CostCoach

Technical Documentation

The purpose of this document is to provide information regarding the overall design of the app and the technologies used for the completion of this app

The document is divided into the following sections:

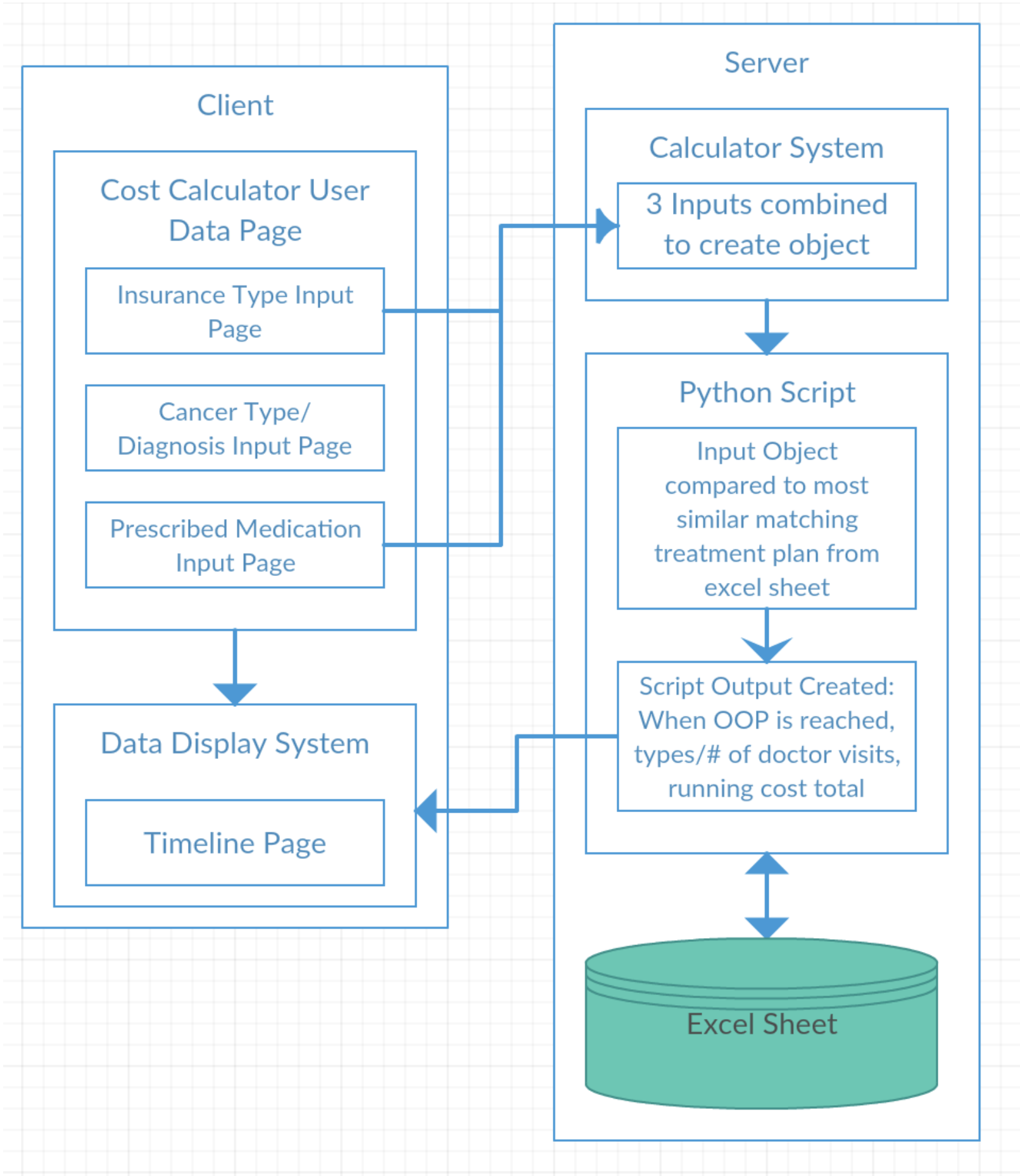
1. Design Justification
2. Technologies Used
3. Running the Application

1. Design Justification

Overview of Project:

This project's aim is to build a web app. The intended users are cancer patients and medical professionals. The goal of the app is to provide users with an estimate of cost of treatment until out of pocket maximum is reached, and an estimated date to reach out of pocket maximum. The app is dependent on a data set and python script provided by the client, which takes the inputs provided by the user, and calculates the outputs. The app has several other features, such as a conversational guide, which provides content to users relating to having a conversation with a patient or medical professional about the expenses of cancer treatment. Below is a general lay out that describes the different pages in the app, and how they work in tandem:

The goal of our design was to create a simple, user-friendly design that allowed individuals of all age demographics to easily navigate through the contents of the web application. In addition to creating a simple/easy to use user interface, we also prioritized giving the web application a professional look, and a color scheme that would allow it to be integrated with the client's other web applications (Pathlight).
Below is a HLA diagram displaying how data is currently interchanged in the web application:



To summarize the HLA chart above: the user inputs data in the cost calculator page which is comprised of three other pages (stepper pages) that allow the user to input their insurance type, diagnosis, and prescribed medication. The data is then passed as an object to a python script (calculator system) that creates an insurance object. This object is the processed by another python script that scans an excel sheet (provided by Duke-Margolis Health Policy) and finds the data point that most resembles

the user inputted object. A timeline is then generated based on the data (OOP date reached, type/# of doctor visits, and running insurance cost total) passed by the script.

2. Technology Used

The main technologies used for the app were:

Technology	Version
Vue.js (Frontend)	2.5.2
Vuetify (UI Framework)	1.3.5
Flask (Backend)	1.0.2
Jest (Unit Testing)	23.6.0

As of December 6th, 2018

In Section 3, Running the Application, these technologies (main) will be brought up again in greater detail.

A complete list of all the technologies used and their version models can be found in the package.json file

On November 27th, 2018, the Event-Stream package was hacked (malicious code added). Because of the hack, a patch was released which we downloaded and integrated into the app. The patched version of Event-Stream is 4.0.1

3. Running the Application

Frontend

Project structure

Code for the frontend can be found in the “client” directory. The project was built using the Vue.js Webpack template. Information on the project structure can be found here.

Technologies
CostCoach is conceptually and functionally a simple application, making it very important for us to nail the minor details in its look and use. This drove us to build the frontend on Vue.js for several reasons:

- It’s super lightweight compared to other frameworks such as React, which matches the simplicity and scale of the site
- It enables us to use the UI framework Vuetify, which has all the components we needed to make the app look amazing, from the timeline to the survey steppers (and much more!)
- It has great support for transitions and animations, which are what really set apart a great application from one that is merely functional
- Vue’s client-side rendering and dynamic data-binding allows for a seamless user experience, from navigating between pages to submitting the CostCalculator survey

We also used a few node packages to implement some of the features.

- jsPDF: for generating the PDF in the “Conversational Guideline” page
- Axios: for sending requests to our backend

More details on dependencies can be found in the project’s *package.json*.

Getting started
To install the application’s dependencies, enter npm install in the terminal while in the client directory (make sure you have Node installed).

To run the application, enter node app.js

Backend

Project structure
Code for the backend can be found in the “server” directory.

Technologies
CostCoach’s Cost Calculator is built on an algorithm written in Python. Given this, we needed an easy-to-use, lightweight Python framework for building the backend. We ultimately ended up using Flask.

Getting started
Set up a Python virtual environment in the server directory. Our gitignore assumes you’ve named your virtual environment “env”. Install the required dependencies using pip install -r requirements.txt while your virtual environment is activated.

To run the application on Windows, enter py run.py (the “py” command may vary depending on the version of Python you are running and your OS) while your virtual environment is running.

Deployment

We deployed the frontend and backend separately using Heroku. As of now, there is no SSL certificate on either site, and the frontend is configured to send requests to the secure version of the backend (prefixed by “https”).

Frontend: http://costcoachclient.herokuapp.com/#/
Backend: https://costcoachserver.herokuapp.com/

Making changes
To see your frontend changes reflected in the production site, navigate to the top level project directory in the master branch and enter the following into your terminal:

```
git subtree push --prefix client frontendprod master
```

For the backend:

```
git subtree push --prefix server heroku master
```

Unit Testing

Unit Testing was completed via Jest.

Getting Started
In your client directory, install jest via the following command in terminal:

```
npm install --save-dev jest @vue/test-utils
```

After the download is complete, create spec.js files in the Src/Component folder to create tests for different components. The spec.js files should be named after the component you plan on testing (e.g. Testing Home.vue, testing file name should Home.spec.js). Test syntax can be found here.

Once the tests have been written, in the client directory type the command:

```
npm run unit
```

in terminal to run your tests