quick to load, easy to use and intuitive, stable and reliable with no data loss, secure with strong user authentication and authorization controls, accessible to all users, easily deployable on web and mobile platforms, and maintainable with clear documentation and code organization. By prioritizing these attributes and designing the editor accordingly, developers can ensure that the final product meets the needs and expectations of its users.

6.5 Business Rules

The Webitor offers to its user to edit their text base files in a synchronized way stored on the cloud whichever platform(web or mobile) users desire. The main idea behind this product is both storing and editing text files safely on the cloud independent from the client's local computer.

7 Other Requirements

For now, all of them are covered in the document.

8 References

Object-Oriented Software Engineering, Using UML, Patterns, and Java, 2nd Edition, by Bernd Bruegge and Allen H. Dutoit, Prentice-Hall, 2004, ISBN: 0-13-047110-0.

https://en.wikipedia.org/wiki/Mediator_pattern

https://entityframework-extensions.net/efcore-inmemory-provider

https://react.dev/

https://tanstack.com/query/v4/docs/react/react-native

https://tanstack.com/query/latest

https://reactnativeelements.com https://reactnavigation.org

https://styled-components.com https://mui.com/

https://www.typescriptlang.org/ https://learn.microsoft.com/en-us/dotnet/csharp/ https://nextjs.org/

9 Appendix A: Glossary

- **Backend**: The server-side component of the Webitor application responsible for storing and retrieving user data and managing user authentication.
- **API**: Application Programming Interface. A set of protocols and tools used for building software applications, defining how different software components should interact.
- **Cloud-based**: Refers to software or services that are hosted on remote servers and accessed through the internet, rather than on a user's local computer.
- **Authentication**: The process of verifying a user's identity to ensure that they have the necessary permissions to access and manipulate their data within the Webitor application.
- **Synchronization**: The process of keeping the data in the cloud-based database and on the user's devices up-to-date and consistent, so that changes made on one device are reflected on all devices.
- **Frontend**: The client-side component of the Webitor application that runs on the user's device and provides the user interface for editing and managing their documents.
- **UI**: User Interface. The visual and interactive elements of the Webitor application that allow users to interact with and edit their documents.
- **CSS**: Cascading Style Sheets. A style sheet language used for describing the presentation of a document written in HTML or XML, such as the fonts, colors, and layout.
- **HTML**: Hypertext Markup Language. A markup language used for creating web pages and other information that can be displayed in a web browser.
- **Android**: A mobile operating system developed by Google, used primarily on smartphones and tablets.
- **iOS**: A mobile operating system developed by Apple, used primarily on iPhones, iPads, and iPod Touch devices.

10 Team Members, Roles, and CVs

Mustafa Arinmis, Backend Developer, https://arinmis.github.io/dotcs/Mustafa_Arinmis_CV.pdf

Muhammed Fatih Ozdil, Mobile Developer and Scrum Master, https://fatihozdil.xyz/fatih_ozdil_resume.pdf

Yusuf Ramazan Tanrıkulu, Frontend Developer, https://flowcv.com/resume/4n5o93jm1o