Design Specifications

DAC Project

CS 448

Austin Bos, Aaron Saucedo, Ivan Nieto

Design Specifications 001 for notetaking application requested by the Army Data and Analysis Center

Date

03/19/2020

Table of Contents

Introduction

The purpose of this document is to outline in detail the design of the application. We will describe in detail the implementation of the various systems that the application is segmented into and the interactions between those systems..

1.1 Design Drivers

The main drivers for the design of the application are usability, reliability and functionality. All of these drivers were taken into consideration when deciding on the technologies and design that will be used for the application.

Usability - The application must be user friendly and require little to no training for a user to be able to successfully utilize the various features of the application. This means that the user interface must feel familiar to the user by having a similar layout to most popular websites.

Reliability - Data entered by the users into the application must be backed up in some way. For this we will have each user created project be its own git repository in order to allow us to version that project's contents.

Functionality - Successfully creating projects and adding tasks and findings to those projects is of the highest priority. This driver also includes the creation of specifically requested tools such as the ability to export the contents of a project to a word document that will serve as the beginning of a report.

1.2 Acronyms, Abbreviations, Definitions

- CCDC-DAC – Combat Capabilities Development Command Data and Analysis Center

-Frontend - The applications graphical user interface where the user will be able to access all functionality for the application.

-Backend - Segmented part of application handling application tasks not seen by the users.

-DOD - Department of Defense

-CEAD - Cyber Experimentation & Analysis Division

-SME - Subject Matter Expert

-Electron - An application building framework that simplifies the creation of desktop graphical user interfaces using web technologies and programing languages.

-React - A JavaScript library that helps in the creation of large web-based applications.

-Express - Node JS library that allows us to create a local server.

-Communication Interface - Collection of gRPC scripts for communicating over a LAN connection with another application.

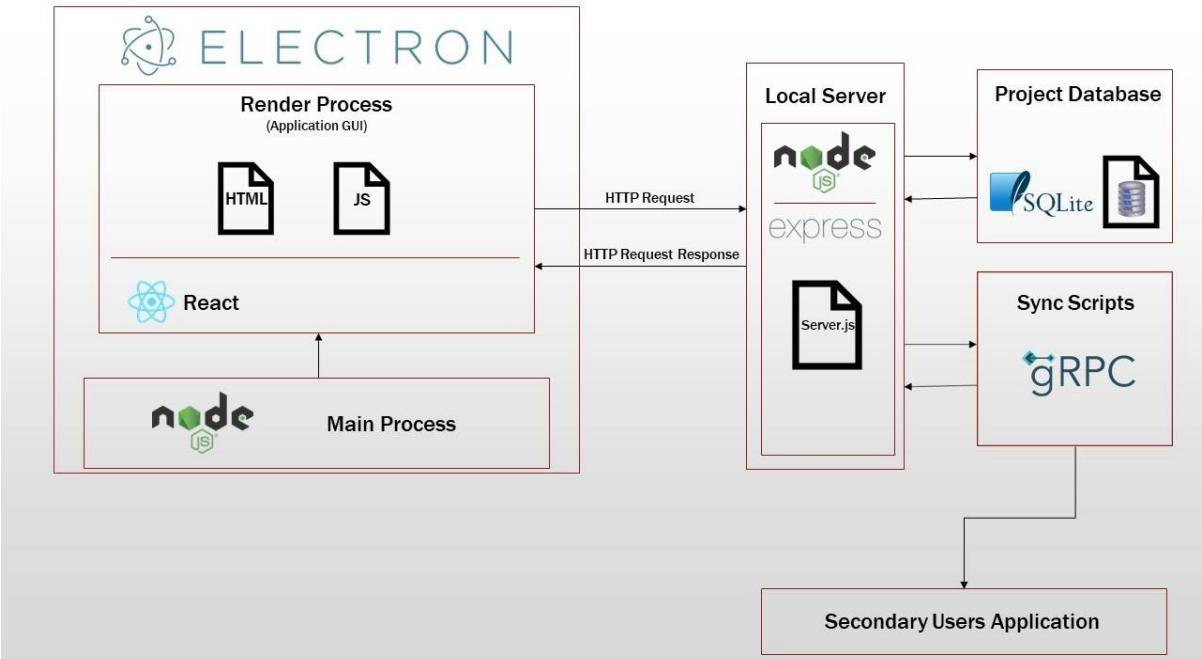
Platform, Language, and Environment

Overview of the environment and aspects of the project

For our project we have decided to use Electron as our framework to create the executable application. This framework will allow us to create our application using web tools, specifically React to easily create our application. We have chosen to use React as our main javascript library so we can take advantage of the many React libraries available for free online to develop a more professional looking application while simultaneously saving us development time. React will also allow us to create a frontend using modular components so we can easily develop and take full advantage of reusable components which will further reduce our overall workload.

System Architecture

High level project architecture specifications



Our application will be packaged by Electron into an executable. The user will start the application by running the executable which will trigger the main process of Electron. The main process will then start the render process that will be using React to create a GUI window which the user will then interact with.

Simultaneously to this, the main process will also start the Node express server which will receive http requests from the render process. Whenever the application needs to interact with the SQLite database or sync with another user (denoted Secondary User Application in the diagram), the React GUI will make an http request to the express server then the express server will call scripts to do the work and then return the results to the render process and display them to the user.

System Design

Low level design specifications

4.1 Architecture Component Interfaces

The web application will have interfaces that interact with each other depending on the functions the end user or team lead is implementing.

4.1.1 User Interface

The user interface created by the React library using JavaScript, facilitates the creation of a user-friendly GUI that allows the end user to communicate with the communication and database interface through the use of http requests. When the user wants to query the database and see all findings and tasks for a specified project the frontend will indirectly query the database by sending an http request to the local server which will query the database and return the desired information. This process is replicated for all interactions with the database and communication interface.

4.1.2 Communication Interface

The communication interface gets called on whenever the end user needs to sync notes to the team lead or when the team lead needs to sync data to all team members. This process is started by the React frontend which sends an http request to the node server which then runs the necessary scripts to sync the database. Whenever the sync function is clicked on in the user interface, it will call upon the communication interface that has a gRPC server that handles the transfer of data.

4.1.3 Express Server

The express server will be the middleware handling communications between the React frontend and the SQLite database or communication interface. On startup the server will begin listening on a private network. On this network the frontend will send http requests to the server. Based on the request the server will call the

necessary scripts to either access the database or the communication interface and return any relevant information.

4.2 Architecture Component Designs

4.2.1 User Interface

For the user interface we will be dividing the GUI into pages. These pages will be used to display different information to the user. We will have pages specifically dedicated to display notes and tasks for a specific project, a project creation page, a project deletion page etc. These pages will be created using modular components so as to speed up the development process. The pages will follow a similar design so as to allow more reusability and modularity between pages. We will also be using the Material-ui React library to import React components. These components will help display data, collect user input, and help in the overall user experience and professional look of the GUI.

4.2.2 Express Server

The server will be written in JavaScript with the use of Node JS and the express library for Node. The server will be accessed by the frontend through http get and set requests. For example, if the frontend needs a list of all the projects that have been declared by the user the frontend will send an http get request to the server requesting this list, the server will then call a script in another file to query the database for this information then pass it along to the frontend in the request response. These requests will pass along any relevant information to the server which will then call other Node scripts that will communicate with the database or communication interface depending on the specific get or set method that was requested.

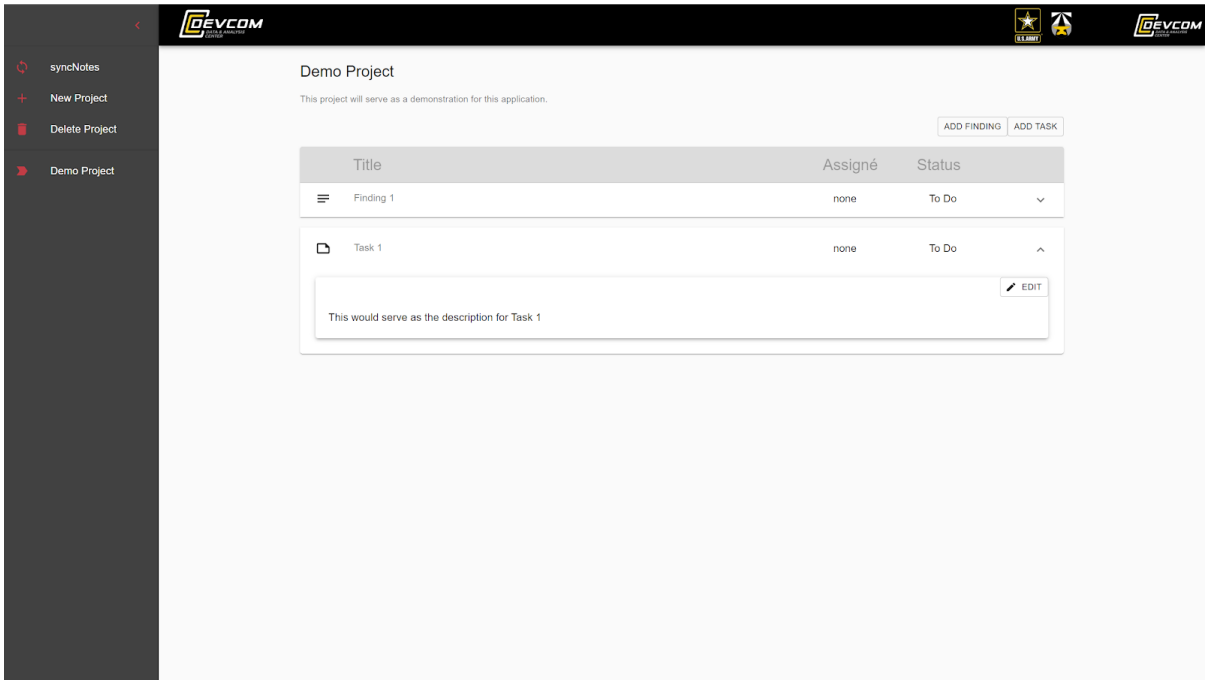
4.3 Communication Interface

The communication interface will consist of a client server architecture using gRPC and google protobuf as the communication protocol and serialization respectively. The team lead will act as the server, opening up a port on an external interface. The purpose of the server will be to accept a tcp connection from one of the team members. This connection will be secured with TLS. Through this tunnel, the team member can send data from their database, to the team lead, where the team lead will store these requests. The functionality will be enacted through the user's interface. If need be, a mutual authentication can be established between the team lead and team member to ensure authentication of each person, before any data is transmitted. This will be implemented if desired by the customer.

User Interface Design

GUI Interface specifications

The user interface is a web page created by React with JavaScript; the frame is built around Electron. The interface includes many other pages depending on what the end user's task is or what the user is implementing. The front page has the basic overview of the projects that have been created and assigned to the analyst(end user). The page also has a side panel on the left side of the page, which allows the end user to pick a specific project, sync projects, create projects and delete projects. Each of these options take you to a new web page to allow the end user to modify their projects with findings and other relevant information contained to the current project.



When creating or editing projects, the application keeps it simple by having a few text boxes. For example, if creating a project, there will be two textboxes, one for the title of the project and the other for the description. If editing a project, it will also have a textbox to add findings from the specific task or other findings not associated with that particular project.

Deployment Design

Application delivery mechanism

The application will be delivered in the form of an executable. The various operating system specific executables will be available for download from our GitLab or GitHub repository. Once downloaded the user only needs to double click the application to run it. Any further maintenance of the application will be done by the DAC team per their request.

Coding Standards

For our coding standard we have taken two online sources into consideration to construct our coding standards. These sources regard the best practices when programming with JavaScript and React. Our sources are **https://medium.com/@krishnarai1985/coding-standards-and-good-practice-for-react-native-apps-c8401e87f2d** and **https://www.w3schools.com/js/js_conventions.asp**

1. **Naming**
- folders and subfolders should start with small letters.
- Name should include the component according to its path
- If file is inside folder of same name, no need to repeat name
- Include all the controls in a single import to know they belong in the same module and end with semicolon.
- Class name should be declared as file name that will be easy to import
- Objects and variable declaration should be in camelCase.
- If we use a semicolon then should be used in all places at end of statement

2. **Layout**
- Always end a statement with a semicolon
- If continuation lines are not indented, we indent them one tab (four spaces)
- Add at least one blank line between method and definitions
- Should be no line space between similar looking statements or similar code
- Define arrow functions as class instances, arrow function will point to the class

3. **Commenting**
- Place the comments on a separate line, not at the end of a line of code
- Begin comment text with an uppercase letter
- End comment text with a period
- Insert one space between the comment delimiter and comment text

4. **Language Guidelines**
- Variable can contain any data
- Numbers are both integer and floating.
- Strings must be surrounded by quotes
- Backticks(``) are extending functionality quotes, allows to embed variables and expression into string by wrapping them
- Boolean only has two values T or F

5. **Variable names**
- use camelCase for identifier names (variables and functions)
- All names should start with a letter
-

6. **Code Indentation**
- Always use 2 spaces for indentation of code blocks
- Do not use tab for indentation

7. **Statement rules**
- Always end a statement with a semicolon
- Put opening brackets at the end of the first line,
- Use one space before the opening bracket
- Put closing bracket on a new line, without leading spaces
- Do not end complex statement with a semicolon

8. **Object Rules**
- Place the opening bracket on same line as the object line
- Use colon plus one space between each property and its value
- Put quotes around string values
- Do not add a comma after last property-value pair
- Place the closing bracket on a new line, without leading spaces
- Always end an object definition with a semicolon

9. **Line length**
   - Avoid lines longer than 80 characters

10. **Naming conventions**
- Global and constant variables should be written in uppercase