CIS 573 Software Engineering – Summer 2021
Project Setup Guide

This guide will help you set up the database and web server for the project using:

- **Node.js** for the JavaScript runtime environment
- **Express** for handling basic web server functionality
- **BodyParser** for parsing form data in a POST request
- **EJS** for dynamically generating HTML content
- **Mongoose** for accessing the database
- **MongoDB** for storing data

This document also describes how to use the Administrator app and also set up and use the Organization (command-line) and Contributor (Android) apps.

These instructions were written for use on a Mac/Linux platform. You may need to make slight modifications for using Windows.

You are strongly urged to get your project environment set up and installed by Tuesday, July 6, so that you can begin working on Phase 1 tasks, which are due a week later. If you run into any trouble, please post a **public** note in Piazza so that your classmates and TAs can help you, or visit a member of the instruction staff during office hours.

Step 1. Download App Code

The project starter code is given in the GitHub repo and you should create a fork of this repo to work off of. To do this, you should click "Fork" on the top right of the repository page. You can read more about working with forks here.

Then clone/download the repo to your computer so that you have a local copy to use and to run.

Step 2. Install and Configure Node Express App

The software system in this project depends on Node Express for acting as a mediary between the Java applications and the database, and also for hosting the Administrator app and generating dynamic HTML content.

You can learn more about Node Express here but the instructions below should be sufficient to get it installed and set up so that you can run the apps on your computer.

1. Install Node.js
   - Go to **https://nodejs.org/en/download/** and follow the instructions for your platform

- Once you've installed it, open Terminal/Command Prompt and check that installation is correct by running the command: **`node -v`**
- Make sure all modules are up to date using: **`npm install npm -g`**
- You may need to have administrative privileges on your system to run these commands, e.g on Mac/Linux you can do: **`sudo npm install npm -g`**

2. Install Node packages.
   - Navigate to the **admin** folder of the project that you downloaded in Step 1.
   - Run **`npm install`**.
   - If everything worked correctly, you should now see a **`node_modules`** folder within the admin folder.

Step 3. Install MongoDB

The system uses MongoDB for storing all data. You will need to install a database instance for the system to use.

Go to https://www.mongodb.com/try/download/community and follow the instructions for your platform to install MongoDB Community Server.

If you cannot install MongoDB on your computer, you may want to consider a cloud platform such as https://www.mongodb.com/cloud/atlas ; the free version should be sufficient for our purposes.

Once your project gets under way, you may want to set up a shared Mongo instance that is used by you and your teammates, so that you can share data. You can use Mongo Atlas for that and it should be a simple change to your database configuration file, but for now we recommend that each student have their own instance.

Step 4. Start the Admin App

First you need to start MongoDB. From Terminal/Command Prompt, navigate to the directory in which you installed MongoDB in Step 3 and run the following:
Mac: **`./bin/mongod --dbpath [path to db directory]`**
Linux: **`mongod --dbpath [path to db directory]`**
Windows: **`.\bin\mongod --dbpath [path to db directory]`**

A few things to note:
- the [path to db directory] must refer to a directory that already exists and is where you will store the data for Mongo. It can be an absolute path or a relative path
- Be sure to use two hyphens (two minus signs) before "dbpath"

When you start MongoDB, you'll see a number of messages written to the console, and the last one that you see should include "msg": "Waiting for connections" and something about port 27017. If you receive an error, please consult the MongoDB documentation at https://docs.mongodb.com/manual/

Once MongoDB has started, open another Terminal/Command Prompt window, navigate to the **admin** directory of the project that you downloaded in Step 1, and run the Admin server using the command:

```
node admin.js
```

You should see "Listening on port 3000" meaning that the web server is running and is awaiting incoming HTTP requests on port 3000.

Step 5. Access the Admin App

Now you can access the Admin webapp via a browser. Assuming you're opening the browser on the same computer where you started Node, open http://localhost:3000 and you should see the app. Yes, we know it's ugly…

Click "Administer Organizations" to view a list of all the organizations in the database. At first, because there are no organizations, you should click the "Create New Organization" link to create a new one, and specify its name, the login ID and password to be used in the app for organizations, and a description. Note that no error handling is done on these fields so enter the data carefully!

When viewing an organization, you can edit or delete it, and view its funds (or campaigns). At first, there will be no funds, so you can create one by entering its name, description, and target amount, which should be a positive number, though this is not enforced in the app.

When viewing a fund, you can edit or delete it, or make a new donation from one of the contributors. However, since there are no contributors at first, you will need to create some accounts.

You can do this by clicking "View All Organizations" and then, from there, click "Administer Contributors," or just navigate to http://localhost:3000/allContributors

From there, you will see a list of all contributors, and you can view, edit, or delete them. At first, there will be no contributors, so you can create one by entering the name, the login ID and password to be used in the app for contributors, an email address, and credit card info ("CVV" is the three- or four-number security card on the back of the card). You do not need to enter real info, though, as no transactions are actually made in this system. As above, note that no error handling is done on these fields so enter the data carefully!

In addition to the fact that the Admin app does not do error handling of input fields, the rest of the system is not guaranteed to work if data is missing for an organization, fund, or contributor, so be sure that you do not leave any fields blank!

Step 6. Start the RESTful API Server

To make the data in Mongo available to the Contributor (Android) app and the Organization (command-line) app, run the RESTful API server from the **admin** project directory using the command:

```
node api.js
```

You should see "Listening on port 3001." This is a separate web server from the one you started in Step 4: although it also awaits incoming HTTP requests, it sends back data in JSON format rather than content in HTML.

Step 7. Set Up Organization App

For the Organization (command-line) app, navigate to the org folder. This is just a regular Java application and the main method is in the UserInterface class. However, you will need to use the JSON-Simple library in order to parse the JSON data that comes back from the RESTful API, so be sure that json-simple-1.1.1.jar (available in the project repo) is in your classpath when you run your program.

To run the program, you should run the **`UserInterface`** class, and pass the login ID and password that you created for an organization via the Admin app as the runtime arguments to the main method. Then you will be able to view funds and donations, and create new funds. Note that the app does not do any error handling regarding input options so enter the data carefully!

This app uses the "N-Tier Architecture" design as follows:

- the UserInterface class represents the UI Tier and contains all code for interacting with the user via the command-line, as well as the program's main method
- the DataManager class represents the Processor Tier and contains all code for processing data in the program
- the WebClient class represents the Data Management Tier and has a method for interacting with the RESTful API by sending HTTP requests and receiving JSON objects in HTTP responses
- the Organization, Fund, and Donation classes represent the data shared by the tiers
- the DataManager_createFund_Test class is a JUnit class that contains tests for the DataManager.createFund method

Step 8. Set Up Contributor App (optional)

Note that working with the Contributor (Android) app is *not* required for all students in the class, and a group of three is unlikely to need to work with this app at all; a group of four will most likely need to do some work with it.

In order to use the Contributor app, you will first need to install Android Studio and create an AVD with API level 28 or higher.

Then, start Android Studio and select "open existing Android Studio project" and point it at the contributor directory of the project. You may get some warning messages about directories and needing to install some packages, but Android Studio should be able to figure out everything you need.

Once the project builds, run it in an AVD, and then log in using the login ID/password credentials that you created for a contributor via the Admin app to log in. You can then make donations to a fund, though be sure that you have created at least one organization and one fund using the Admin app prior to running the Android app.

This app uses the "N-Tier Architecture" design as follows:

- the MainActivity, MenuActivity, ViewDonationsActivity, and MakeDonationActivity classes comprise the UI tier and contain all code for interacting with the user, e.g. handling button clicks and updating the content displayed in the Views in the classes' Layouts
- the DataManager class represents the Processor tier and contains all code for processing data in the program
- the WebClient class represents the Data Management tier and has a method for interacting with the RESTful API by sending HTTP requests and receiving JSON objects in HTTP responses
- the Organization, Fund, Contributor, and Donation classes represent the data shared by the tiers
- the DataManager_getFundName_Test class is a JUnit class that contains tests for the DataManager.getFundName method