High-Level Design

Revision Number: 1.0

Last date of revision: 9/24/21

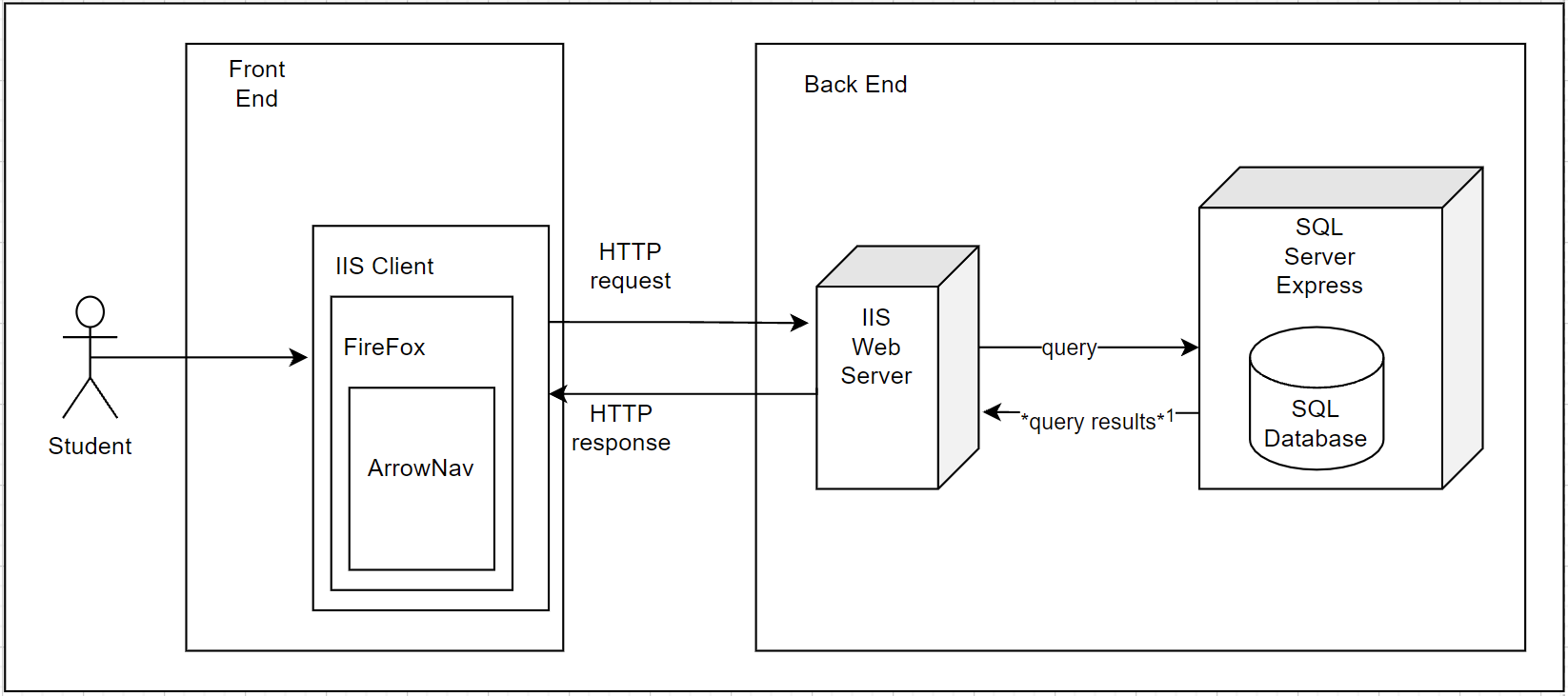
Brayan Fuentes, Curtis Nishihira

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

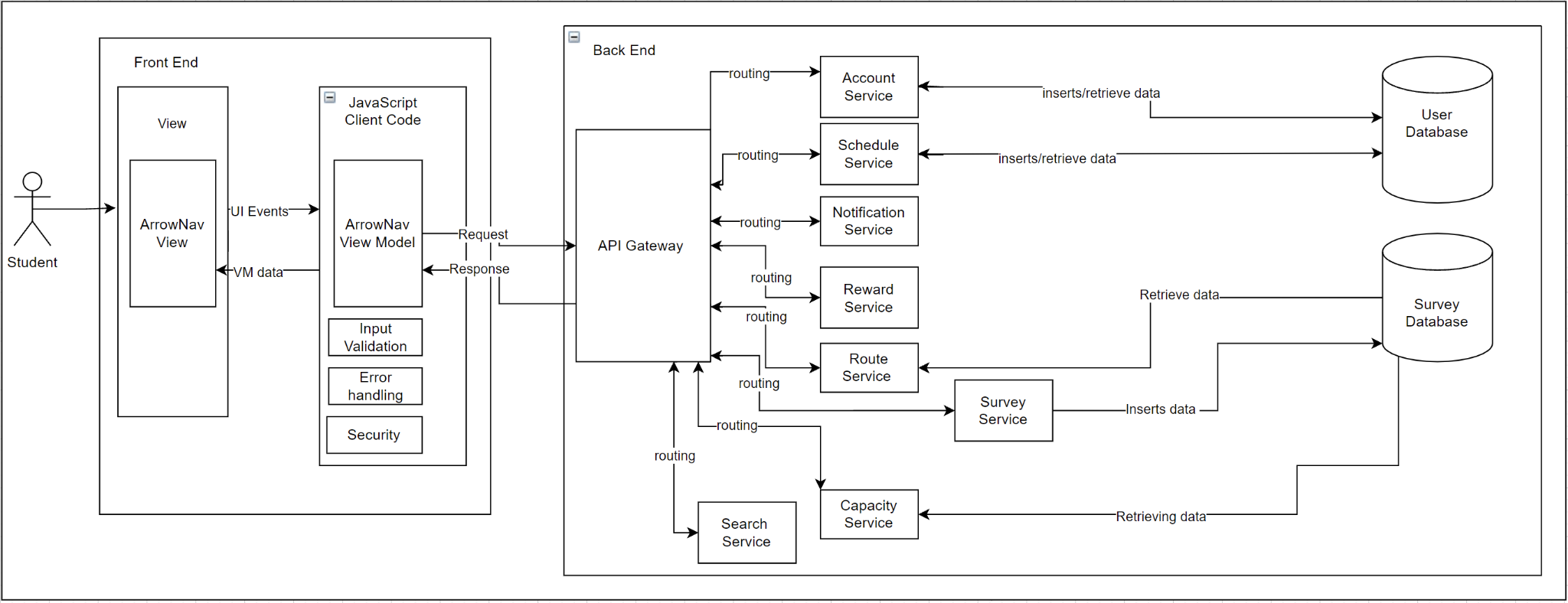
**Project Overview**

* **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 fire fox and accessed by student users
* **Front End Components** 
  + Client
  + Browser
  + Application
* **Back End Architecture**
  + The client will interact with a IIS web server which contains our complex logic
  + All persistent data will be stored on the SQL Express Server’s database
  + Chose to have to servers because: ??
* **Back End Components** 
  + IIS Web Server
  + SQL Server Express
  + SQL Database
* **Front End to Back End Communication ?**

**Software Design **

* **Front End Architecture**
  + SPA MVVM
    - The development team chose to do a Single Page Application with a view model
    - The development team chose a SPA incorporating view model since they can load more quickly, fetching data in the background, and individual user actions are more responsive.
    - The thin client is the view and view model that will handle the presentation and simple logic based on the user input. Leaving the complex logic to the backend.
* **Front End Components**
  + ArrowNav View
    - Initial HTML that is updated only by async data
  + ArrowNav View Model
    - Model of the initial HTML page and allows for augmentation with async data updates
  + Error Handling
    - There will be error handling in the front end to let the users know that there was some form of error through a pop-up message or some type of alert. The actual handling of the error will occur in the backend
    - Exceptions shouldn’t crash the system.
  + Input Validation
    - Validating inputs on the client-side is important to prevent roundtrips.
  + 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.
* **Back End Architecture**
  + Microservices
    - Development teach chose microservice architecture for its scalability which in turn creates greater availability
    - Also a result of having a thin client which places the complex logic of our application on the back end
    - Having a self-contained individual service results in a reduction of dependencies as they are independent.
    - High maintainability and testability
* **Back End Components**
  + User Database
    - Contains all data regarding the user, this includes their account information username/password, and schedule
  + Survey Database
    - Holds all the baseline data to be built on by user surveys for traffic prediction and capacity calculation
  + API Gateway
    - Takes in the model and routes to the correct service based on the data being requested
  + Account Services
    - Inputs and receives data regarding a user account from the user database
    - Allows for username/password changes
  + Schedule Services
    - Inputs and receives data regarding a user’s the schedule/calendar from the user database
    - Allows for inputting courses into the schedule and or events/reminders
  + Notification Services
    - Supplies notifications to the web application based on developer and user-defined conditions
    - Ex: water reminder
  + Reward Services
    - Controls the developer-defined rewards available through usage of the web application
    - Alerts and provides users with rewards for completing actions within the application
  + Routing Service
    - Interfaces with survey data and baseline data to present view model with traffic predictions
    - Supplies model a route from one user-defined location to another
  + Survey Service
    - Controls the surveys sent to the model
    - Inputs and retrieves data from surveys into the survey database
  + Capacity Service
    - Handles the logic for calculating capacity based on survey and baseline data
    - Sends capacity data to the model
  + Search Service
    - Supplies locations based on user request
  + Input Validation
    - Input validation will also be checked in the backend at the controller since we shouldn’t blindly follow the user since they are likely to enter something incorrectly.
  + Error Handling
    - Handles exceptions
      * Will send message to client if needed
    - Exceptions shouldn’t crash the system.
  + Logging
    - Client code and views do not require logging thus it will only be necessary to do logging in the back-end.