Software Detailed Design

**Data Design**

There are two ways to think about data design in this project; how to retrieve and display data to the user, and how to manage and control data within the application. The application will have a database controller that retrieves information from the database and displays it on the page. Upon entering the student client view, Get/Set calls to the database will be automatically made and displayed. Some data may be altered when the user saves information and an update will be returned to the database. The admin client view will be able to make Get calls to the database but that is all.

**Firebase:** We are using Firebase, which a schema-less, noSQL database. Instead of tables and records, firebase works in JSON trees. All Firebase Realtime Database data is stored as JSON objects. Upon adding data to the JSON tree, it becomes a node in the existing JSON structure with an associated key.

**Authentication:** Upon sign-up/login, there will be a check with the database to see whether the user is a student/admin and populate the correct UI accordingly. The Student UI allows for quick and easy updating of their checklist items and shows their current state. The Admin UI allows for quick and easy querying of student data and returns data from queries.

**Architectural Design**

**Application**

The web form will be constructed by our group. All implemented logic is guaranteed to control the flow of the application and offer minimal maintenance. Maintenance may come in the form of changing a UI element; however, nothing data related should need to be manipulated outside of what we will grant access to a super user.   
  
**Model/View/Controller (MVC):** The application will consist of Firebase, HTML/CSS, and Vue.js, in that order, for the Model, View, Controller design pattern we are implementing. MVC is commonplace in modern applications and is an industry standard for n-tier architectural design. Simply put, our application will allow a user to manipulate UI elements but be strict on how it allows the user to accomplish this. They will never be able to directly access the database; that is why the controller is implemented. Our controller will be a mechanism, think pipeline, between the client and the database. e.g. The user sends a request - the controller receives the request and sends the request to the database - the database receives the request and returns to the controller – the controller returns the expected request to the user. This is data hiding and separations of concerns at work.

**Database**

Our DBMS will be constructed from a list of items determined by our sponsor. We will be adding data exactly as directed to stay consistent with their current system. It will be maintained by the nursing program via a super user access client.

**Interface Design**

**User Interfaces**

There will be two separate interfaces, a client for students and a client for admins. The student interface will have a list of items that they can manipulate while the admin interface will allow querying of student progress. For the student interface, the user will send a status of their checklist item to the database. Since the sole purpose of the interface is to update items, the user does not need much in the way of retrieval. However, the admin interface will be retrieving information from the database. The user will be sending queries and retrieving data.

**Data Interfaces**

We will use Vue.js as a controller for our database. The best documented option for Vue.js at this point is Firebase. Vue is responsible for bringing in a list of student tasks as well as updating/managing them with Firebase.

**Procedural Design**

For the procedural design element, we have been sketching out model, view and controller elements. We have worked out a primitive mockup of the UI as well as a database tree with pipelines to associate the work. This has helped us facilitate an idea for the structure of the application with extreme focus on the data layer.

**Data:** The procedural thought process is similar to SQL, sketch tables with record information. We will need a “taskStore” key that we can store/retrieve our list from. Also, a “taskAdd” that adds a new task to “taskStore.” Perhaps admins will get privileges to a “taskInactive” key that will remove an item from “taskStore” and hold it in “taskInactive.” For user verification we will need a “users” key to store users and have a check to verify whether a user is a student or admin. Further, we will need a “addStudent” key and “addAdmin” key that puts a student or admin in “users.” There are other optimizations for the database but this is the baseline.

**Contributions**

This project does not require much in the way of split declarations to reach the final product. The contributions of each team member will be in the full stack and we will be working on version control with GitHub, mainly to stray from redundancy/overlap. This is intended not only because of the straight-forward design but also for our own knowledge of the full process.