Group 1 Cloud Final Project Report

By

Christopher Burbine - christopher_burbine@student.uml.edu
Alexander Glasser - alexander_glasser@student.uml.edu
Eric Calandriello - eric_calandriello@student.uml.edu
Nicolas Stanzione - nicolas_stanzione@student.uml.edu
Glen Anderson - glen_anderson@student.uml.edu

Adapted from IEEE documentation standards (particularly IEEE-1016 and IEEE-830)

Table of Contents

1. Introduction

1.1. Document Description

The following document contains all the details about the cloud component of the group 1 semester project. This document provides explanations for all of the technologies and processes used to develop the component. Depicted also is a brief explanation of how the cloud component would integrate with other components in our system. Each subsystem inside of the cloud component is explained in detail so as to acclimate the reader with the component.

1.1.1. Introduction

The purpose of this project was to create an application that integrates three distinct components: a sensor, a data aggregator, and the cloud interface. The sensor component collects data from a user (client). The data aggregator component then stores that data (and other inputted data) for the cloud interface component to display. This document is involved specifically with the development of the cloud component. Our primary goal was to use the data provided by the sensor and data aggregator components and create an easy to use web interface for a specific user base to view and manipulate the data.

Due to the limited time allotted to complete the product, the feature set is heavily limited. The development team was able to reach many of our set goals, but was unable to add a substantial number of useful features. As such, this is a proof of concept and depicts the progress that has been made in developing the application.

Our product allows doctors, physical therapists, and rehabilitation specialists the ability to set and track goals for patients. These goals are tracked using data provided by the sensor and data aggregator components, as well as patient and physician inputted data.

1.1.2. System Overview

The overall system involves three distinct components. The cloud subcomponent also involves three subcomponents: the login/registration component, the user display component, and the administrative panel component.

The first page shown in the application is the registration component, which allows existing users to sign into their account and new users to create accounts. Unfortunately, our component is heavily dependent on the data aggregator group providing us with a working database to store user credentials and profile information. We have not received access to their database (or the details of how they are implementing their backend), so have been unable to implement proper user management. In its place, we use a naive dictionary lookup to simulate authentication.

The user page subcomponent is reached when a patient signs into the application. It uses various tools (including jQuery, Bootstrap, and amCharts) to display the user's goals in graph and progress bar form. This page allows the user to track their progress for goals set by their physician, as well as contact a physician if necessary.

The administrative panel is reached when a physician signs into the application. Here, the physician can view and interact with all of their patients. This administrative panel provides a physician with the ability to add/remove/modify user accounts, create and set user goals, and control application settings. Note: a lot of this is currently being implemented.

2. Design Considerations

2.1. Assumptions and Dependencies

Our software has a major dependency which is the data aggregator group. It is their responsibility to provide us with a database where we can get and store information for each patient. Unfortunately our team has not received any information about this database so we were unable to hook it up to our site. We also assume that our users are medical patients and our administrators are physicians but at this point in development the product can target any market.

2.2. General Constraints

The primary constraints involved with developing the cloud component were related to the inability to interact with the other subgroups to develop a proper backend for the application. In its place, we have developed an application that depicts how the application would present data and functionality to the user and administrators. Eventually, proper authentication will be implemented and dynamic user profile display will follow. This, of course, is dependent on the development of the data aggregator component.

2.3. Goals and Guidelines

One of the major goals of this project was to create a user friendly web application. Our team wanted our users (patients and doctors) to have minimal difficulty when managing and tracking goals. We provide user with the data they are seeking in multiple visual and textual formats, including graphs, progress bars, plain text percentage, and raw data.

2.4. Development Methods

Our development team followed a process that was heavily inspired by agile, including the use of scrum. The structure of scrum begins with a pre planning phase where we established when the scrum sprint would take place and what we wanted to finish within the scrum sprint. We would then work on the project during the established time and conduct daily meetings to see how far each of the group members have gotten. Finally at the end of the sprint we submit the work we had completed and reviewed ways we could improve the product in the next sprint.

3. Architectural Strategies

Originally we planned to use Amazon Web Services to host our application and database. We attempted to use the free tier of the service since the restrictions on the database would still have suited our needs. However, after a week of using this, we discovered that the instance we set up was not free, and was going to cost us too much for that week alone. The team therefore decided to stop using AWS. Instead we chose to use local JSON variables for the first deliverable to simulate data input. For communication during scrum meetings, we made video conference calls using Google Hangouts. We would've liked to meet in person a little more, but scheduling conflicts made it difficult, so online conferencing was the team's only option. Each member made their own section of the website on their own, and then all the subcomponents were sent to Alex to integrate and normalize page styling.

4. System Architecture

When developing a web application, it is appropriate to decompose the application into distinct pages or systems. The four main subsystems are the registration component, user component, administrative component, and the database. Each subsystem's main role is to be able to interact with other subsystems to facilitate data manipulation and exchange. The registration component allows our application to create users in our database and track their data in order to populate the user profiles and administrative panel. Each subsystem interacts to allows patients and doctors the to assign goals and track progress to facilitate a healthy lifestyle or to maintain a regimen.

5. Policies and Tactics

- Design must be kept simple due to nature of clientele
    - A decent portion of our clientele are the elderly and disabled either looking to exercise or go through physical therapy. This meant that we needed to keep the general design of our UI as simple and easily navigable as possible. A more complex UI might have confused the or frustrated our potential clients.
- Testing conducted by teacher

    - Though we never needed to do this, our original plan for testing our software was to bring in to our instructor for review. In class the instructor offered to act as a client stand-in, an offer we would have taken if we had thought it necessary.
    - Most testing conducted by development team throughout and after implementation process
- Maintainability through modularity
    - Most of the systems and subsystems we constructed were made as modular as possible given the time constraints. The primary functionality of the application is displaying user data, so we implemented the page such that any data can be displayed, given that the proper partial HTML file is created. With this architecture, displaying arbitrary data is possible.
- Source code organization
    - We split our source code into two main groups the structural HTML and the working sub-files (CSS, amCharts, PHP and JavaScript). Each of the working file type were then separated into their own directory to keep things organized.

6. Detailed System Design

- Login/Signup Page
    - Classification: Subsystem
    - Definition: Used by users to either register a new account with our database or sign into a pre-existing account.
    - Responsibilities: This subsystem should take in information from different form fields and use that information to allow users to access and modify their profiles. For registration, the subsystem uses inputted information to create a new profile in the database and then redirect them to their new profile page. For login, the user must have an account already registered with our database. Using that information will allow them to access their profile.
    - Constraints: Although this was not implemented, we intend to have methods to limit the number of account a user could register with our database. This would be done by looking at repeated email addresses and limiting the number of account one address is registered under.
    - Composition: The login form consists of two input fields and a submit button, this is used by users to login to their accounts. The signup form consists of five input fields and a submit button, this is used by users to register new accounts. The PHP working file are called by the aforementioned forms to access the database and user/admin pages.
    - Uses/Interactions: This subsystem uses the userpage and the admin page as redirect points. Dependent on who logs in, the login/signup page will redirect a

use to one of this pages as it's last action upon login/signup. The database is also used a great deal with this subsystem, as it needs the database in order to pull/push relevant user data.

  - Resources: Like all of the other UI subsystems this subsystem uses Bootstrap to style and manage it's looks. This subsystem also uses PHP to interact and modify the database.
  - Processing: For login: a registered user enters their information, the information is then passed by the form to a PHP page. The information entered is then checked against the information in the database making sure that the email and password entered match the email and password that is stored. If there is a match, the PHP page then redirects the user to either a user page aor admin page based on the type of account found in the database. Otherwise, if no match is found the PHP page returns them to the login/signup page and throws an error. For signup: a new user must enter the required information for create an account, if one of the fields is entered incorrectly an error occurs. Otherwise, the form throws to a PHP file which inserts the information into the database. That information is then used to populate a user/admin page.
  - Interface/Exports: This subsystem allow users to login and register accounts, which gives them access to the remaining UI subsystems of the project.
- User Page
  - Classification: Subsystem
  - Definition: Displays user sensor information in an easy to understand format. Tracks user progress for different exciting and fitness goals.
  - Responsibilities: Subsystem should take sensor data from the database and display that information for the user to see. This data is either logged by the sensor automatically or, if there is an error, manually by the user.
  - Constraints: This page does not interpret the logged data only displays it, thus if the information is interpreted incorrectly the user might be mislead about their current progress. There is not too much we can do about this as the job of interpreting the sensor data falls to the other groups.
  - Composition: The user page consists of three main parts. The first if the user bio, which displays the user's name, age, height, weight, a profile picture and what type of account the user signup for. The next part is the progress section which displays current progress on goals. Lastly we have the progress graphs section which displays current user progress on several key goals in a amChart graph.
  - Uses/Interactions: This subsystem is used by the login/signup page as a redirect point. This subsystem also uses the database to populate its various sections.

    - Resources: Like all of the other UI subsystems this subsystem uses Bootstrap to style and manage it's looks. The user page also uses amCharts to display user goal progress.
    - Processing: A user logs in with the correct credentials under a user account and is redirected to their user page. Form their the user can browse their currently logged data and track their progress.
    - Interface/Exports: This subsystem allows users to view and track their current progress.
- Administrative Panel
    - Classification: Subsystem
    - Definition: Displays application controls to authenticated administrators. Allows administrators to interact with users in many ways.
    - Responsibilities: Provides administrators with the ability to add, modify, and remove user accounts; add, modify, remove, and set user goals; and control application settings
    - Constraints: The administrative panel does not provide all functionality yet and serves as a proof of concept for what an administrator would be able to control when the application is complete.
    - Composition: The administrative panel consists of three primary views: the selection panel (allows the administrator to choose which settings to modify), the modification panel (allows the administrator to modify the selected setting), and the event log, which allows administrators to view recently completed goals, changed profiles, and any other sort of logging configured to display.
    - Uses/Interactions: This subsystem is used by the login/signup page as a redirect point. This subsystem also uses the database to populate its various sections.
    - Resources: Like all of the other UI subsystems this subsystem uses Bootstrap to style and manage it's looks. The administrative panel is simplistic by design.
    - Processing: The administrative panel's processing functionality is unimplemented.
    - Interface/Exports: This subsystem provides administrators with the ability to add/modify/remove user accounts and goals, and change application settings.
- Database
    - Classification: Third-party component
    - Definition: The database is provided by the data aggregator group and will store any and all data provided by users or administrators.
    - Responsibilities: The database will store user/administrator credentials, user profile information, goals (and progress), and application settings.
    - Constraints: The database is provided by the data aggregator group. Therefore, the database itself is an unknown constraint on the cloud component.

- Composition: Unclear. Not provided by data aggregator group.
- Uses/Interactions: Creating and using a user/administrator account involves interaction with the database. Building the user/administrative panels involves querying the database for certain data to display. Changes made to application settings will involve querying and modifying the database.
- Resources: Unclear. Not provided by data aggregator group.
- Processing: Unclear. Not provided by data aggregator group.
- Interface/Exports: Unclear. Not provided by data aggregator group.

7. System Installation and Execution

At the time of submission, we have this project running off of a simulated database. As such, the simulated database must be setup prior to running this project. First you must install a MySQL or SQL based simulator like XAMPP (https://www.apachefriends.org/index.html). When the simulator is installed, launch it and start the Apache and MySQL/SQL modules. Open a web browser and navigate to http://localhost/phpmyadmin/ to access your local SQL database manager. Either use the import tab or copy/paste the provided SQL code into the SQL tab to create the 'se_a4' database.

To get the frontend of the project working with SQL, you must place the main directory for the project in your 'htdocs' folder. This folder can be found under your XAMPP installation location. Once the director is installed in 'htdocs' you should be able to run the files from localhost (ex: localhost/yourdirectorynamehere).

The following subdirectories/files comprise the frontend and should be present in htdocs:

/css/stylesheet.css	Primary stylesheet
/css/admin.css	Admin panel stylesheet
/css/navbar.css	Navigation bar stylesheet
/css/signup.css	Registration page stylesheet
/scripts/main.js	Primary script
/scripts/signup.js	Registration script – superseded by php/signUpWork.php
/scripts/graphs.js	Graphing script
/scripts/alert.js	Custom error/warning/info alert script
/scripts/admin.js	Admin panel script
/scripts/goals.js	Goals script
/index.html	Skeleton page for loading partial HTML files
/user.html	User page – partial HTML file
/admin.html	Admin panel – partial HTML file
/signup.html	Signup page – partial HTML file
/about.html	About page – partial HTML file
/Profiles.JSON	User profile "database" (JSON file)

Individual HTML files should not be accessed by entering their address. Using the partial HTML filename as a hash works, instead. For example, http://cloud/#user will load the user profile page component, but http://cloud/user.html will load the content without the entire page. The registration script, signup.js, should be used when the database is not in place. signUpWork.php should be used in its place when the database is present. Security is nonexistent without a database.

Once these files are in place and the backend is working, authenticating as a user or administrator will redirect appropriately.