GreetNGroup

High Level Design Document

CECS 491A Sec 05 8332

Team Gucci
Dylan Chhin 014430570
Eric Lee 014303261
Jonalyn Razon 014580772
Winn Moo 014633357 (Team Leader)
11/01/18

Contents

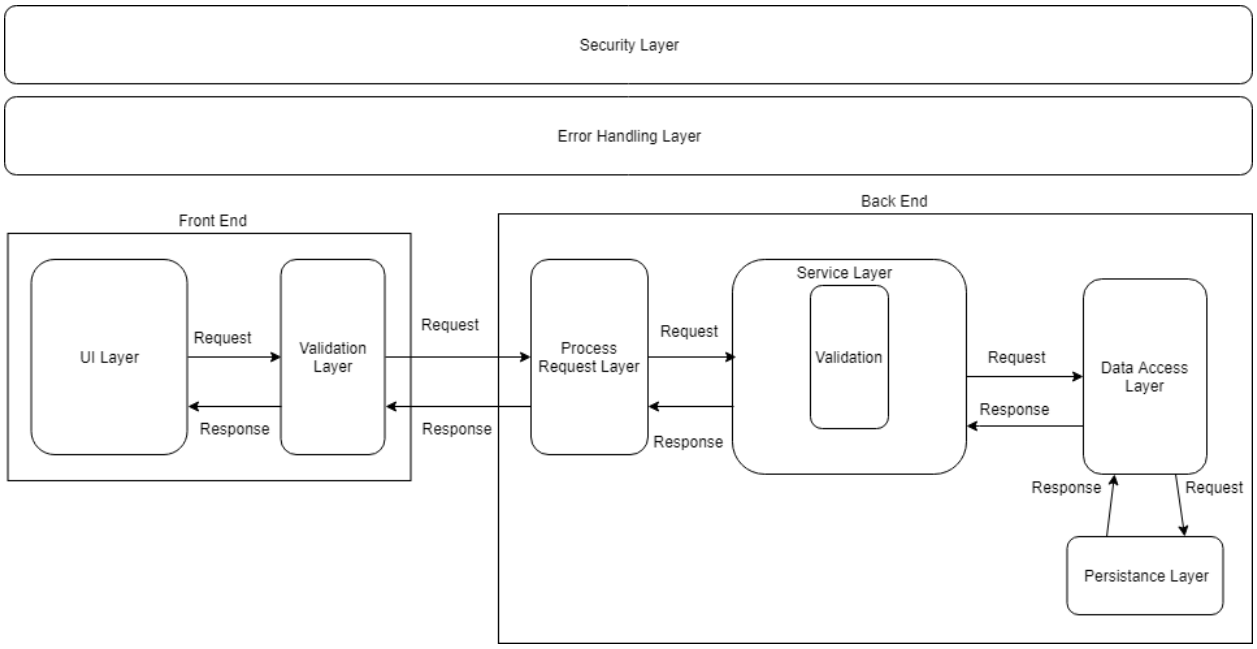
Introduction

This document describes the different layers needed for GreetNGroup to function. GreetNGroup is a Single Page Application (SPA) that has an initial page load with new content dynamically loaded onto the page. This web application will consist of the following:

- Front End
    - User Interface (UI)
    - Client Side Validation
- Back End
    - Process Request Layer
    - Service Layer
    - Data Access Layer
    - Persistence Layer
- Security
    - Encompasses the entire system
- Error Handling
    - Encompasses the entire system

Abstraction Layers

This diagram is the backbone of the website as the majority of the interactions with this website will follow this diagram. The only differences that will occur when using this diagram is the type of security measures that will be used when requesting different data for different purposes.



To prevent unnecessary entries to the server, which can slow down the system, validation must occur on both the client side and the server side.

Security Layer

The security umbrellas all of the other components of the system and is apart of each operation the system makes. Security is a major part of the system because if the security is not done correctly, then the other components will not be able to function properly. Security takes shape into two forms, authorization and authentication.

- Authentication is checking who the user is and if the user is actually who they claim to be
- Authorization is checking what the user is allowed to do and what they have access to

Error Handling Layer

The Error Handling Layer umbrellas all other layers because each layer has the possibility to cause an error. The Error Handling Layer figures out what the error is, where it happened, and what actions to take to resolve it.

UI

Interaction with the application is done through the UI layer. Provided a display and recognizable interactable objects, users can input into the application. Any information returned will be outputted through the UI.

Validation Front End

Inputs given through the UI are first sent to the front end validation layer, to check whether the information they are sending is in the correct format or not. If it is, then the data will be sent to the process request layer. Otherwise, it will not be processed and will return an error to the user, so they can re enter the information again.

Process Request Layer

The entry point of the website that performs the following:
Takes the data from the web server and/or web form and translates it into a readable statement that can be acted upon or used to perform functions internally on the server.

Service Layer

After the data is processed, it is sent to the service layer to to get authorized and validated to ensure whether or not the user is to be allowed access to information or if what they are doing is a valid action. If a user is not authorized to access certain content on the website, the session will be recorded (including the different elements the users has interacted with), and the user will be sent back to the homepage.

Data Access Layer

- The purpose of the data access layer is to:
    - Translate requests into a readable statement for the persistence layer to fetch or store data
    - Translate data into a readable format
- Allows for flexibility in persistence layer
    - Only the data access layer needs to be changed if changes in the persistence is made rather than the entire system needing to change

Persistence Layer

- The purpose of the persistence layer is to have a place to hold data on the website that can be stored or accessed by users
- Persistence layer should not be directly accessed by users and should only be accessed by the data access layer to store or retrieve data
- Persistence layer format should be able to change without affecting the system
- Data is received here and is sent back through all the layers to return to the user
    - If no results are found, when a user is accessing the database, then user is notified that the system was unable to find what they were looking for

System Architecture

SOA	MVC	MVVM
Pros: ● Does not need to keep track of what state the system is in, making the system scalable ● Services can be used in multiple systems ● Makes use of every reference in the system	**Pros:** ● Allows for significant control on how the system should behave ● Any changes to the Persistence Layer only requires changes in the Model	**Pros:** ● Data binding ensures that any dynamic changes to variables in the View Model is also updated in the View ● Able to test functionalities meant for the View in the View Model
Cons: ● Validation of each request takes longer to process and increases response time of the system ● Implementing SOA increases complexity, resulting in more resources needed	**Cons:** ● “Model” barely or not used at all, resulting in unnecessary references in the system ● Architecture requires a significant amount of resources on the working environment	**Cons:** ● “Model” barely or not used at all, resulting in unnecessary references in the system ● It will be difficult to debug the system as data binding errors are not caught

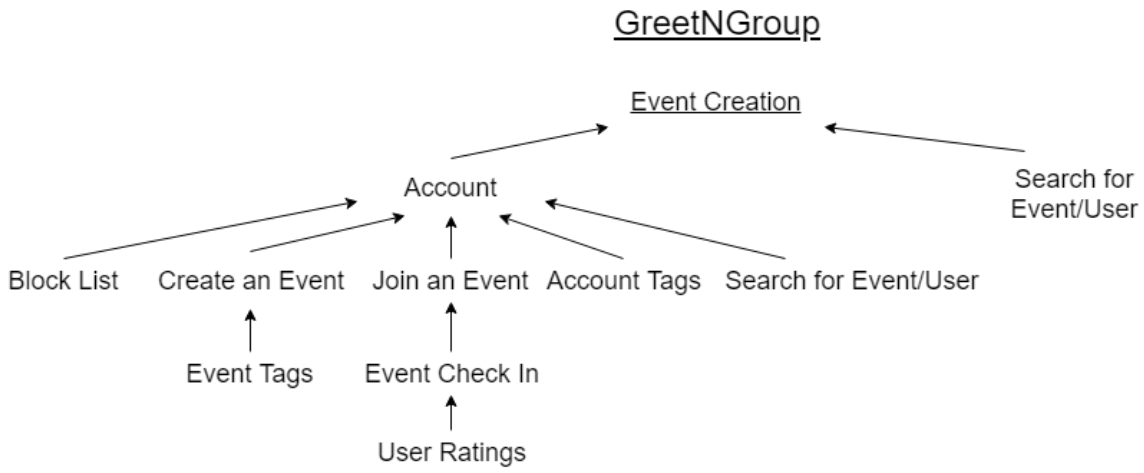
This website will be using SOA for its system design. This structure allows the website to be widely scalable due to its organization of services which can act independently of the actions that call them. SOA also makes use of all references in the system, whereas MVC and MVVM can have unused references that must stay in the system to follow the architecture, but will slow the system down. Although validating every request made will result in longer response times from the system, this downside will only affect live requests, whereas the unnecessary references in MVC and MVVM will affect both the response times of live requests and the time to deploy the system.

Data Format

The system will be using data of the JSON format, it is a format similar to that of JavaScript which will be included within our system. Even more so, because JSON is the officially recognized format for Javascript objects it is beneficial to use it within this project.

Paired with the simplicity of use, JSON has been selected over our alternative XML because of its ability to ease implementation through simplicity in the use of simple tags, and lightweight design for data exchange --as it is described within the json.org website. For our usage in this project, XML does not provide any extra feature we can make proper use of that JSON does not also provide.

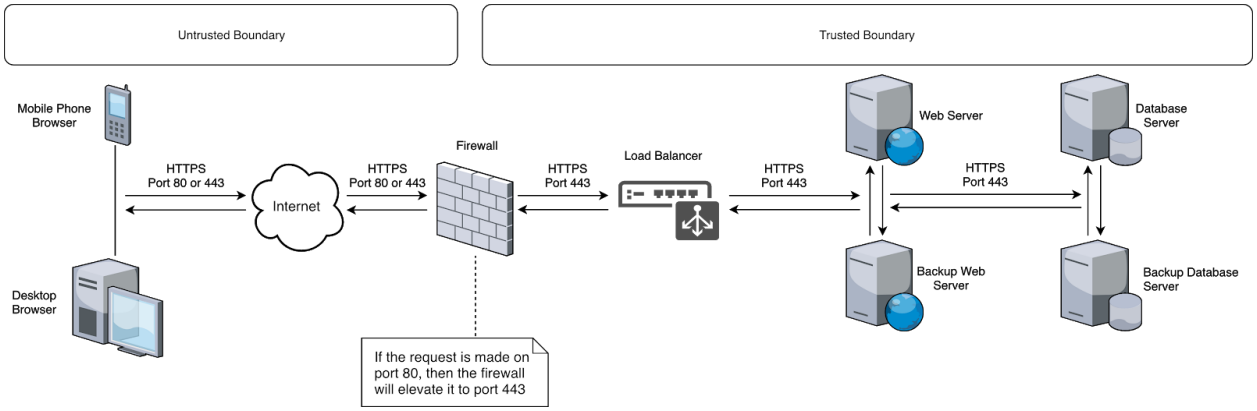
Dependency Diagram



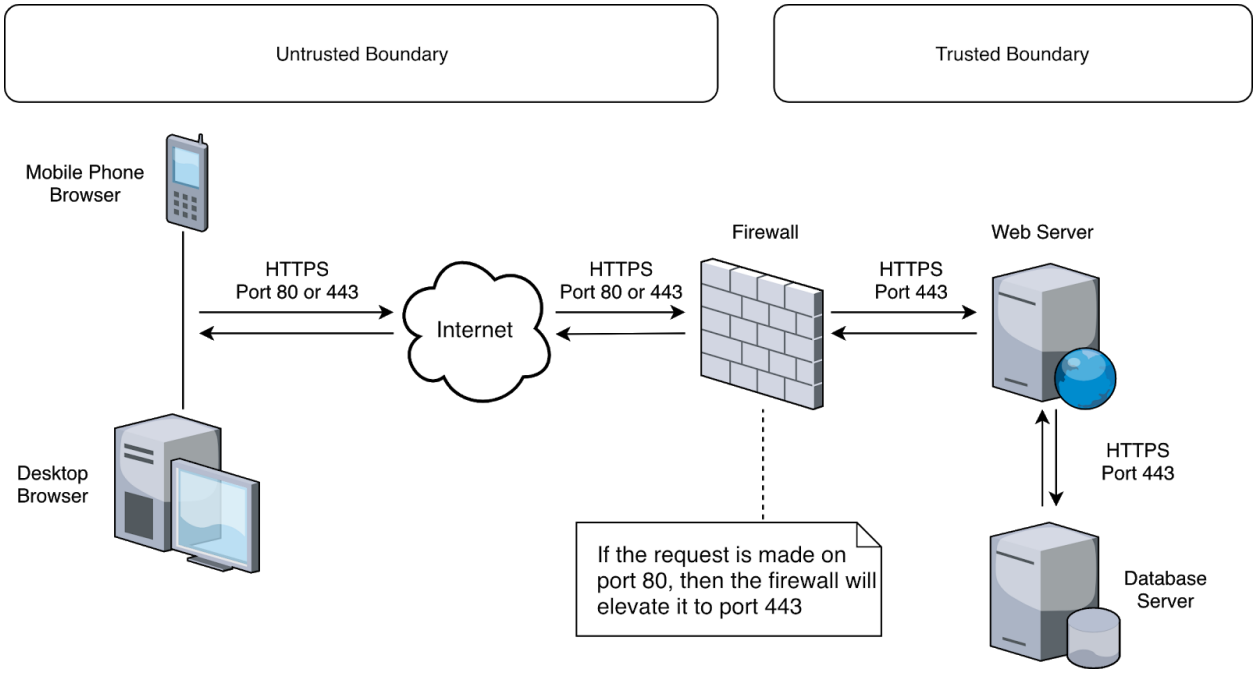
Visual representation of the dependencies of the functions for the website.

Network Architecture

Network architecture ideal diagram



Network architecture actual diagram



Network Architecture cont.

Design considerations:

The actual network architecture lacks a load balancer because it's either

1. Expensive for an easy solution
   or
2. Hard to configure for a cheap solution

A cheap solution would take a lot of resources from our team, which isn't great for a web application that is starting out. Therefore, the expensive solution would be our only option, however, at this point in time, we'd like to keep our costs to a minimum thus forgoing a load balancer.

The actual network architecture also only uses a single web server, and a single database server because it would double the costs of our servers if we had backup servers. Ideally, as more people use our website, we'd be able to cover the costs of the servers and the load balancer and be able to implement them both at a future time.