**COMP4920 Senior Design Project II, Spring 2023**

**Advisor: Prof. Dr. Mehmet Ufuk Çağlayan**

**WEBREM: A Remembrances or Condolences Web Service**

**Requirements Specifications Document**

**WEBREM**

**High Level Design**

**Design Specifications Document**

**Revision 2.0**

**19.03.2023**

**By:**

**Atakan Aktakka, 18070001032**

**Alican Daşdemir, 18070001041**

**Doğukan Tez, 18070001051**

**Kadir Ülge, 18070001014**

# Revision History

|  |  |  |
| --- | --- | --- |
| **Revision** | **Date** | **Explanation** |
| 1.0 | 08.12.2022 | Initial high-level design |
| 1.1 | 10.12.2022 | 3 Software Subsystem Design added |
| 1.2 | 11.12.2022 | 5 Testing Design and References added |
| 1.3 | 18.12.2022 | 4.1 and 4.2 sections updated |
| 1.4 | 20.12.2022 | Update: WEBREM software system structure |
| 1.5 | 20.12.2022 | Update: WEBREM software system environment |
| 1.6 | 20.12.2022 | Update: WEBREM software system detailed design |
| 2.0 | 19.03.2023 | Update: 1, 2 and 4.1 sections |

# Table of Contents

[Revision History 2](#_Toc130133211)

[Table of Contents 3](#_Toc130133212)

[Table of Figures 4](#_Toc130133213)

[1. Introduction 5](#_Toc130133214)

[2. WEBREM Software Design 5](#_Toc130133215)

[2.1. WEBREM Software System Architecture 5](#_Toc130133216)

[2.2. WEBREM Software System Structure 6](#_Toc130133217)

[2.2.1. Controller Layer 6](#_Toc130133218)

[2.2.2. Model Layer 6](#_Toc130133219)

[2.2.3. View Layer 7](#_Toc130133220)

[2.3. WEBREM Software System Environment 7](#_Toc130133221)

[3. WEBREM Software System Detailed Design: 9](#_Toc130133222)

[3.1. WEBREM Main Class 9](#_Toc130133223)

[3.2. WEBREM Subsystem Classes 9](#_Toc130133224)

[3.2.1. NonRegisteredUser Class 9](#_Toc130133225)

[3.2.2. RegisteredUser Class 9](#_Toc130133226)

[3.2.3. Moderator Class 11](#_Toc130133227)

[3.2.4. MasterModerator Class 11](#_Toc130133228)

[3.2.5. Admin Class 11](#_Toc130133229)

[3.2.6. MasterAdmin Class 11](#_Toc130133230)

[4. Testing Design 12](#_Toc130133231)

[4.1 Testing Strategy 12](#_Toc130133232)

[References 12](#_Toc130133233)

# Table of Figures

[Figure 1. WEBREM Software System Structure 6](#_Toc130133197)

[Figure 2. WEBREM Software System Environment 7](#_Toc130133198)

# 1. Introduction

The purpose of the software project is developing a web application named WEBREM in an object-oriented programming language NodeJS, in LINUX/Windows/MAC and MongoDB environment. WEBREM has some main functions which are listed below.

* Allowing users to upload memories about important people or institutions, in the form of text, photos, audio recordings, or video recordings.
* Storing these memories in a database, in a moderated way to ensure that only appropriate content is included.
* Providing a searchable index of the memories, allowing users to easily find and access the memories they are looking for.
* Providing a web-based user interface for accessing and interacting with the memories in the database.

The design is based on WEBREM Requirements Specification Document, Revision 1.5, in file WEBREM -RSD-Rev-1.5.doc [1].

The software architecture and overall high-level structure of WEBREM Service are given in Section 2, 3 and 4 and design details of all application functions and the user interface in terms of methods of all classes are given in Section 5 of this document.

# 2. WEBREM Software Design

# 2.1. WEBREM Software System Architecture

MVC architecture is the most suitable option for this system architecture. MVC (Model-View-Controller) architecture has several benefits, including separation of concerns, testability, reusability, maintainability, scalability, and flexibility. This architecture divides an application into three components, allowing for independent development and easier management of complex applications. MVC promotes code reusability, making it easier to build new applications or add new features to existing ones. Additionally, it allows developers to choose different technologies for each component, providing flexibility and scalability. Overall, MVC provides a structured approach to developing applications that is reliable, robust, and scalable.

# 2.2. WEBREM Software System Structure

diyagram içeren bir resim

Açıklama otomatik olarak oluşturuldu

Figure 1. WEBREM Software System Structure

WEBREM services use an MVC architecture that consists of three layers: the controller layer, model layer, and view layer.

# 2.2.1. Controller Layer

The controller layer is responsible for implementing the core functionality of the application and contains classes and methods that handle tasks such as creating and updating tasks, managing user accounts, and generating reports.

# 2.2.2. Model Layer

The model layer is responsible for managing data storage and retrieval. It uses a database to store information about tasks, users, and other aspects of the application. It provides the controller layer with methods for accessing and updating this data.

# 2.2.3. View Layer

The view layer defines the interface that the other layers in the system use to communicate with one another. It typically defines a set of functions, methods, or classes that the other layers can use to access the services provided by the layers below it. This layer acts as an interface between the different components of the system and plays a crucial role in ensuring that the different components can work together effectively.

# 2.3. WEBREM Software System Environment

A WEBREM service is designed to run on a browser that is running the Windows/linux/Mac operating system. The service is built using the ExpressJS, NodeJS and it uses a MongoDB database to store data.

The software system environment for this service includes Linux/windows/Mac operating system, ExpressJS framework, NodeJS and MongoDB database. It also includes all external services to which the service is connected, such as a database service for images, sound recordings, text messages etc. sent by the user, or a database service for downloading the desired post.

The user makes a request to the program by communicating from the view layer. The controller layer receives this appropriate response from the model layer after the necessary controls and transmits it to the view layer.

In this environment, the service must be able to function properly on the Windows/linux/Mac operating system, using the ExpressJS framework, NodeJS and the MongoDB database. It must also be able to interact with any external services that it depends on, such as the posting and downloading services.

To ensure that the web service functions properly in this environment, the developers may need to conduct tests to verify that the application works as intended on the Windows/linux/Mac operating system, with the ExpressJS framework, NodeJS and the MongoDB database. They may also need to test the service's interactions with any external services to ensure that they are working properly.

Overall, the software system environment is the set of external factors and conditions that can impact the functioning and performance of a software system. It is important to consider the environment when designing a software system to ensure that it can function properly in its intended environment.

**WEBREM Software System Back-End Environment**

* + - NodeJS 18.13.0v

**WEBREM Software System Front-End Environment**

* + - ReactJS 18
    - Html & CSS
    - Javascript

**WEBREM Software System Database Environment**

* + - MongoDB 6.04v

**WEBREM Software Testing Environment**

* + - JestJS 29.5.0v
    - Puppeteer

**WEBREM Software System Frameworks Environment**

* + - ExpressJS 4.18.2v

# 3. WEBREM Software System Detailed Design:

# 3.1. WEBREM Main Class

Main class is the main program or main process for developments in WEBREM.

# 3.2. WEBREM Subsystem Classes

# 3.2.1. NonRegisteredUser Class

**NonRegisteredUser Class - Method createAnAccount()**

* This method takes NonRegisteredUser class as an input parameter, then creates account for non-registered users.
* If non-registered user wants to create an account, the user will fill out a form containing the following information. Fields that are marked with a \* are mandatory (There should be a message about this to the user).
  + Email\*
  + Country\*
  + Password\*
  + Last name\*
  + Name\*
  + GSM Phone\*
  + Username

Sign-up Buttons: Submit and Cancel

**NonRegisteredUser Class - Method activateAccount()**

* This method takes NonRegisteredUser class as an input parameter, then sends an activation e-mail to non-registered user.
* After non-registered user creates an account, an e-mail is sent to the user to activate the account.

# 3.2.2. RegisteredUser Class

**RegisteredUser Class - Method makeQuery()**

* Registered user makes queries site by search bar and surfs within the site by this function.

**RegisteredUser Class - Method signIn()**

* This method takes RegisteredUser class as an input parameter, then logs user into the system.
* When registered user activates his/her account, this function can sign in him/her. User signs in by clicking sign in button.

**RegisteredUser Class - Method updateRegistrationInfo()**

* This method takes RegisteredUser class as an input parameter, then updates registered user’s information.
* The registered user will fill out a form containing the following information to update registration information.
  + Email\*
  + Password\*
  + Buttons: SHOW MY REGISTRATION INFO, CANCEL and I FORGOT MY PASSWORD

**RegisteredUser Class - Method shareMemories()**

* This method takes RegisteredUser and Memory classes as an input parameter, then shares memory.
* The user will fill out a form containing the following information to share memories. At least one field must be filled (There should be a message about this to the user).
  + Add Photo
  + Add Text (memories)
  + Audio recordings
  + Video recordings

**RegisteredUser Class - Method updateSharedMemories()**

* This method takes RegisteredUser and Memory classes as an input parameter, then updates shared memories.
* Registered WEBREM users will be able to update their memories on request by using the UPDATE button. WEBREM The registered user will be able to delete the shared memory during the update with the DELETE button. This action does not go through moderator approval.

The registered user will fill out a form containing the following information.

* + Photo/Text/Audio Recordings/Video Recordings\*

Buttons: UPDATE, DELETE and CANCEL

**RegisteredUser Class - Method downloadMemories()**

* This method takes Memory class as an input parameter, then downloads shared memories.
* The registered user can download the shared memories by pressing the download button whenever he/she wants.

**RegisteredUser Class - Method createMemorialPage()**

* This method takes RegisteredUser class as an input parameter, then creates memorial page.
* The user fills out an application form about the deceased famous person or institution that he/she wants to be opened, and the Master Moderator examines and approves this application.
* The user will fill out a form containing the following information. Fields that are marked with a \* are mandatory (There should be a message about this to the user).

(Their means here famous person or institutions).

For persons,

* + Their First Name\*
  + Their Last Name\*
  + Their Date of Birth (year – month - day) \*
  + Their Date of Death (year – month - day) \*
  + Their City of Birth\*
  + Gender (female – male – other) \*
  + Add Photo
  + Add Information\*

For institutions,

* + Their Name\*
  + Their Date of Establish (year – month - day) \*
  + Their Date of Bankruptcy (year – month - day) \*
  + Their City of Birth\*
  + Add Photo (logo)
  + Add Information\*

# 3.2.3. Moderator Class

**Moderator Class - Method report()**

* This method takes RegisteredUser and Memory classes as an input parameter, then reports users and memories to admin.
* Moderator reports to the admin about users who make inappropriate comments, upload images, and behave.

# 3.2.4. MasterModerator Class

**MasterModerator Class - Method assignModerator()**

* This method takes Moderator and MemorialPage classes as an input parameter, then assigns moderator to memorial pages.
* After the Master Moderator reviews and approves the application, one or more Moderators will be assigned for that person/institution page, and basic information about that person/institution will be displayed on the WEBREM service.

**MasterModerator Class - Method checkPageOpeningForm()**

* This method takes MemorialPage class as an input parameter, then brings page opening form to master moderator for checking.
* The user fills out an application form about the deceased famous person or institution that he/she wants to be opened, and the master moderator examines and approves this application.

# 3.2.5. Admin Class

**Admin Class - Method banUsers()**

* This method takes RegisteredUser class as an input parameter, then bans user.
* After evaluating the report about the user, it decides to ban or not.

# 3.2.6. MasterAdmin Class

**MasterAdmin Class - Method assignAdminMasterModerator()**

* This method takes Admin, MasterModerator and MemorialPage classes as an input parameter, then assigns admin and master moderator for memorial pages.
* Master admin can change admin and master moderator later.

These classes are written to give information about the general architecture and operation of the project. More detailed explanations will be added during the COMP4920 course.

# 4. Testing Design

The purpose of this testing design is to describe the approach and strategies that will be used to test the web-based task management services.

# 4.1 Testing Strategy

* Unit testing: Jest will be used to perform unit testing on individual components or classes in the service to verify their correct functioning in isolation. This will involve creating test cases that exercise the different methods and features of each component and verifying that they produce the expected results.
* Integration testing: Jest will be used to perform integration testing to verify that the different components of the service work together correctly. This will involve testing the interactions between the view layer, the model layer, and the controller layer.
* End-to-end testing: Jest and Puppeteer will be used to perform end-to-end testing to verify the overall functioning of the service as a whole. This will involve testing the service from the perspective of a user, simulating different user actions and verifying that the service responds as expected.

# References

1. System Requirements Specification Document, Revision 2.0, in file WEBREM-RSD-2022-12-19-Rev-2.1.doc.
2. IEEE Std 1016-2009