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)

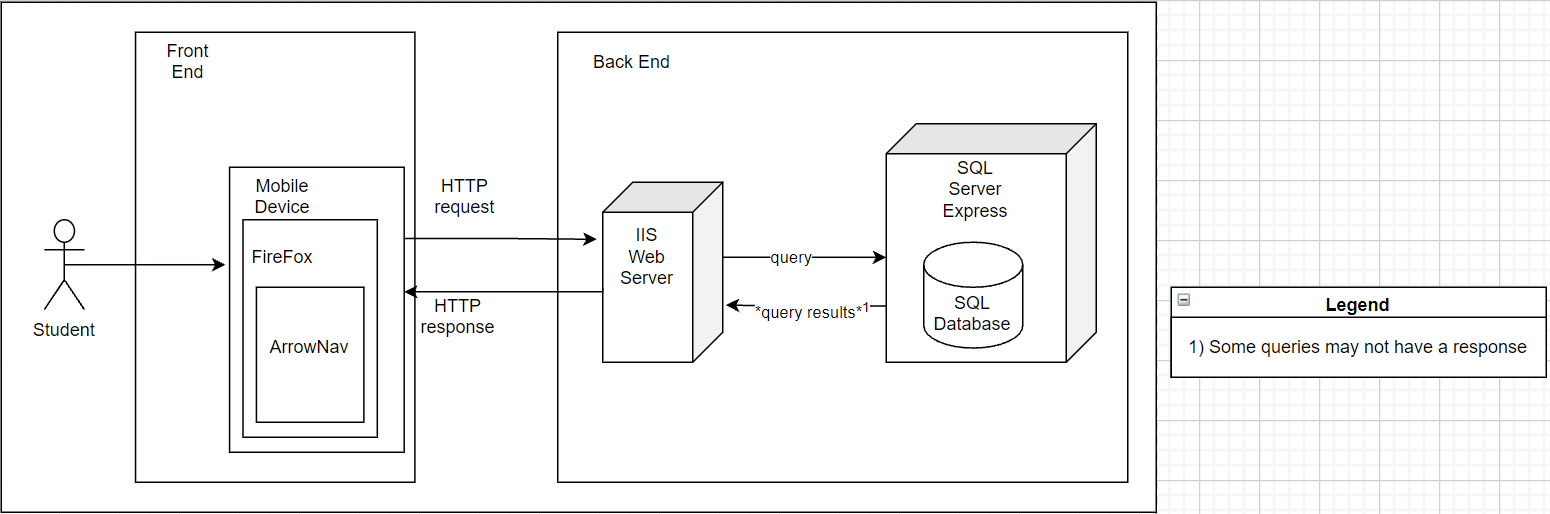
October 6, 2021

| 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**

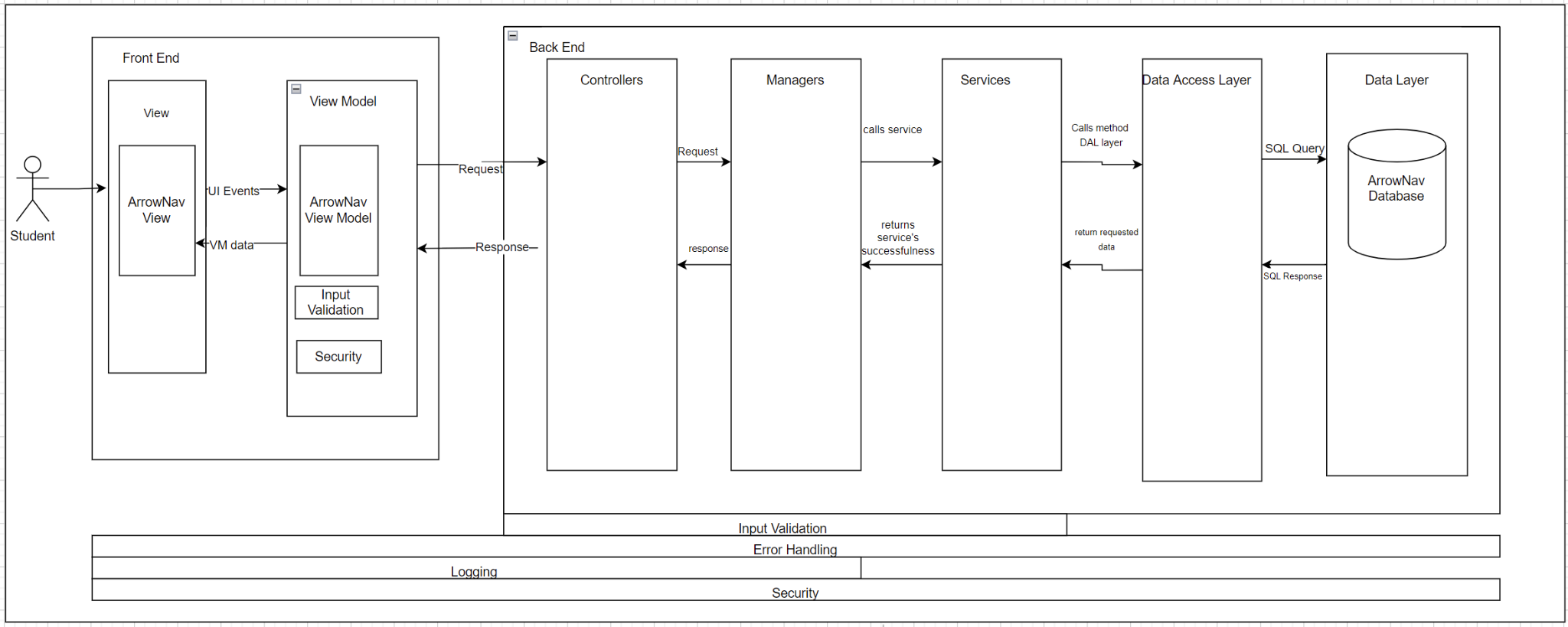
* **Purpose**
  + The purpose of this document is to specify the high-level design for the ArrowNav web application. This document will act as an outline for implementation and discuss the design considerations.
* **Audience**
  + This high-level design is intended to be used by members of the development 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**
  + Layered Architecture
    - **Selection Reasoning**: The Development team’s choice to utilize a layered architecture was motivated by component isolation and open close abilities.
    - 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 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**
  + Service Layer
    - **Description**: The main functionality of features will exist in the service layer. The code for completing feature requirements will mostly exist in the service layer.
    - **Interactions:** Depends on the data access layer for all interactions with the data store but otherwise has no dependencies. Will be called and handled from the manager layer.
    - **Error Handling:** Handles any errors occurring as a result of the service implementation and or call.
    - **Security:** Enforces requirements that only certain services can be enacted by the correct manager or authorized user
    - **Input Validation:** Validates that the request for a service to be made meets the correct requirements to perform that service.
  + Manager Layer
    - **Description:** Managers all the services in the service layer. Individual managers are in charge of organizing the different correlating services in the service layer.
    - **Interactions:** Is dependent only on the service layer and the service layer’s dependencies. Will be called from the controller layer after a request is made.
    - **Logging:** The last of the logging layers in the back end with a flow moving from the front end to the back end. This is because the manger will know what services are being called and therefore any data access being called as well.
    - **Security:** Only allows services to be called for authorized and authenticated users
    - **Error Handling:** Handles any errors that result from the controller request or service result.
    - **Input Validation:** Validates that calls to services are valid and can be made
  + Controller Layer
    - **Description:** Takes in web request from the data model for completing feature functionality.
    - **Interactions:** Takes in request from the front end mvm architecture and in turn calls managers that will handle the services required for that request to be completed.
    - **Input Validation:** Handles any invalid requests that are made in order to complete the request.
    - **Logging:** Logs all requests that are made from the front end layer to the back end.
    - **Security:** Enforces request protection to now allow for requests from places outside of the mvvm.
    - **Error Handling:** Handles any errors that occur as a result of requests being fulfilled
  + Data Access Layer
    - **Description**:The main layer in charge of inputting or receiving any information from the persistent data layer.
    - **Interactions:** Is dependent on the existence of the data store. Will take all request and method calls from the services layer. Individual services will take data from the data layer to complete functionality.
    - **Security:** Enforce authorization and authentication for which users can access what data from the data store layer
    - **Error Handling:** Handles exceptions when trying to write or read anything data from the data store. Additionally handles any connection issues to the data store.
  + Data Store
    - **Description:** Is the persistence data layer where the databases will exist and store information for the program.
    - **Interactions:** This service will only interact with the data access layer which will provide information to the data access layer for the application.
    - **Error Handling:** Passes sql errors and exceptions to the data access layer to be handled there.
    - **Security:** Protect from sql injections by using the built in security microsoft sql server employs