EECS 348 Group 10

Web Calculator for University of Kansas Students
Software Architecture Document

Version 1.0

Revision History

Date	Version	Description	Author
11/12/2023	1.0	Initial version	Harlan Williams

Table of Contents

Software Architecture Document

1. Introduction

1.1 Purpose

This document provides a representation of the major architectural design of the system as well as specific details of the architecture.

1.2 Scope

This document covers the design of the Web-based calculator app at a system level and class level.

1.3 Definitions, Acronyms, and Abbreviations

This document uses the same definitions, acronyms, and abbreviations described in the Project Plan.

1.4 References

This document references the Project Plan for definitions, acronyms, and abbreviations, the Software Requirements Specification for the functionalities and constraints of the software to be developed, and the UML class diagrams in the project documentation folder.

1.5 Overview

This document details how the architecture of the system is represented, the system requirements and constraints that informed the architectural decisions, the specific details of the system architecture, including its major subsystems and packages, and a description of how the architecture contributes to the constraints and non-functional requirements of the system.

2. Architectural Representation

The architecture is represented using UML class diagrams to represent the modules of the architecture and the connections between them.

3. Architectural Goals and Constraints

The web-app will be constructed in a layered design, consisting of an App class that routes HTTP requests to endpoints that are then handled by the Services layer beneath the App layer. Beneath the Services layer is the Database Access Object layer, which provides a standardized interface for Services to interact with the web-app's User, History, and Script databases.

The calculator module will consist of three classes to perform the data manipulation necessary to achieve the main functionality. These three classes will be an interface class, a tokenizer class, and an evaluator class. This architecture creates separation between the functionalities of the calculator processes, which allows for modifications to individual processes without affecting other processes.

4. Logical View

4.1 Overview

The system consists of several services and classes in a layered model. Specifically, these services are a UI service, a History Service, a LogIn service, a Logger class, the Calculator service, a Scripting service, and an AccountManagement service.

4.2 Architecturally Significant Design Modules or Packages

The significant design objects are represented in the class diagrams located in the project documentation folder.

5. Interface Description

There will be a web interface consisting of endpoints where users can access the calculator to perform operations, write scripts, evaluate scripts, and access their account.

In the calculator endpoint, the user will be able to enter an expression in a text box or form and submit their input. The app will then check the input for errors and return either an error or the result of the evaluated expression. There will also be an input for a user to view their history while they are on the calculator endpoint.

In the script writing endpoint, a user will be able to input a list of expressions possibly containing variables and assignment of return values into a text box. Once complete, a user will be able to save and share their script. Each script will receive an identifier once it is saved so that it can be shared. The original creator of the script will be stored as the owner and only that user will be able to edit the script.

In the script evaluation endpoint, a user will be able to assign values to variables inside the script in the form of an input box, and then the user will be able to submit their input and one or more return values will be set and returned to the user.

In the user account endpoint, users will be able to change their email, password, view their history, and view scripts they have written.

6. Quality

The layered design of the web application makes it easier to implement additional services without affecting much code above and below the services layer. The abstraction of accessing a database with database access objects makes it easier to change databases in the future.