CECS 491A - Sec 6 - High Level Design Document

Project Name: ArrowNav

Team Longhorn:

Brayan Fuentes

Christian Lucatero

Curtis Nishihira

Miguel Zavala

Spencer Gravel (Team Leader)

October 6, 2021

HL Design Changes Needed

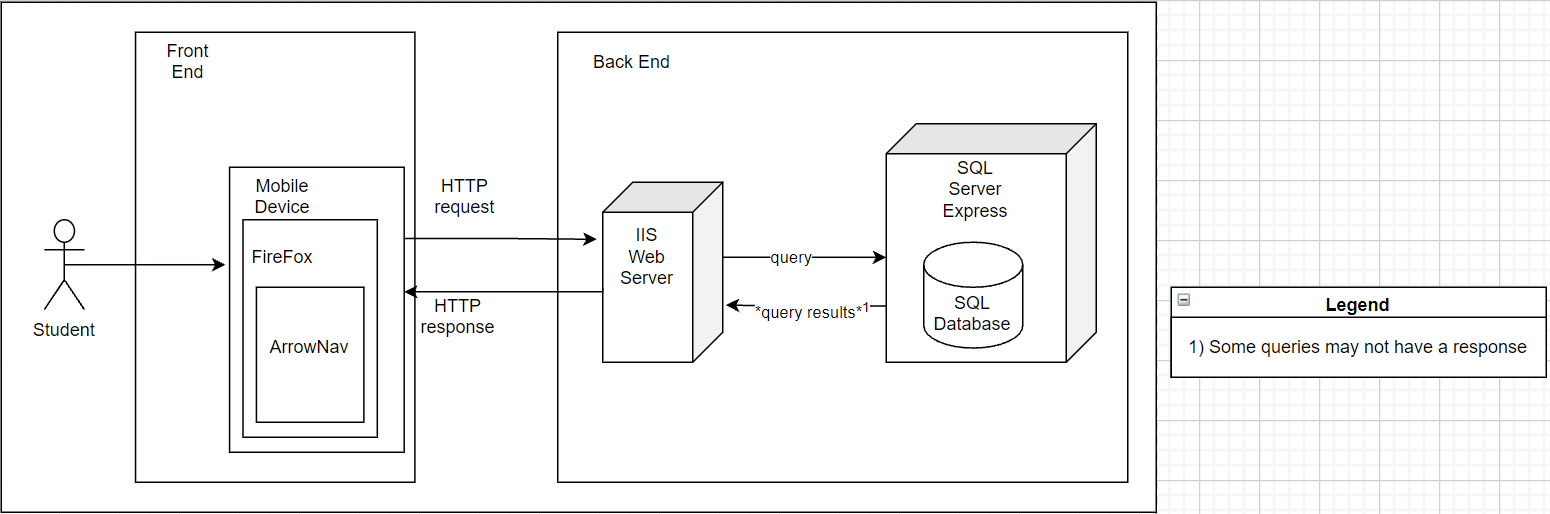
* The front end has error handling set at the VM level thus it is a part of a component, while in the backend error handling is set as a cross cutting concern which is outside of the scope of a component. This inconsistency will reduce comprehension of your implementation.
* An API Gateway does not receive view models as requests. API Gateways handles security, versioning, and routing of requests.
* The HL design does not go into the specifics of the application. It only focuses on the outline of the application.
* The software design diagram does not have a clear organization structure for the backend unlike the frontend. You should have correlated the Microservices design pattern just like how you correlated the MVVM design pattern for the frontend.

| Revision | Date | Author | Comments/Changes |
| --- | --- | --- | --- |
| 1.0.0 | 30 September 21 | Brayan and Curtis | Initial Revision |
| 1.0.1 | 2 October 21 | Brayan and Curtis | Changes based on the feedback given by the client. |
| 1.0.2 | 5 October 21 | Brayan and Curtis | Last client changes |

**Project Overview**

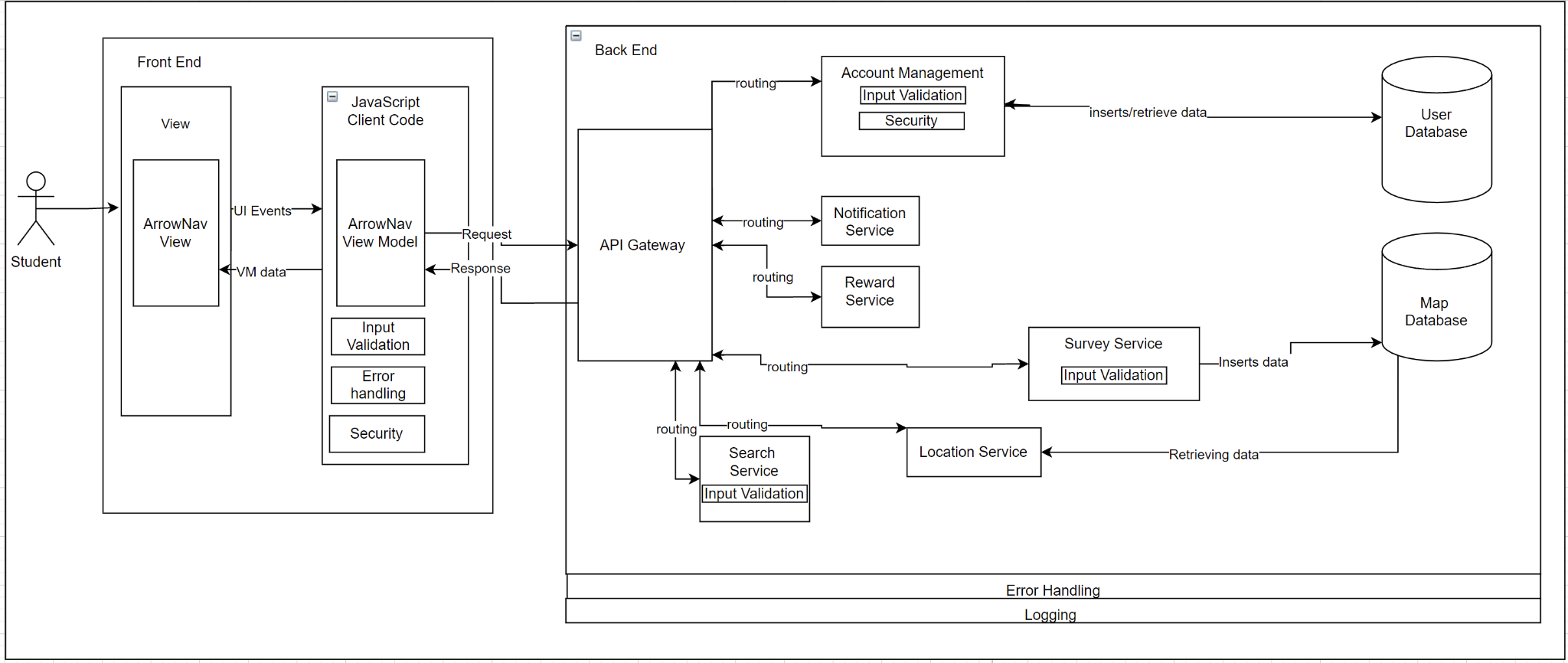
* **Purpose**
  + The purpose of this document is to specify the high-level design for the ArrowNav web application. This document will act as an outline for implementation and discuss the design considerations.
* **Audience**
  + This high-level design is intended to be used by members of the development team that will implement the functionality of the ArrowNav. This document will also be used to communicate the high-level design and design considerations to the ArrowNav project members.
* **Design Process**
  + The high-level design was selected by deciding what aspects of the system were most important and then building architecture around them. The pros and cons of each architecture and technology were discussed in meetings. For each technology proposed we explored and researched feasibility and capability. Web services technology would allow the system to be less coupled and more cohesive

**Design Details:**

**Hardware Design**

* **Front End Architecture**
  + The application will be available on the mobile version of firefox and accessed by student users
* **Front End Components** 
  + Client/Mobile Device
    - Will handle the communication between the web app and the backend web server.
    - Takes in input from the user and makes a request to the web server
  + Browser/Firefox
    - Clients connection to the web and also where the application will be viewed
  + Application/ArrowNav
    - The application will be accessed by the user through their mobile device’s browser firefox
* **Back End Architecture**
  + The client will interact with an IIS web server that contains our complex logic
  + All persistent data will be stored on the SQL Express Server’s database
  + Depending on the webserver of choice, it might allow HTTP traffic rather than HTTPS which could result in network packet sniffing.
  + Web server helps you understand more clearly the transaction occurring between your website and the SQL server.
* **Back End Components** 
  + IIS Web Server
    - Handles the storing, processing, and delivering web pages to the client-side.
  + SQL Server Express
    - The SQL server houses the relational databases for the project.

**Design Details:**

**Software Design **

* **Front End Architecture**
  + SPA MVVM
    - The development team chose to do a Single Page Application with a view model
    - **Selection Reasoning**: The development team chose a SPA incorporating a view model since this model allows for quicker loading, background data fetching, and individual user actions are more responsive.
    - **Decision Impacts**: One result of the view model is a thin client, meaning our front end will only handle simple logic
      * Having simple logic in the front end means that the team is accepting that we will have our complex logic in the back end of our software architecture
* **Front End Components**
  + ArrowNav View
    - **Description**: The MVVM architecture always includes an Initial HTML that is updated only by async data
    - Rather than sending HTML pages to the view every time a new request is made asynchronous updates will be made in the form of data or view snippets
    - **Interactions:** Updated by the view model in the form of asynchronous data updates
  + Client Code
    - **Description**: This represents all the code that will be on the client-side entailing all the simple logic, view model, and extra components such as input validation, error handling, and security versions. Reasonings for these locations will be provided in each respective section
  + ArrowNav View Model
    - **Description:** Model of the initial HTML page and allows for augmentation with asynchronous data updates
    - Any changes to the view will have to be changed using this model as to avoid unnecessary html page sending for each request as stated in the view component
    - **Interactions**: Updates the initial html view through asynchronous updates. Sends request and receives responses from the API Gateway for complex logic operations.
  + Error Handling
    - The development team chose to place error handling in the front end to handle simple user errors through pop up messages or alerts
    - The processing of complex logic errors will occur in the backend
    - Exceptions caught or casted should not crash the system.
  + Input Validation
    - Validating inputs on the client-side is important to prevent roundtrips in which a user is needlessly having to send multiple requests to the back end
    - Front end input validation is enacted on simple user inputs of password fields to check if the input is empty, contains known symbols
  + Security
    - The security that will be in place will be a login which will occur in the client code. It will ask the user for their account and respective password.
    - Login allows for connection of users to their preferences, the front end security will be to protect user schedules
* **Back End Architecture**
  + Microservices
    - **Selection Reasoning**: The Development team’s choice to utilize a microservice architecture was motivated by its scalability which in turn creates greater availability
    - Each of the services can be self-contained and independent because they are deployable units with little to no dependencies on each other
    - The decision to use this architecture was influenced by the choice to make a thin client which places the complex logic of our application on the back end, SPA architectures are often paired with microservice architectures for this reason
    - Additionally High maintainability and testability are benefits to this architecture
    - **Decision Impact:** The development team accepts that this architecture will force the use of independent services in our back end and not allow the for the majority of back end logic to be in one central spot
* **Back End Components**
  + User Database
    - **Description**: Contains all data regarding the user, this includes their account information username/password, and schedule
    - **Interactions**: Receives and Sends data to and from the Account and Schedule Services part of the microservice architecture.
    - Multiple databases allow for a more efficient microservice architecture as a result of less dependency relationships between services and database
  + Map Database
    - **Descriptions**: Holds all the developer-defined baseline data to be built on by the data created from user surveys
    - Contains the data utilized to generate surveys for users to build off baseline data
    - **Interactions**: Sends data to route and capacity services after reach services make a request
  + API Gateway
    - **Description**: Receives view models and routes data to the correct service based on the data being requested
    - **Interactions**: Interacts will all services in the microservice architecture is the main entry point for directing data entering the back end
  + Account Services
    - **Description**: Inputs and receives data regarding a user account to and from the user database
    - **Interactions:** This service will only interact with the user database it is accessing. Sends data back to the API Gateway.
    - **Input Validation**: Prevents improper data from being entered into the database.
    - **Security:** Moves the username and password to storage as well as validates login attempts
    - **Error Handling:** Handles exceptions when trying to write or read any data from the user database. Additionally handles any connection issues to the user database.
    - **Logging:** Logs all transactions between the service and database. Logs any exceptions or errors handled.
  + Notification Services
    - **Description:** Supplies notification data to the view model based on developer and user-defined conditions
    - **Interactions:** Sends data back to the API Gateway.
    - **Error Handling:** Handles exceptions/errors and will send messages to the client if needed. Exceptions nor errors should crash the system.
    - **Logging:** Logs any exceptions or errors handled. Log any data sent back to the API Gateway.
  + Reward Services
    - **Description:** Provides model with reward data based on developer-defined requirements
    - **Interactions:** Sends data back to the API Gateway.
    - **Error Handling:** Handles exceptions/errors and will send messages to the client if needed. Exceptions nor errors should crash the system.
    - **Logging:** Logs any exceptions or errors handled. Log any data sent back to the API Gateway.
  + Location Services
    - **Description**: Receives survey data from map database and includes said data when returning routing data to the view model. Augments survey data from the map database into capacity data then passes it to the view model.
    - **Interactions:** This service will only interact with the user database it is accessing and augmenting the view model data. Sends data back to the API Gateway.
    - **Error Handling:** Handles exceptions when trying to write or read anything data from the map database. Additionally handles any connection issues to the map database.
    - **Logging:** Logs all transactions between the service and database. Logs any exceptions or errors handled. Log any data sent back to the API Gateway.
  + Survey Service
    - **Description:** Inputs survey data received from the front end into the map database
    - **Interactions:** This service will only interact with the user database it is accessing. Sends data back to the API Gateway.
    - **Input Validation:** Prevents improper data from being entered into the database.
    - **Error Handling:** Handles exceptions when trying to write or read anything data from the map database. Additionally handles any connection issues to the map database.
    - **Logging:** Logs all transactions between the service and database. Logs any exceptions or errors handled
  + Search Service
    - **Description**: Augments the view model based on the user-defined parameters, then sends that data back to the front end
    - **Interactions:** Augmenting the view model data. Sends data back to the API Gateway.
    - **Input Validation:** Prevents improper data from being entered into the database.
    - **Error Handling:** Handles exceptions/errors and will send messages to the client if needed. Exceptions nor errors should crash the system. Log any data sent back to the API Gateway.
    - **Logging:** Logs any exceptions or errors handled