Linnæus University
Sweden

Design Document

# Project course in software development

*Authors*: *Student 1,*
*Student 2,*
*Student 3,*
*Student 4,*
*Student 5*
*Semester*: *VT20*
*Course code*: *2DV609*

# Table of Contents

# Purpose

In this section, the purpose of the design document will be described. The system that will be designed will be divided into smaller system parts which will be described shortly what each system part will do. First and foremost the system will be divided into Frontend and Backend then the related system parts will be listed below each of them. After that the system parts will be shortly described and what their job in the system will be.

## Frontend

- User profile
- Habit
- Category

The frontend includes the domain components user profile, habit and category. The reasoning behind showing these components are because they are components that will contain the main system functionalities. To get a better understanding of what role each component will have in the system we will go through each one a bit more in detail. The user profile which handles the user's details and lets them edit some of them if they want to. Habit handles the created habits that will be scheduled in the application and added into a category. It will also handle if the habit isn't completed if that occurs an email will be sent to the user to remind them they have a habit to complete. Categories are created by the user to store their habits so that the user can easily access the habits.

## Backend

- Database
- Authentication

The backend includes the domain components database and authentication. We chose to go with these two since they made the most sense and contain most of the backend functionality of our system. So, first of all, we have the database which handles all the personal details from the user, unique id, and password which is encrypted. The authentication part works parallel with the database to make sure that the user gets signed in to their account. When signed in, the authentication uses the unique id tags in the database and compares that to the id tag of the person that signed in to make sure that the right personal information is presented.

# Reference to requirements

This section will describe what requirements will be related to which system part and then below each system part there will be a short explanation of how they are needed.

## User profile

- USERPRO - A user has a personal profile page.
- EDITPD - A user should be able to edit their personal details.
- VIEWPD - A user should be able to view their personal details
- PROGRESS - A user should be able to see their progression through a progression bar after completing habits.
- IDTAG - A user should have a unique id tag.
- AUTH-USER - A user should be the only one that can access their profile.

To have a user profile, a user needs to have a personal profile page where they can view their personal details while also editing some of them. The users should also be able to see their progress through a progression bar that fills up whenever a user completes a habit. To make sure that no one else can see the personal details every user will get a unique id tag. When a user logs in their unique id will not allow other users to access their user profile.

## Habit

- HABDESC - A user shall be able to create an object that describes a repetitive behavior.
- UNIQHABIT - Users shall be able to create unique habits by describing a habit and time to do the habit.
- ADDHAB - A user shall be able to add a habit to a category.
- CREATHAB - A user shall be able to create a habit.
- CATHAB - A user shall be able to categorize their habits.
- CATEG.HAB - A user can specify the difficulty of the habit.

Every user should be able to create their own unique habits and schedule them in a timeline when they want to do them. The habits should be categorized to make them easier to find. Whenever they create a habit they can also choose what difficulty it should have, based on the difficulty the user will gain more progress in their progression bar.

## Category

- ADDHAB - A user shall be able to add a habit to a category.
- CATHAB - A user shall be able to categorize their habits.

A user should be able to create categories and then put the habits to the category it belongs to.

## Database

The database will include all data on the users such as their personal details, unique id tag. The database does not have any specific requirement tied to it.

## Authentication

- IDTAG - A user shall have a unique id tag.
- AUTH-USER - A user shall be the only one that can access their personal profile.

The authentication will work with the database system part to get the user's password, email, and unique id tag. This makes sure that the users get their personal accounts which include, for example, their personal user profiles.

## Email notifier

- EMAILREM - The application shall send an email at most 5 minutes after failing to do a session of habit.

Works with the habit system part, this system part will send out emails to the users to remind them if they have missed a habit. The email shall be sent at the latest 5 minutes after missing a habit.

# Priorities

This section will discuss what priorities will be focused on in this design document and why they are important. The priorities for the system will be security, simplicity, server stability, and reusability.

## Security

Security will be a priority because we need to make sure that users can only access their own accounts and user profiles. The user profile will contain some personal information about the users so we need to make sure no one has access to it besides the user.

## Simplicity

Another priority will be simplicity, we want to create a web application that is easy for people to use. We believe that people that are not that adverse with computers should be able to use this application so we need to make the navigation as clear as possible. A way to achieve this would be to only have one navigation menu and have a few web pages as possible.

## Reliability

The third priority will be to have good reliability in the server so that the web application rarely goes down and can host a good amount of users. The Firebase hosting feature will be used since it has been used in a previous project and it had good reliability while also being a good database.

## Reusability

The final priority will be reusability, we want to create an application where the code can be reused for future projects if needed. A good effect of reusable code will be that it will lower the complexity, increase cohesion, and lower coupling. These effects will lead to an easier time for multiple people working on an application since it will be easier to understand the code. This, in turn, will increase the efficiency, which leads to a faster conclusion of the project and a better product.

# High-level description

In this section the architecture for the application will be presented and why it was chosen over other architecture patterns.

## Client-server pattern

The architecture pattern our group chose for the application was client-server. The basis of this was that we felt like it made sense to have a pattern that works with the client and server when we are creating a web-based application. Another architectural pattern that was discussed was the layered pattern, it was also a good option. The major decider between the two patterns was that we as a team had more familiarity with the client-server architecture. We will use the Firebase Hosting as the server and then connect our client to it.

## Flux

Implementing Flux software pattern is highly compatible with Reacts framework. The point of doing this will be to add some structure for the coding, increase the cohesion, and decrease the coupling. The reason we chose Flux was that it is the most compatible software pattern to develop React applications. We wanted to decrease the risk of unnecessary confusion and troubles that can happen in new patterns and architectures, that's why we choose Client-server architecture pattern and Flux.

# Design Issues

To develop an application, we have to use different tools. When choosing what software architectural pattern, design pattern, language, framework, back-end platform, etc, we considered many solutions. Such as simplicity, flexibility, reusability, and finally efficiency. Below are listed the process and the design issues we encountered before the final decision is made.

## Software architectural pattern and Design pattern

When we wanted to implement a design pattern into our application, we figured that we should use one that everyone had some experience with, so we decided initially to use Client-Server architecture for the software architectural pattern and Mode, View, Controller (MVC) as the design pattern. What we did not realize initially were how badly the MVC pattern would fit into our web-based application that uses React. We noticed that we could not really make classes that would fit the rules for the packages. We thought about using Multi-Layer Architecture (MLA), but we were afraid that we would run into similar problems we had with MVC. The issue we had with React is that in every class we need to render which basically is some kind of output on the webpage, this makes React inapplicable for both patterns MVC and MLA. So we decided to change to Flux which is the recommended pattern when using React.

## Habits and categories

Something we thought would be difficult before starting with the coding phase of the project was how we would implement the habits and categories to a user. We were unsure how we would store the data since every habit and category a user creates should be their own and should only be accessed by them, we thought of different ways to do it, but decided, in the end, we would try to tie the habits and categories together with a users id tag. This way only a certain user could access their own habits and categories.

## Android or web application

One of the first issues we had was the decision of how we should design our application. The choices were between android application and web application. We had a long discussion about what to choose. Before we made our decision, we went through many phases of selection.

The first phase was our preferences. We had two alternatives. All of us wanted to develop an application with Android. There was a genuine interest in learning mobile-application development. Although we had the interest to learn app development, we soon recognized that we did not have time on our side. Since it required a lot of time to learn, it would not be efficient. We already had a familiarity with web-based applications, so we went with the more convenient option. The best thing for us to do was to go for the knowledge we already have. If we had more time we would choose to develop an android application.

The second phase was recourse availability. Guidelines and assistance are more available if we develop web-based applications. It was also recommended by the teacher if we use to develop the application with a web-based.

## Web framework

The web framework we chose was React.js. Since all of us are familiar with Angular, we had many discussions about what kind of web framework we want to use. Even though we had familiarity with Angular, we chose React.js. The reasoning behind the decision was because we had more available tools to use. We went back and forth between React and Angular. The decision was made on the usability and the flexibility of tools in React.js.

## Database

We chose to use Firebase as our backend database platform. The only form of backend database handling we are familiar with is Firebase. We appreciated the simplicity and the integration Firebase has with Google Sign-in. The other alternative for our backend database handling was through SQL. Again, lack of knowledge about the language made us choose Firebase. Our goal is to develop an application through simple means and Firebase enables us to focus on the frontend of the application while Firebase handles all the back-end operations.

## Programming language

The main programming language we chose to be TypeScript. There are other alternatives to script languages we could choose such as Ruby. We choose TypeScript due to its integration with React and also due to the knowledge we already acquired in previous courses. The final decision was made fairly quickly due to our choice of the web framework, which was React.js. We didn't have other alternatives to choose from since React.js is a JavaScript library.

## Authentication system

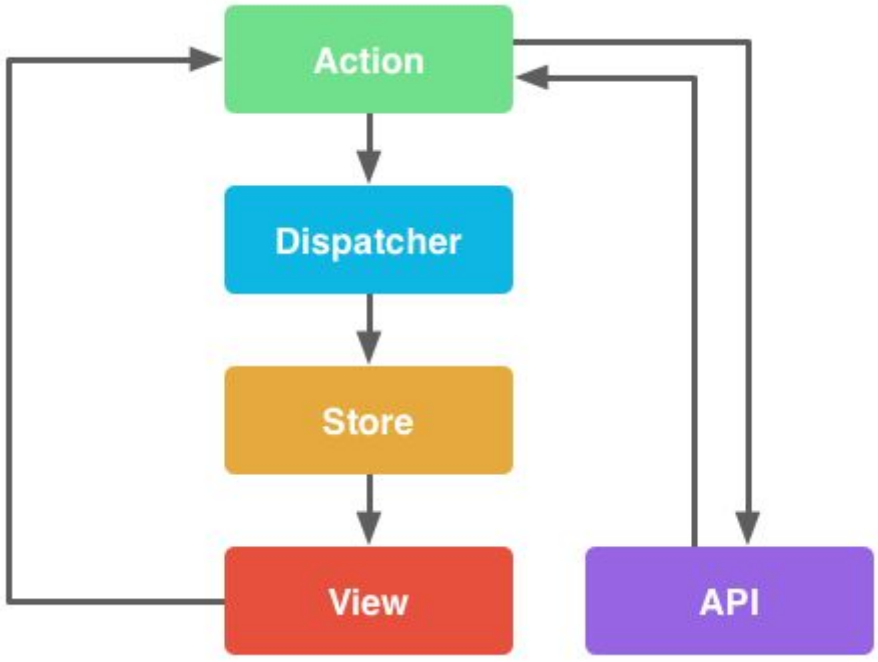
The authentication system we will use is Google Sign-in. The other alternative we had was the general email sign-in. We realized that using Google Sign-in is far more efficient and simple, especially when we already using the Firebase web platform. We also recognized that many users have Google accounts and it increases the steps of creating an account. We want the user to have a simple login experience. Having Google Sign-in allows the other to skip the process of creating an account. The disadvantage is that not all users have Google accounts. In that sense, the flexibility of the sign-in component is restricted to one type of login. The final decision was made due to Google Sign-in integration with Firebase, also the simplicity of handling only one type of login component.

# Software Architecture

## Reference architecture pattern

The following components are designed using Client-server based architectural patterns. Since we are hosting a series of users, we have chosen a client- server-based pattern.

The model is Flux pattern and the components are divided into their appropriate categories in regard to their connections to each other. All of these components are linked to each other. The picture below shows the process of  a Flux pattern.




*(taken from https://scotch.io/tutorials/getting-to-know-flux-the-react-js-architecture)*

Here is the responsibility of each component:

Actions: Different functions that help transfer data to Dispatcher
Dispatcher: Takes the data from Action and stores it into callbacks
Store: Manages application state and handles the logic.
Controller Views: Grabs the states from Stores and moves it into different components within Controller View.

## Components

In this section we describe the different components in our system and how they are tied together using the FLUX Design Pattern in the section “UML Component Diagram of the Architecture”.

Firebase

| Component | Description |
|---|---|
| Firebase | This component is responsible for initializing Firebase components and firebase authentication. |
| Context | Initialize the FirebaseContext component. |

Model

| Component | Comment |
|---|---|
| Difficulty | This component represents a list of difficulties and a label. |
| CreateHabit | This component is responsible for creating a new habit and sending data to the database. |
| CreateCategory | This component is responsible for creating a new category and sending data to the database. |
| Auth | This is the authentication component of the system. It controls the input entries of a system user with their save credentials. |
| Login | This component is responsible for validating the login credentials of the system user. |
| Navbar | This component contains the navigation bar of the system. |
| PrivateRoute | This component controls the visible components through authentication. |
| UserProfile | This component gives the User their own personal profile with data from the database. |

View

| Component | Comment |
|---|---|
| UserProfileView | This component gives the ability to manage a system user profile with no admin rights. |
| AboutView | This component gives a system the ability to view the about the company team page. |
| HomeView | The home view of the page with information about the content which can be found on the website. |
| ContactView | information about how to reach the company and support team. |

## UML Component Diagram of the Architecture

In this section of the document, we show the planned implementation of components in the system. The diagram displays the interaction between these components and illustrates how the system functions visually.

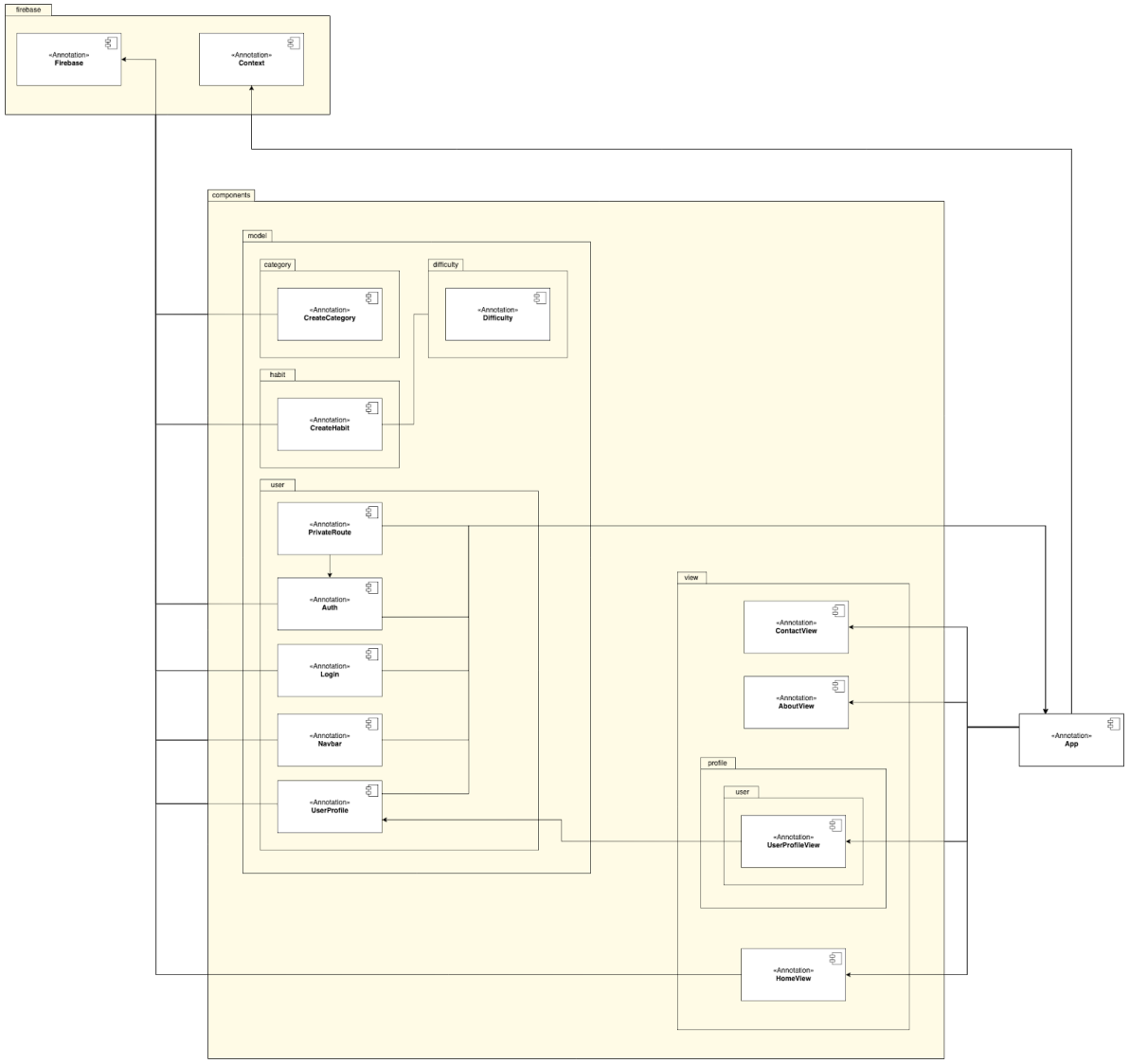


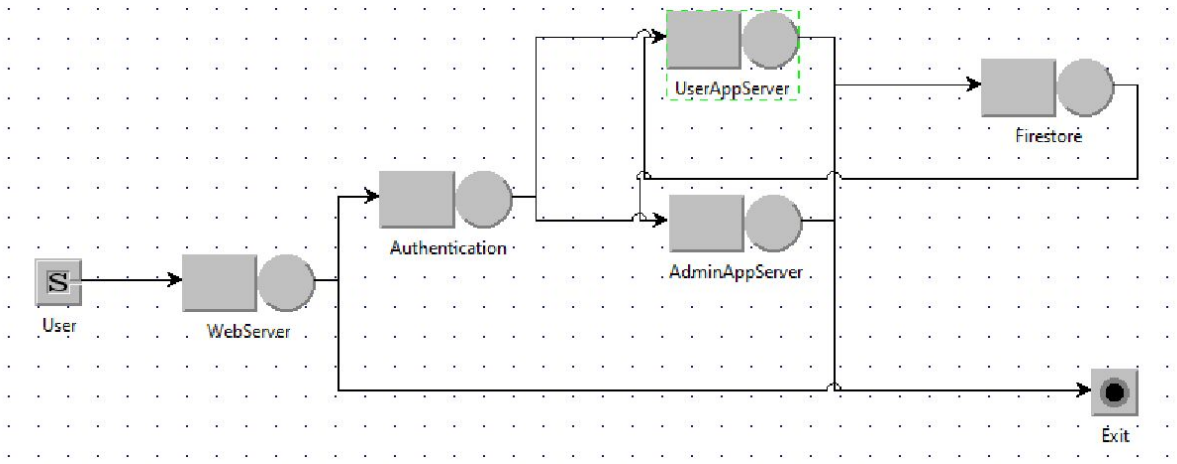

Figure 1: UML Component Diagram of the System Architecture.

# Performance modelling and evaluation

In this section, we will use JMT to model and to simulate the response time, throughput and utilization. The values that are input in the service time, probabilities are estimated and are the same as the values presented in the MARTE section in D1. The systems model is also based on the activity diagram from the D1 document.

## Model

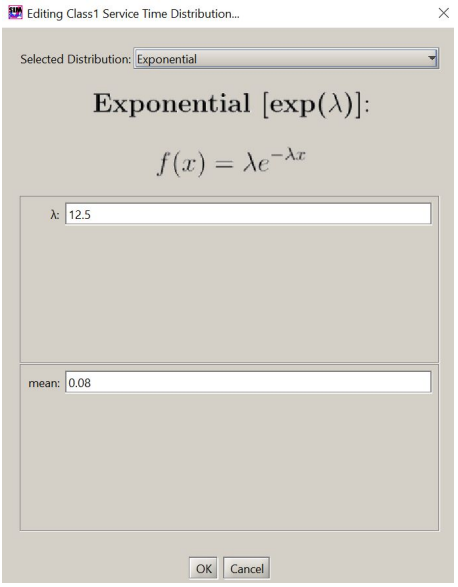
As stated before the model is based on the activity diagram from document D1 in the section Performance Requirement Modelling with MARTE.



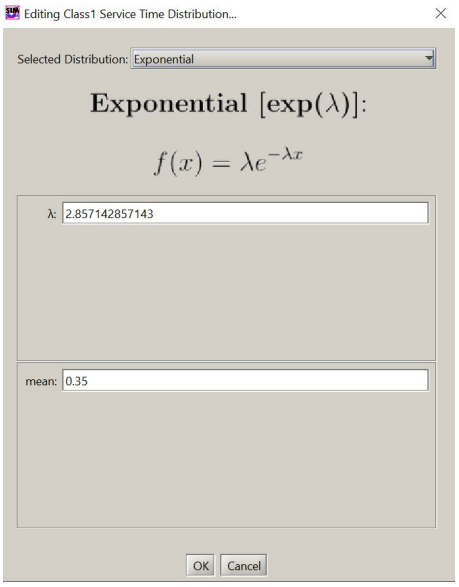
## Service time

The service times are based on the values that were estimated in D1 section Performance Requirement Modelling with MARTE.

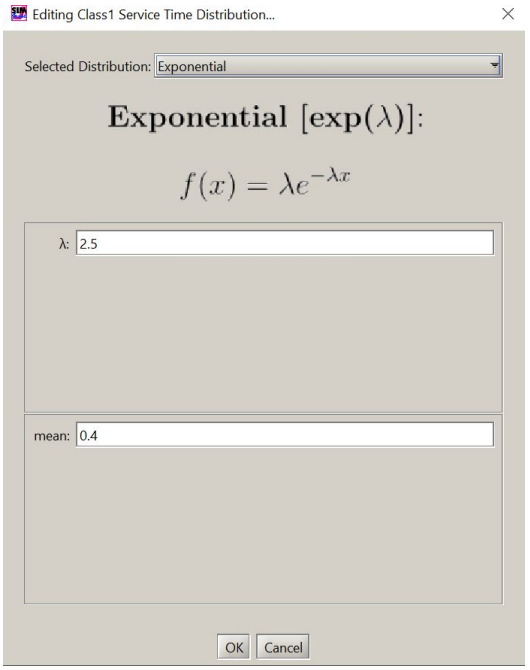
### WebServer



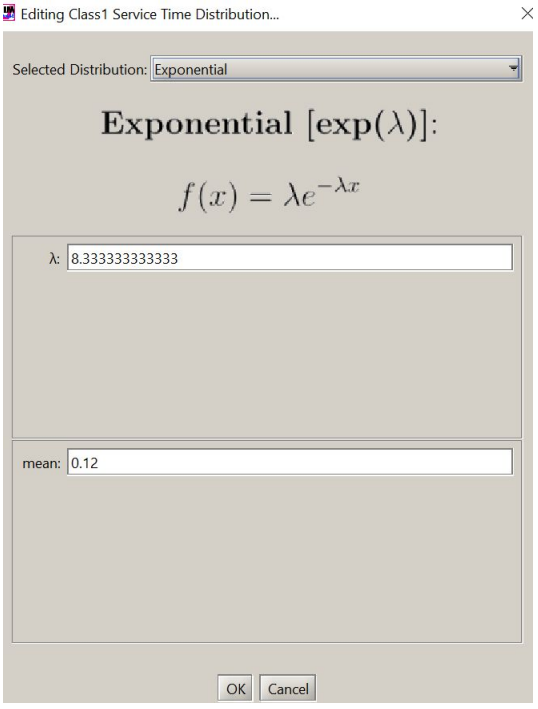

### Authentication



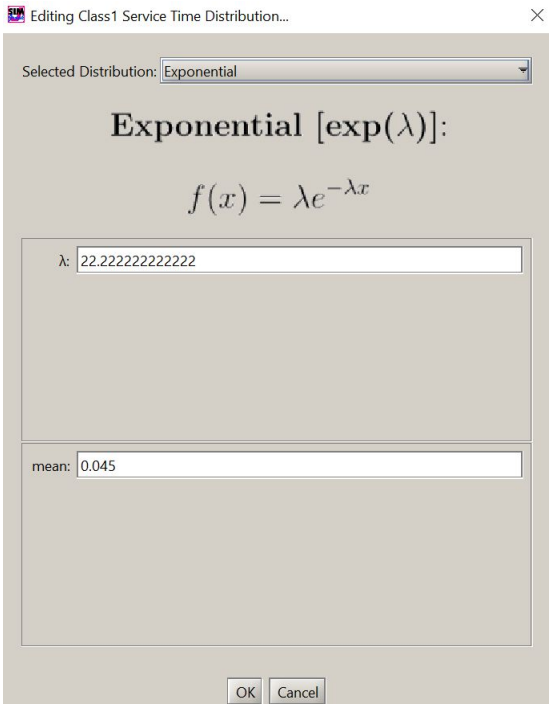
UserAppServer




AdminAppServer



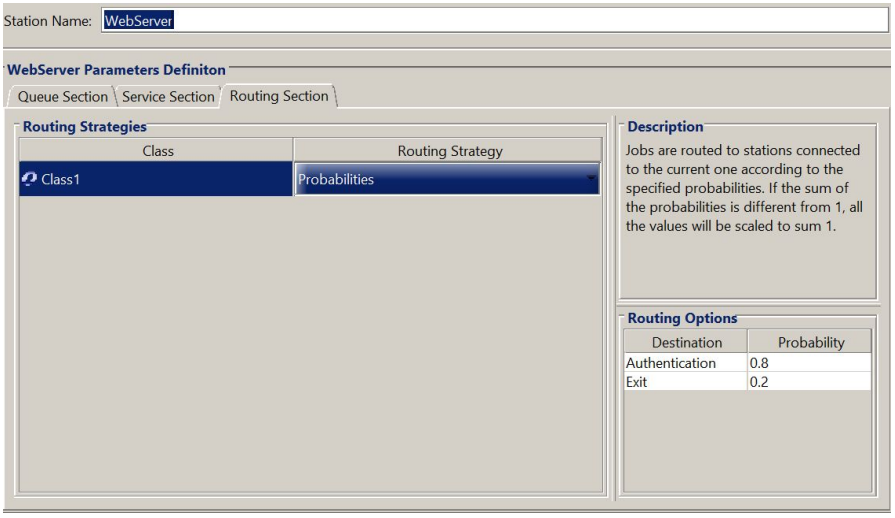
Database


Editing Class1 Service Time Distribution...
Selected Distribution: Exponential
Exponential [exp(λ)]:
$f(x) = \lambda e^{-\lambda x}$
λ: 22.222222222222
mean: 0.045
OK
Cancel

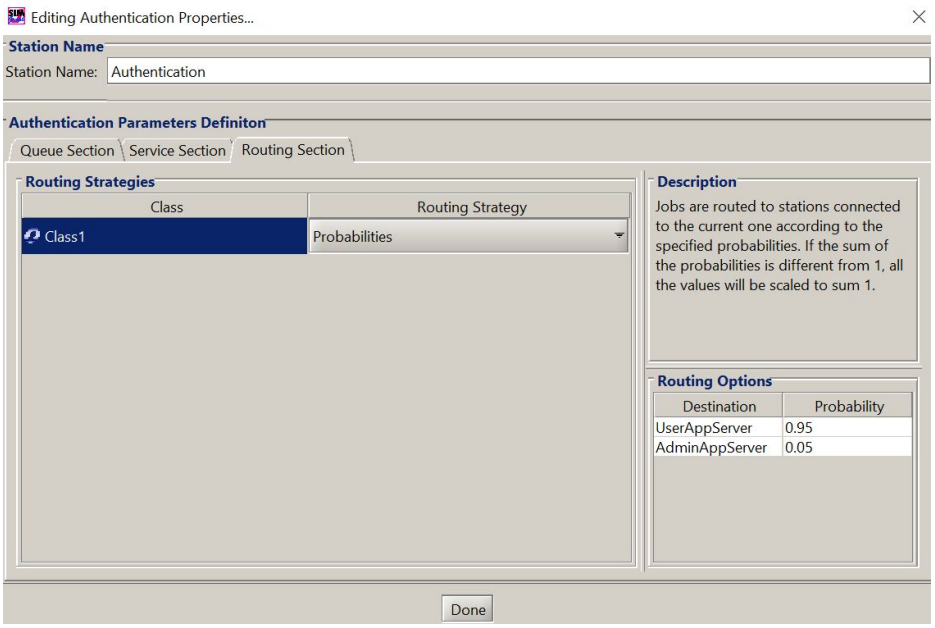
## Probability

The probability is based on the percentages that were estimated in document D1 in section Performance Requirement Modelling with MARTE.

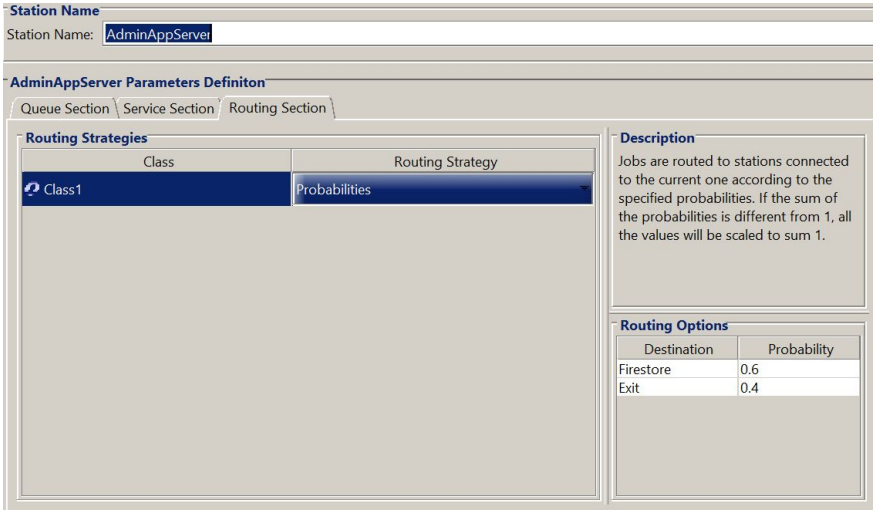
### WebServer



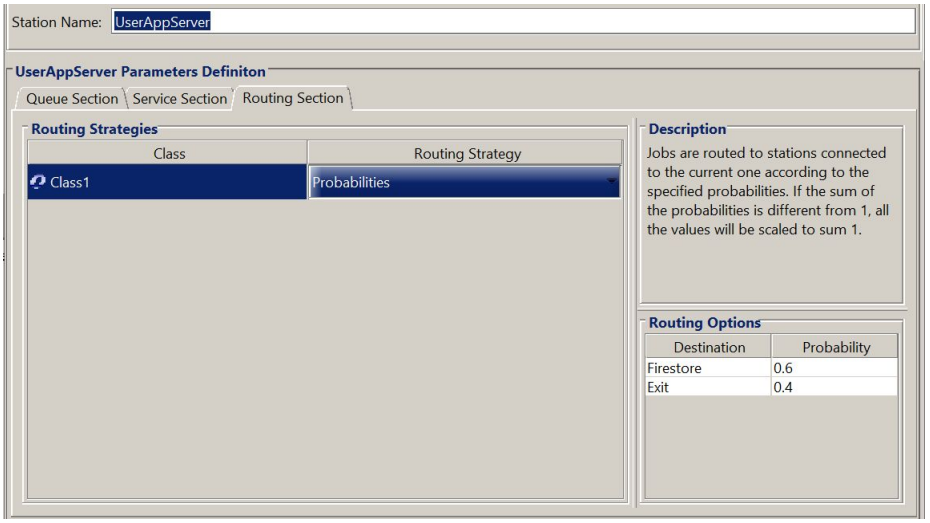
Authentication




UserAppServer

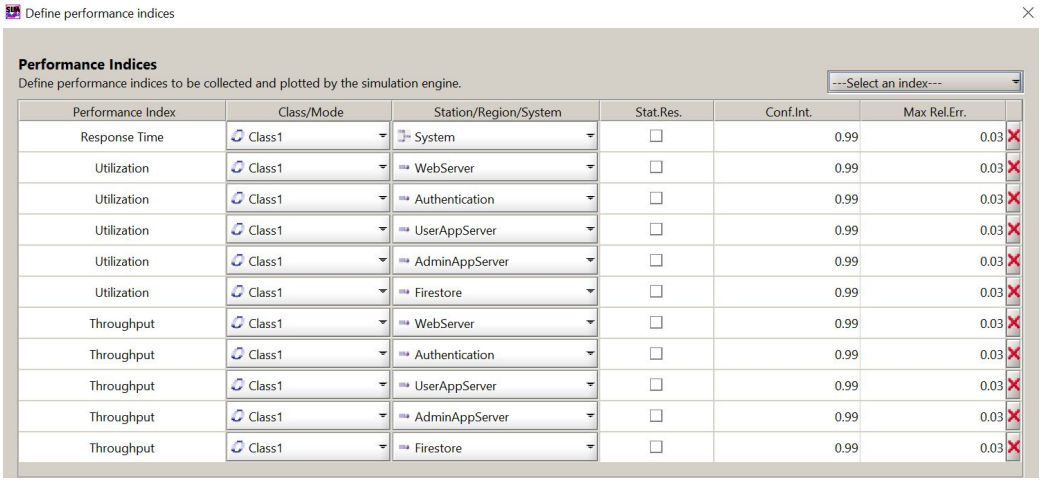


AdminAppServer



## Performance indices

The values that are interesting for our system is the response time which means the total time it takes for a system to respond to a request for service. Utilization which is how much time in which a part of the system is used and then finally throughput which is the rate of production or what rate a service can be processed. So for the response time, we want to know the full system's total time to respond and for utilization and throughput, we want to know each part of the system's output.

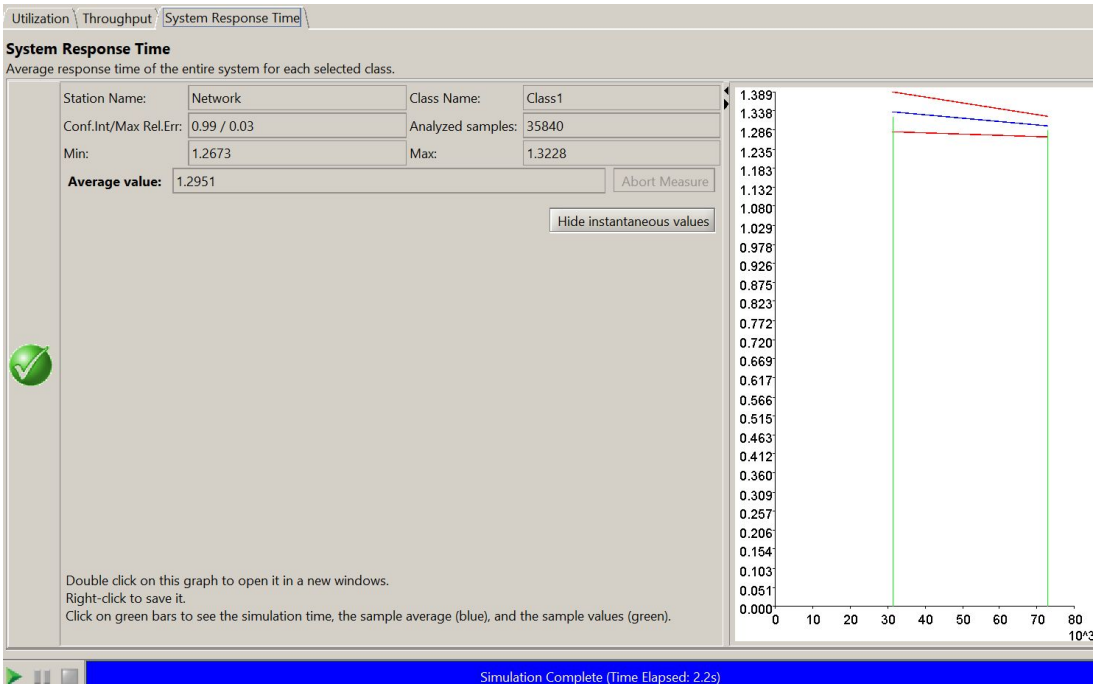


## Simulation

For the simulation, we wanted to see three different outputs, the system's response time, the individual parts of the system's throughput and utilization.

## Response time

In the response time, we have a somewhat slow average response time, over one second to respond to a request is not really acceptable in modern times systems.

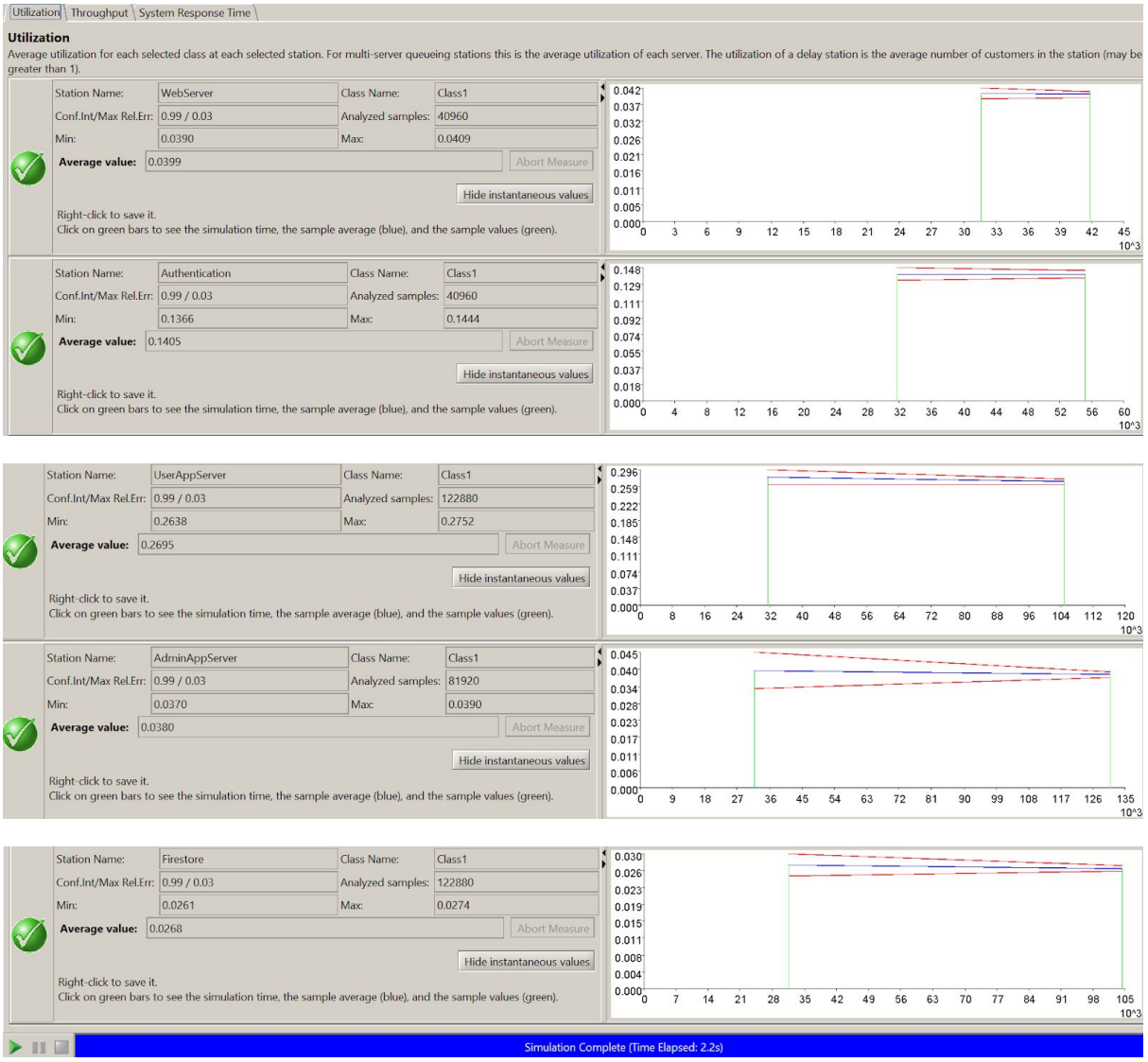


## Throughput

Every part of the system has approximately a production rate of 0.3-0.7 request/second which is all right, but a more accepted production rate would be over 1 request/second.

## Utilization

From the looks of it, the part of the system that is used the most is the Authentication and the UserAppServer. This was to be expected since those are the parts of the system that are most likely to have the user visit them according to the probability we have estimated in document D1.



# Component Implementation

## Design principles

In this section of the document, we analyze the components and address if any design principles will be added. The reasoning behind this is to make the design of the application simpler by dividing bigger parts into smaller subsystems and make sure to use cohesion when possible to keep things together that are related in the system.

Design principles table

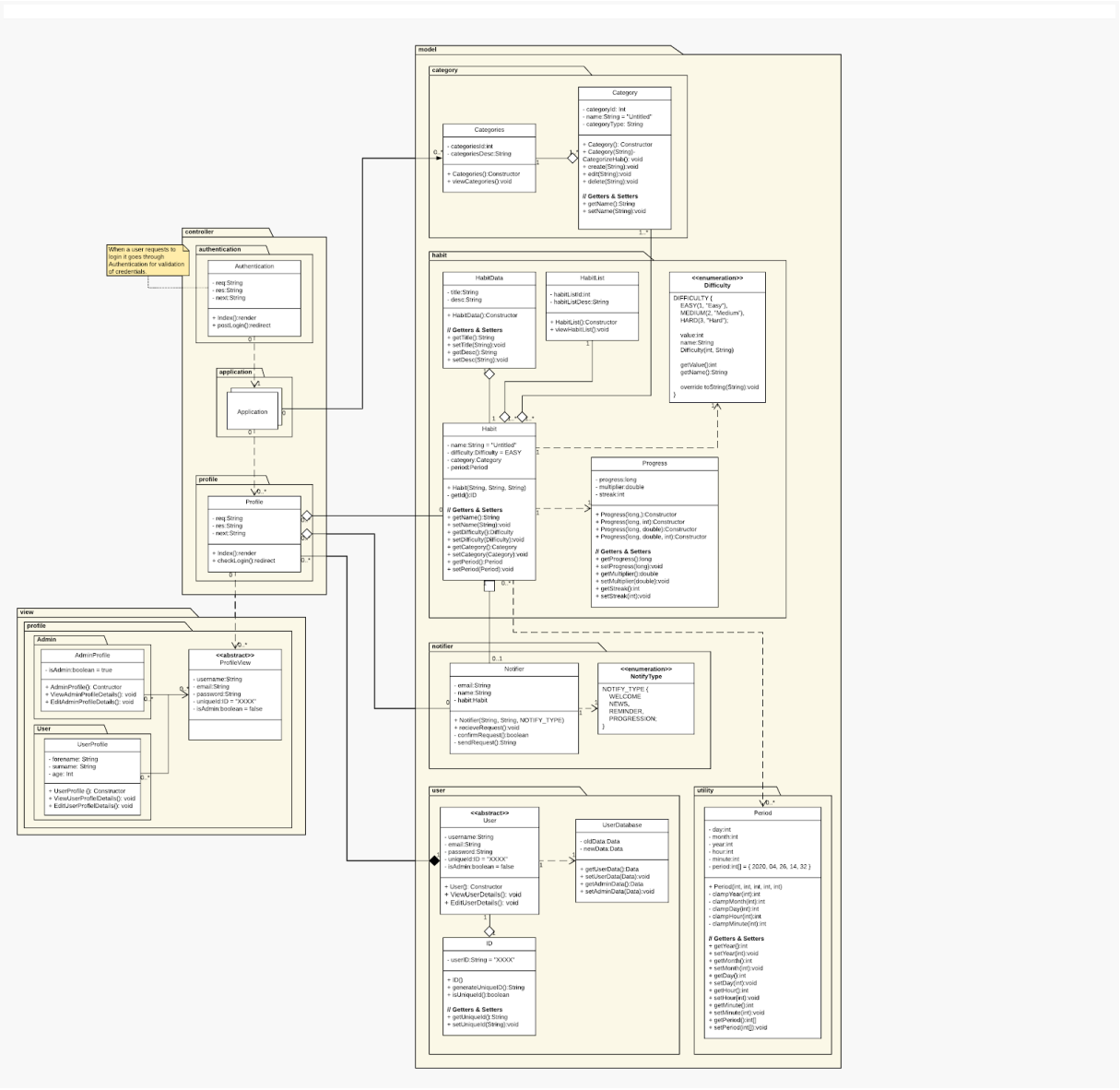
| Component | Principle type | Comment |
|---|---|---|
| UserProfile | Abstract | A profile will make use of an abstract class called ProfileView. |
| AdminProfile | Abstract | UserProfile will make use of an abstract class called ProfileView. |
| User | Abstract | Users will make use of an abstract class called User. |

Design principles addressed

We have chosen to go with an abstract approach for the classes in the design principles table to remove any unnecessary dependency issues that might occur later on in the project. Our goal is to have a code where the layers can be independently designed and therefore modified at any point without having to change the entire structure of the code.

# UML Class Diagram

In this section of the document, we have made a UML class diagram over the intended implementation of the components that are being discussed earlier in the document.




Due to time-constraints it has not been updated according to the pace of the project and is now deprecated. However the main takeaway of the system is still represented somewhat accurately and can be used as a reference.