Andrew Hammon 27 April 2021

Ian Brinkerhoff

Dustin Henson

Jacob Gonzales

Jared Darnell

Student Time Tracker

# Summary and Hand-Off Documentation

This application uses a Postgres (SQL) database to store data that is communicated through a C# (ASP.NET Core) back end to an Angular 1.6.6 driven front end which uses Bootstrap 4 styling. \*\*\*AngularJS will be deprecated on 30 June 2021, after which Google will no longer maintain the library. This was discovered by the team in April 2021 and was therefore not incorporated into the Spring 2021 semester goals. This project will need to be updated to the new Angular framework and possibly from Javascript to Typescript depending on Brad’s wishes. This will drastically affect the front end code and will require a complete overhaul of the current .html and .js files.

The database is accessed from functions in the DataAccess class, located in the Models folder. These functions are called from the HomeController controller in the Controllers folder. This controller contains all of the REST endpoints that are then hit via asynchronous requests from the Angular front end.

The basis of the front end starts in the \_Layout.cshtml file in the Views/Shared folder. This file is the base HTML file that contains references to Javascript and CSS libraries, as well as references all the Angular scripts.(This will need to be changed with new angular) The Angular scripts and controllers are located in the wwwroot/js folder, and use HTML templates that are located in the wwwroot/templates folder. Some of our libraries are store in the wwwroot/lib folder, but most are pulled in from CDN hosting sites referenced on the \_Layout.cshtml file. The site.css file located in wwwroot/css was used for all custom CSS styles.

## Useful Pluralsight Links

1. To Learn about what docker is. ~ 1.5 hours
   1. <https://app.pluralsight.com/library/courses/docker-kubernetes-big-picture/table-of-contents>
2. To learn about what MVC is. ~ 6 hours
   1. <https://app.pluralsight.com/library/courses/building-aspdotnet-core-mvc-web-applications/table-of-contents>
3. To learn about the new angular. (Projects angular won’t match this video) ~ 5 hours
   1. <https://app.pluralsight.com/library/courses/angular-2-getting-started-update/table-of-contents>
4. To learn how MVC and Angular are put together. ~ 10 hours
   1. <https://www.pluralsight.com/courses/aspnetcore-mvc-efcore-bootstrap-angular-web>
   2. (This pluralsight course almost describes the project perfectly besides the new angular using typescript. You could probably skip the other videos and still understand the project with this alone. The other videos are for more fundamental learning, anyone with absolutely no knowledge of what this project is. This would be extremely helpful in creating a web application from the ground up. The only caveat is this course utilizes Razor pages and Brad informed us that he didn’t want to use razor pages.)

## The Server

Server IP Addresses: 137.190.19.20 <- icarus || 137.190.18.16 <- csproject server

The Time Tracker Application will be hosted on a WSU server specifically set aside for Computer Science Projects. This is a Linux Ubuntu server that is going to be set up with Docker and Kubernetes to allow multiple students to host multiple CS projects at a time.

In order to log on to the CS Server you must first get behind Weber State’s firewall. There are a few ways to accomplish this.

Option 1: Get onto Icarus, then use Icarus to get onto the CS Server

You will first need to ssh into Icarus.

Icarus Default Login information:

Hypothetical User: John Smith W0123456789

Username: js56789

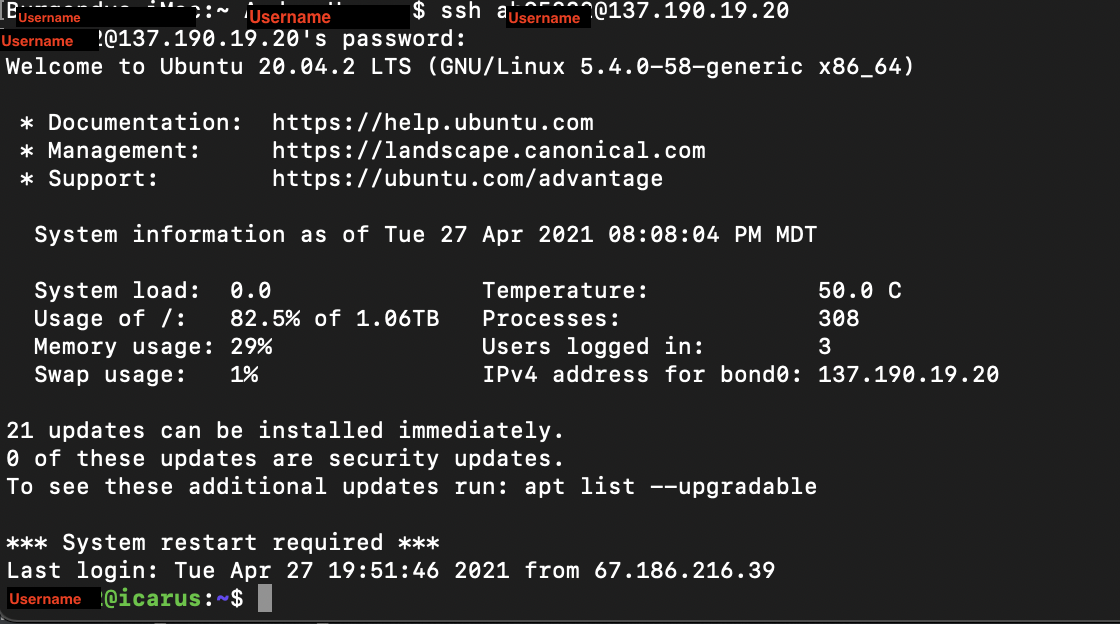
Password: js56789cs! \*\*You should change your password

The command for this is:

ssh js56789@137.190.19.20 \*(if successful, you will get a password prompt)

Password: js56789cs!

When you’ve successfully logged in you will see a screen similar to this:



Now that you are logged into Icarus. You need to ssh into the CS Server

You will need to get a login and password from administration prior to logging in.

(Brad Peterson)

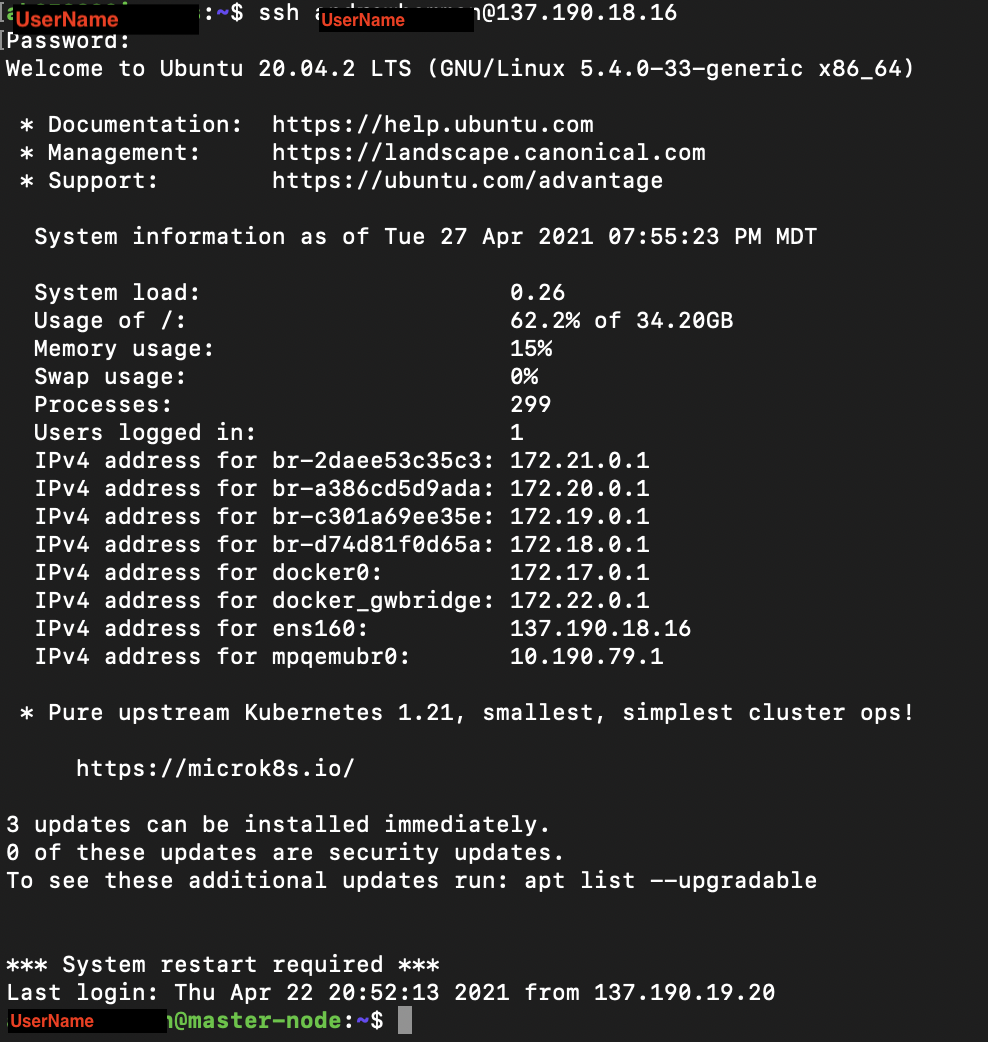
When you have a username and password the commands are the same as icarus.

CS Server Login Information:

username@137.190.18.16

Password:

When successful you will have a screen similar to this:



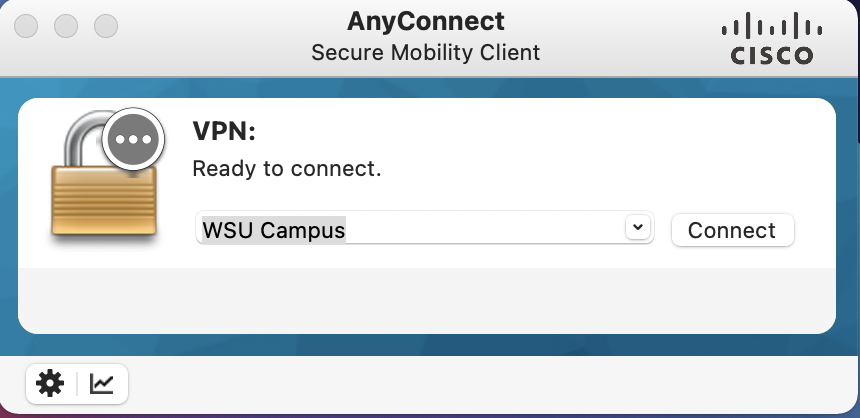
Option 2: Sign on to Weber States VPN then ssh onto the CS Server

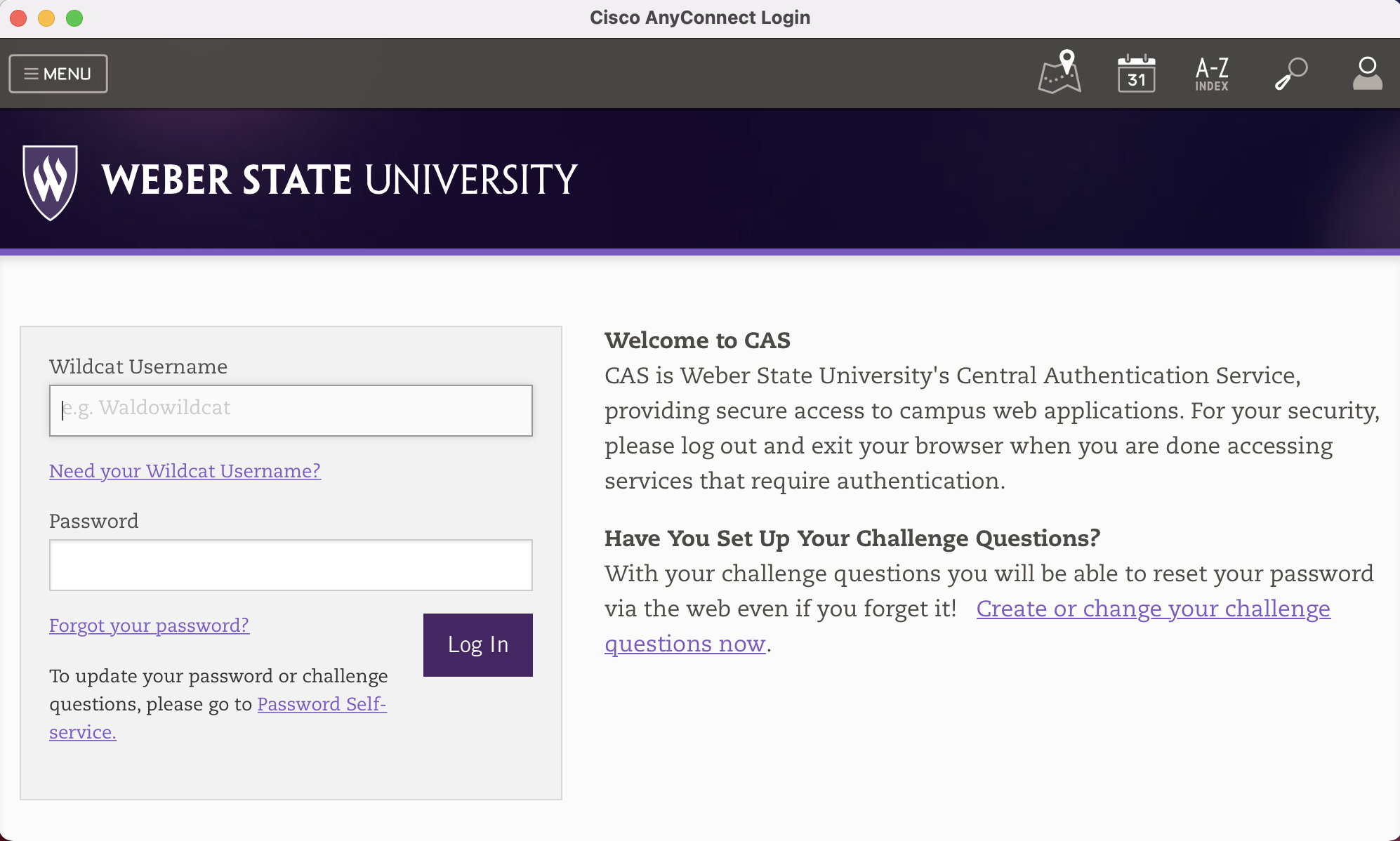
Download the vpn from: <https://www.weber.edu/help/kb/VPN_Install.html>

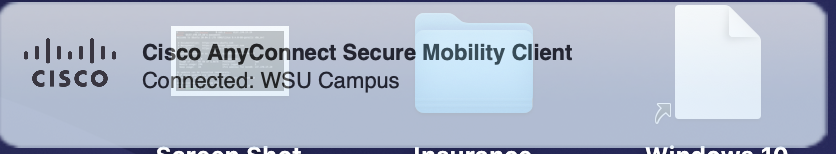
Once you have finished those steps the VPN app should be:

Cisco AnyConnect Secure Mobility Client

Open the Cisco AnyConnect App

Click Connect

 You will then sign into your WSU account



You should get a notification that you are now connected.

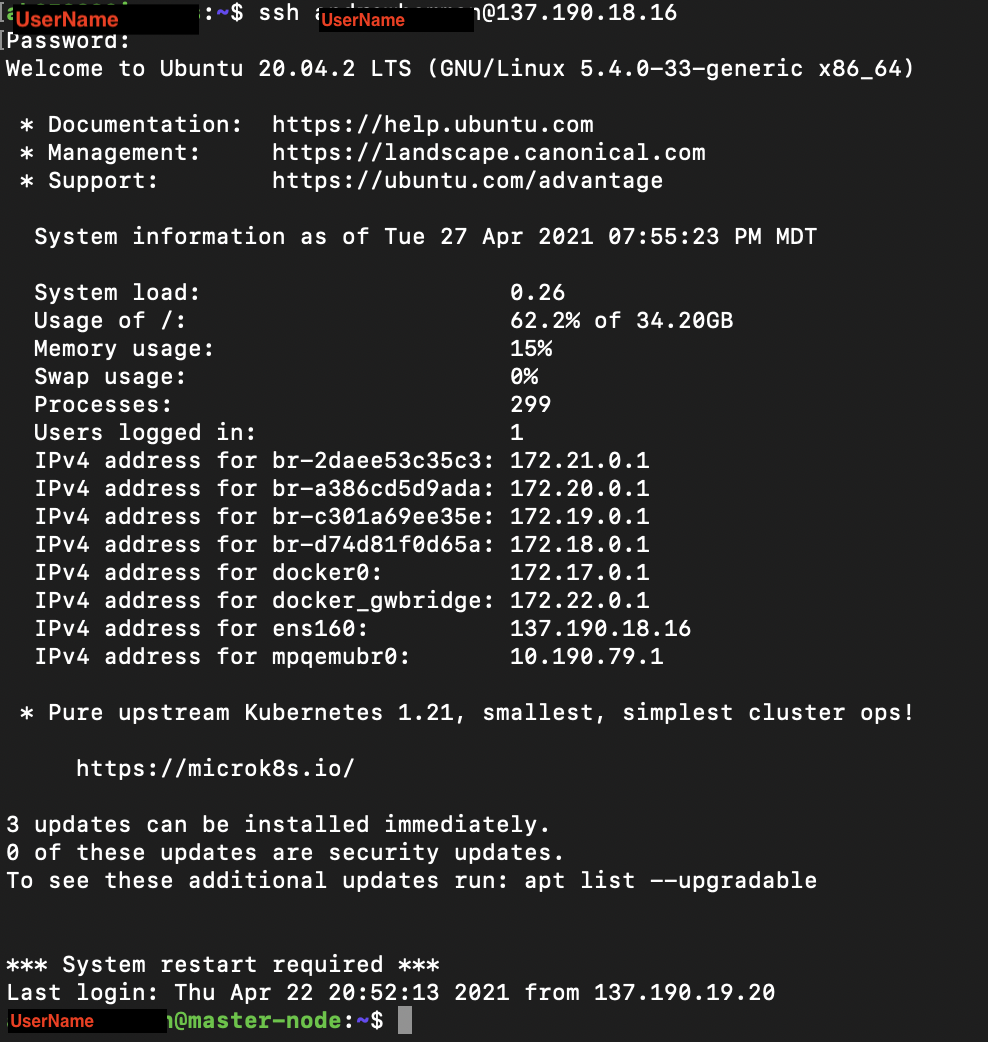
Now that you are on the VPN you should be able to ssh into the CS Server

CS Server Login Information:

username@137.190.18.16

Password:

When successful you will have a screen similar to this:



### Docker

#### Useful Docker Commands:

docker-build

* This command will build the docker image from the instructions in the docker-compose.yml file.

docker-compose up

* This docker command will take the instructions, store them in the docker-compose.yml file and use them to **build** and **run** your new docker image. If you want to run it detached, not with an open console, add a -d flag at the end. **docker-compose up -d**

docker ps

* This command will give you a list of all docker images that are running
  + Add ‘-a’ to get a list of all docker images

docker stop ‘container-id’

* This command will stop a container that is currently running, gracefully

docker kill ‘container-id’

* this command will force close a container that is currently running. Not graceful

docker inspect ‘container-id’

* This command will give you the details about a container including internal ipaddress

docker run -d -p 80:80 ‘containerName’ /usr/sbin/apache2ctl -D FOREGROUND

* This command will start hosting a named container, ‘-p’ on port 80(public) directed to port 80(internal) ‘-d’ running detached, ‘-D’ the docker container is also detached.

#### Useful Command Line Commands:

cd - change directory

cd .. - move back one directory

ls - list the contents of your current directory

curl localhost - shows the html of what is currently hosted on port 80 or 443

curl localhost:’port’ - shows the html of what is currently hosted on a specific port

curl ‘ip address’ - shows html of hosted application or site

exit - ends your current ssh connection

logout - ends your current ssh session

sudo - gives you the root authority

nano ‘filename’ - opens a basic text editor for you to edit files

mk ‘directoryName’ - Make a new directory

rm ‘fileName’ - Delete the named file or directory

sudo apt-get update - Update software

clear - clear the current terminal or command line window

lsof -i TCP|fgrep LISTEN - this command will display a list of open TCP ports

### The Server Things to be done

Docker is up and running on the server and we have been able to successfully get 1 program at a time hosted live on port 80. We have been able to get multiple projects hosted using other ports internally. However, we have not been able to get the public port 80/443 to host multiple projects at a time.

Items to be completed

* Get the reverse proxy set up and correctly receiving incoming requests and redirecting them to the appropriate docker container for that specific request.
* Figure out how different URLs are handled.
  + Does the generic csprojects.weber.edu work? If so, what needs to be appended to the end for a specific project? Example: csprojects.weber.edu/timetracker
  + Or does every project need to have it’s own URL, and we need to contact a domain name server (DNS) to register each unique URL.
  + If each project has their own unique URL do they each get their own ip address?
* Get permissions set up so that students cannot alter one anothers projects
  + See Linux Server Permission Handling below.

#### Linux Server Permission Handling

Because we're hosting multiple student projects on the server, care should be taken to make sure we follow FERPA requirements, including making sure student information is protected. Speak with your professor about how permissions should be handled on the server.

When you're connected to a linux environment, you can type "ls -l" and see a list of all files and directories in your current directory along with it’s permissions, which may look like this:

drwxrwxr-x 4 johnsmith somegroup 4096 Mar 25 18:03 myDirectory

drwxrwxr-x tells you what the permission settings for the given file/directory is, and can be broken down to four parts -> [d][rwx][rwx][r-x]. The first part indicates if it's a directory. ‘D’ means directory, ‘-’ means file. The second set is the permissions the owner has. 'rwx' means the owner has read/write/execute access. A ‘-’ in any place means that permission is missing. The third set indicates permissions for the associated group, and fourth set indicates permissions for anyone else. ‘Johnsmith’ is the owner, and ‘somegroup’ is the associated group.

"getent groups" lists all groups known to the system.

"groupadd <groupname>" adds a group with the given name to the system.

"groupdel <groupname>" deletes the group with the given name from the system.

"usermod -a -G <groupname> <username>" adds a user to the given group.

"chown <username> <file>" changes who owns a file

"chown :<groupname> <file>" changes what gruop is associated with the file (note that the colon there is necessary).

> alternatively, you can use "chgrp <groupname> <file>" to change the associated group.

"chown -R <username>:<groupname> <directory>" changes user and group association recursively for the directory and all contents

“chmod <permissions> <file/directory>” Sets the permissions of the file or directory to the given permissions. Use the -R option for directories to recursively affect all contents.

* <permissions> can either be either a three digit number to indicate permission control, or it can be symbol based to set permissions.
  + If using a three digit number to set the <permissions> argument, note that each digit represents user, group, and other respectively. Each digit is the sum of the permissions, represented as octals, where Read=4, Write=2, and Execute=1. For example, if you use 740 for your permission, then the user owner will have full access, the group will have read access, and others will have none.
  + If using symbols, you can use <u/g/o><+/-/=><rwx>. The first set says what group is affected, the second set decides if you’re adding, removing, or setting permissions, and the final set says what permissions are affected.
    - u=rwx: sets the user owner permissions to read write execute
    - o-wx: removes write and execute access from others
    - u=rwx,o-wx: combines the functionality of both

## The Website Design

The project is designed via MVC (Model View Controller). The models folder holds all tables and row information. The controller folder is responsible for data retrieval and input. The DTOs are also a part of the controller. The view is the front end part that the user sees to interact with the website.

## The Front End

The front end is currently AngularJS 1.6.6. AngularJS is losing Long Term Support (LTS) June 2021. This means that the front end code will need to be updated to the new version of Angular which is currently Angular 11.

This will require rewriting the .html and .js pages that are used for the front end and updating them with Angular TypeScript. The .html files are currently used for design, they take users input and send the data to the .js file. At this point the data is either manipulated on the front-end or sent to the server for data manipulation. The AngularJS will only affect the front-end code and the backend code should largely stay the same.

The front end is the view part of the MVC design. Simply put the communication between the user and the website.

Below is the description of AngularJS that was given to us at the beginning of the Spring 2021 semester:

*“The Angular app uses a nested controller structure, with the MasterCtrl being the parent controller . This controller stores the logged in user information that is referenced through the rest of the app. A viewport within the MasterCtrl is then used to display each page based on the route (URL). The route definitions, including which controller and HTML template to use, are located in the app.js file. (This will need to be changed with new angular)*

*The front end of the application is almost entirely finished, but not all the endpoints are defined on the back end yet. To avoid hitting invalid endpoints, these are all commented out with TODO comments, and are usually followed by some dummy data to allow a user to see what a page should look like. The database must be set up through docker to be functional.”*

## The Back End

The back end is responsible for storing the session information of the logged in user, handling data, and facilitates communication between the front end and back end.

The session verifies that the user has permission to perform any requests from the front end, after verification, the user is allowed to modify information based on his level. The session also allows the user to stay logged in if the page is unexpectedly closed.

The backend gets data from the frontend, manipulates it, and executes the functions to communicate with the database. That information is then sent to the front end to parse and use for the user to see.   
  
 The password should also be hashed in the back end before it is stored or checked. The back end should send No Content (204) responses to the front end for certain failed requests, or send a failure code. No Content is automatically sent if the back end returns null to a request.

## The Database

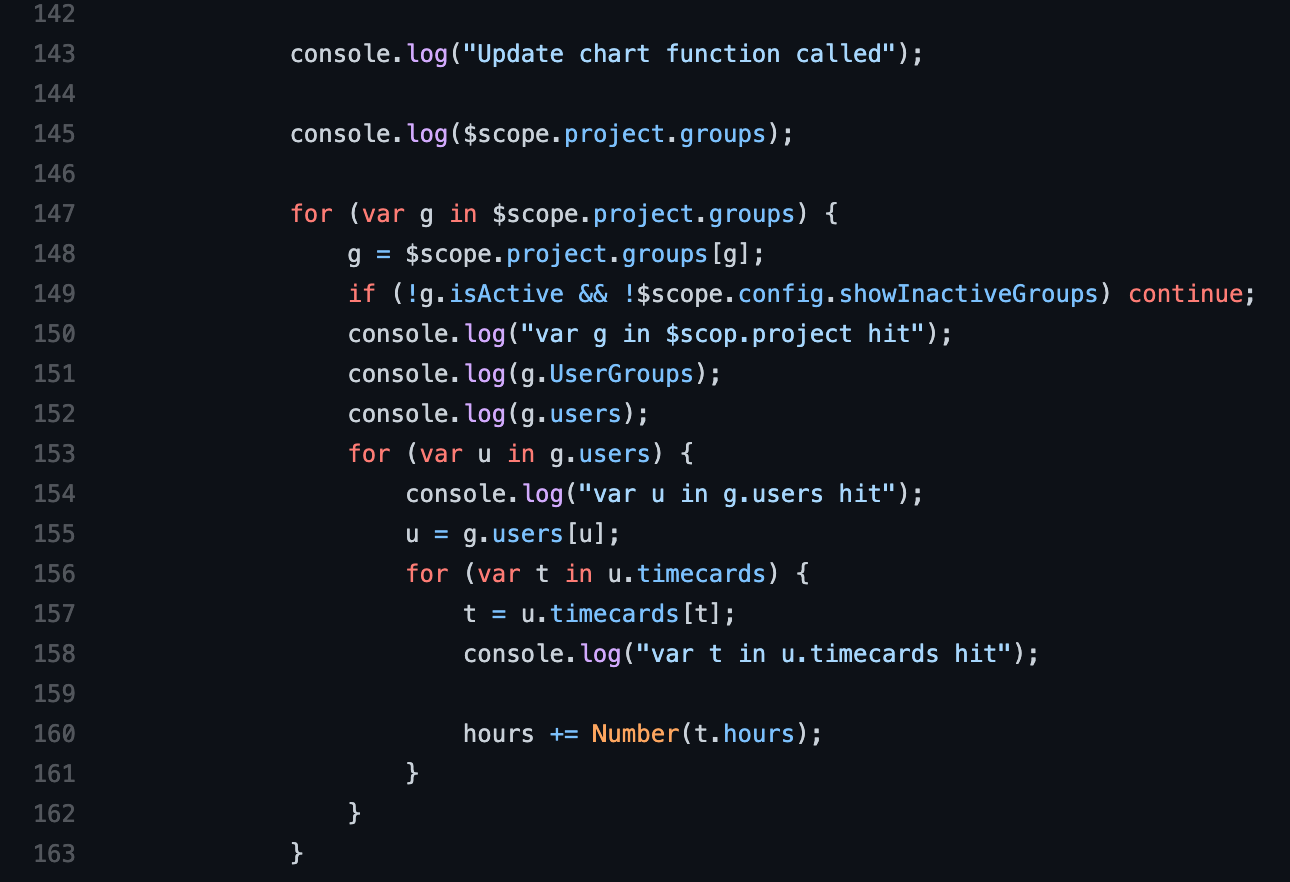
To get this database running you must set up through Docker. When setup the postgres is the specific file that will be holding the database. In the initial setup there are 3 users already set up.

* Admin User
  + Username: admin
  + Password: Password!
* Instructor User
  + Username: instructor
  + Password: Password!
* Standard User
  + Username: user
  + Password: Password!

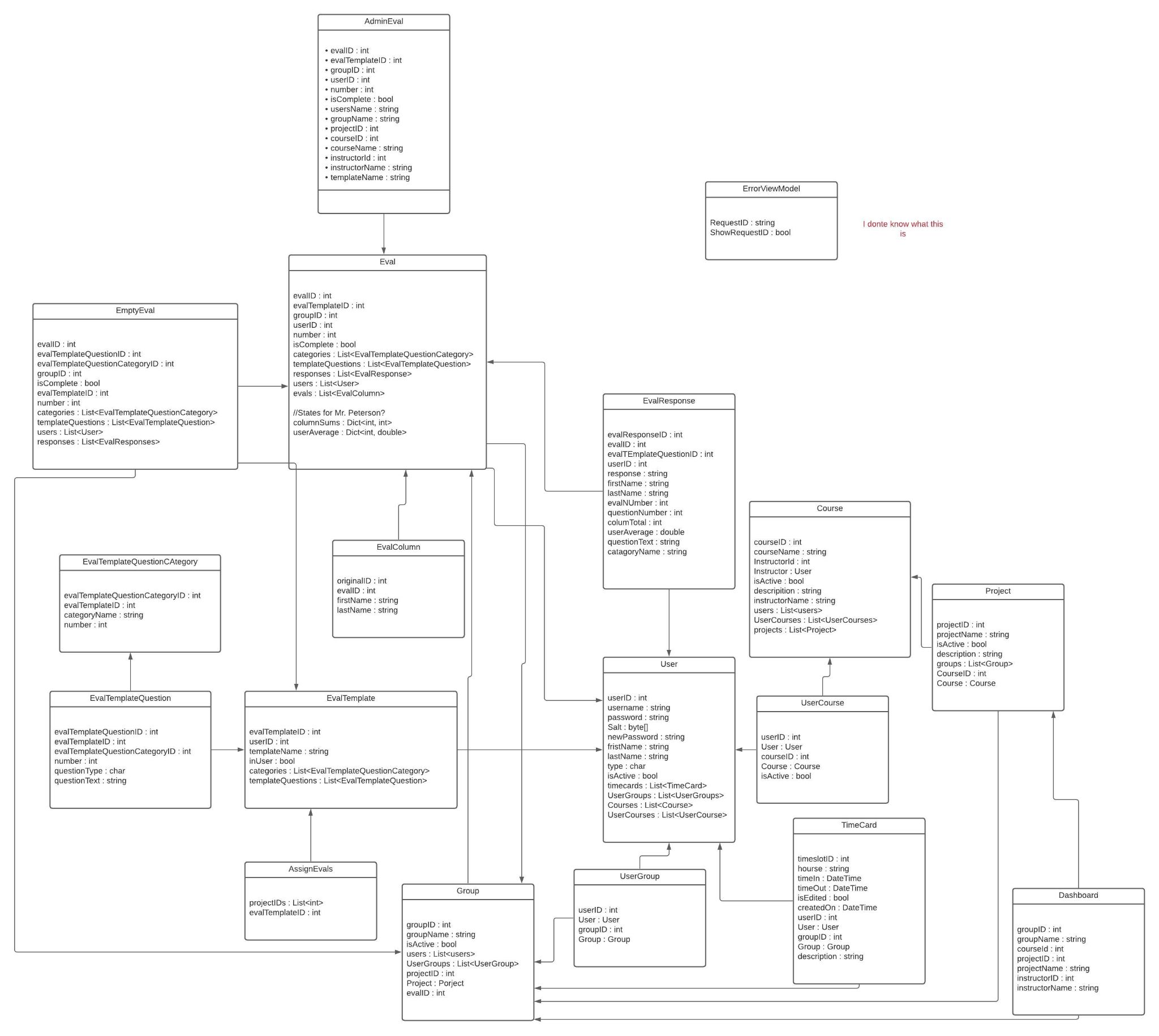
##### DATABASE STRUCTURE

For this database structure we drew up a UML to fully explain all that is happening. The issue with this database is not all the information is being distributed.

One of the errors is for the project page. In the project.controller.js in lines 147 - 163 is a nested for loop to load a pie chart with users. The issue being is that users in the group come up null. Which will never load the pie chart of times to display. So the database structure needs a re doing to make sure all data is shared and distributed.



\*\*\*Something to note, this database is somewhat complicated. There are tables that can’t fully be explained. A quick clean up of the database to remove any unnecessary rows or tables might be needed moving forward.



#### Unfinished Business

The following items are finished and working:

* User registration
* User login
* Getting all Courses
* Creating a Course
* Getting a Course and its associated Projects
* Creating a Project
* Updating changes to a Course’s name and active status
* Updating changes to a Project’s name and active status
* Users joining a Course
* Allowing an instructor to approve or unapprove a User in a Course
* Users creating and joining a Group on a Project
* Getting all the additional information on a Project, including the Users and their Time information

This leaves quite a bit unfinished:

* Clean up database (get rid of unused rows)
* Update front-end from AngularJS to Angular
* Users can edit time but anything past 2 weeks will be blocked in the TimeService.cs. There needs to be a way for the C# to communicate with the front end to notify the user that they cannot change anything past 2 weeks.
* This code will block anything past 2 weeks being editable. (TimeService.cs lines 27 - 59)

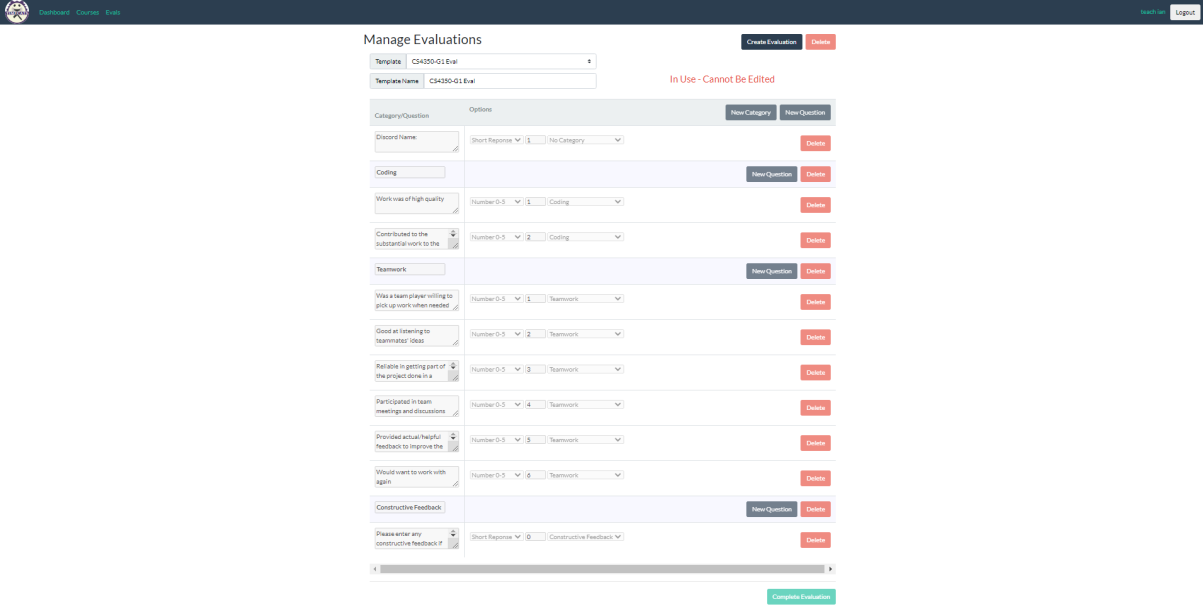


* The Admin and Teachers can delete times but they need to be able to edit times. Including anything past 2 weeks.
* Getting an individual User’s and then allowing them to edit it.
* Getting all Users for the instructor’s Users page, and allowing the instructor to edit whether users are active or administrators
* Making sure that any user referenced is active
* Making sure that there aren’t duplicate usernames in the system
* Evaluation needs listed below:

Information regarding Evaluation functionality (APRIL 2021):

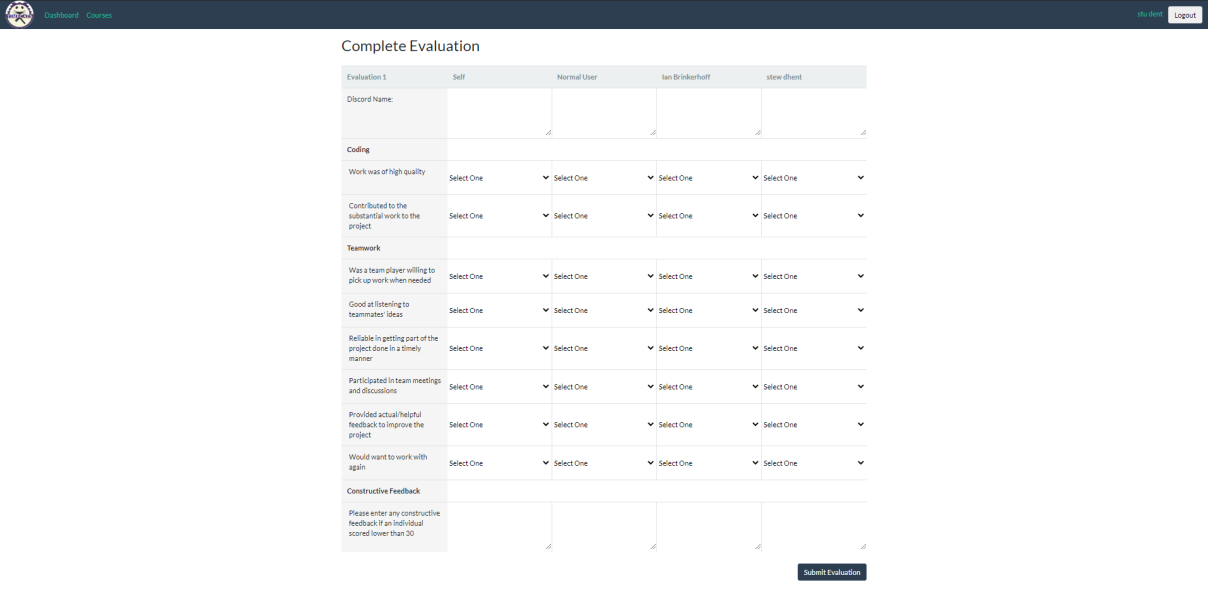
* Current functionality:
  + Instructors can create an evaluation

\*below is a screen snip of an instructor view of an evaluation



* + - CRUD ability for questions and categories
    - After the evaluation is completed an instructor can assign the evaluation to a particular group or project (This is done in the course page)
    - After the evaluation is assigned to the students, the students can access the evaluation from the course page.
      * The evaluation is dynamically updated with the additional users in the group.

\*below is a screen snip of a student view of an evaluation

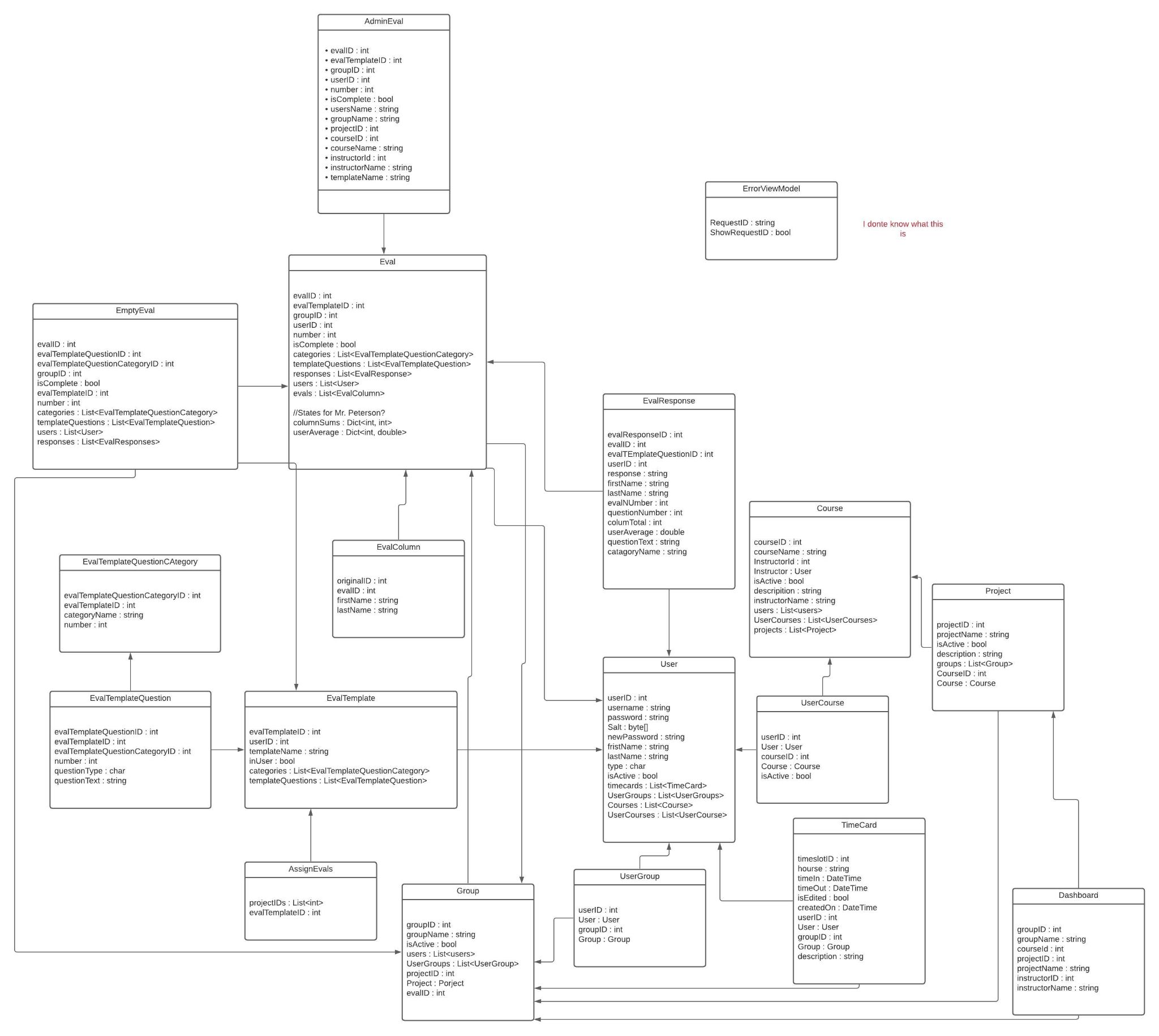


* After the student rates their group mates, they submit the evaluation to the instructor
* The instructor can go to the group page and view all members evals that are completed

Evaluation functionality that needs work:

* Total and Average-
  + Total - eval questions allow a student to rate their teammates on certain topics from 1-5, total is the sum of all ratings. Upon selecting a rating for a user on a specific question, the users Total should auto increment. This functionality is currently in place but is very finicky and needs refinement.
  + Average - This functionality is not implemented yet at all. This would be an additional row on the evaluation, preferably below the total row, that auto incremented to reflect the teammates average rating. This would auto calculate using the Total(listed above) / number of questions
* Instructor reviews finished evaluations-
  + This functionality currently exists but is very finicky. Currently, all completed evaluations show up under one view. This means that if a student completes an eval for groupA and an eval for groupB, they are currently appearing on the same table. They will need to be separated.
* The evaluation functionality as a whole will require a lot of refinement but a good chunk of the functionality is still there. If there are any questions on this in the next semester or anything needs to be clarified, reach out to me -ianbrink10@gmail.com.

## Tables - Located in Models Folder



## Pages Needed - Functional Requirements

Layout examples - (See Screenshots section as these pages have now been built)

* Login - sends/receives User object
* Register - sends/receives User object
* Dashboard - receives list of active Groups (including Project Name, Course Name, Instructor Name)
  + Lists all active Groups that the User is a member of
* Courses - receives list of active Courses (Including Instructor name)
  + Lists all active Courses with Instructor name
* Course View - receives a Course with a list of all active Projects and a list of all active Users accepted and pending for the course
  + Shows all Projects within a Course
  + (Instructor) Ability to add new Projects to the Course
  + (Instructor) Show all Users (possibly separate tab), allow to activate requested Users
* Project View - receives a Project with a list of all Groups, and the total hours of time for each of those groups
  + Show all Groups associated with the Project
  + Allow any User to make a new Group (proceed to Group view)
  + Pie chart of weighted Group Times for the Project
* Group View - Receives a Group with a list of all Users and all Times associated with the Group
  + Show all Users that are members of the Group
    - (Instructor) Ability to remove User from Group
  + Pie chart of member Times
  + Table of group member Times
  + Ability for Users to add/edit *their* Times
  + If Time is edited after a certain amount of time (a week?) set the isEdited flag to true
* Users View – receives all Users in the system (Instructor only page)
  + Ability for Instructors to change activation status of users, or elevate users to be Instructors
* User View – Receives details for a single User
  + Allows that user to edit their details or change their password
  + Allows an Instructor to edit the user’s details, change their password, and change the activation and instructor status

Note: We need to make sure that all checks verify that the user is logged in and is an instructor for actions that matter. Then users can’t mess with the Javascript to do things they shouldn’t be able to do. Additionally, users that aren’t accepted into a project course shouldn’t be able to join groups within it.

## REST Endpoints

Test JSON User

{

“\_id”:“5ad54b26fb49e95bf06a1c92”,

“username”: ”logan”,

“firstName”: ”Logan”,

“lastName”: ”Brown”,

“password”: ”12345”, (only include when sent for login, and as a hashed value)

“isInstructor”: false,

“isActive”: true

}

-----------------------------------------------------------------------------------------------------------

Endpoints reference a function in the HomeController.cs file. (/Home/Login maps to the Login() function on the Home controller)

What we need from the database at different points:

* User (/Home/Login - will receive username and hashed password which should be hashed again on the backend before it is checked again the DB, also check that the user account is active)
  + Individual user that matches provided username and password with
    - username
    - firstName
    - lastName
    - isInstructor
* Dashboard (/Home/Dashboard - use userID from session)
  + All Groups associated with current (logged in) user with
    - groupID
    - Name
    - If it isn’t too much to ask.. Project name, course name, and instructor name too, but that might be too hard?
* Courses (/Home/Courses)
  + All courses with
    - courseID
    - name
    - Instructor’s name
    - isActive
* Course (/Home/Course - will receive courseID)
  + Individual course with
    - courseID
    - name
    - isActive
    - Projects associated with that course with
      * projectID
      * isActive
    - Users associated with that course with
      * firstName
      * lastName
      * isActive (meaning they are accepted within the course, should be a flag at the course level)
* Project (/Home/Project - will receive projectID, ensure logged in user is active in the course)
  + Individual project with
    - projectID
    - name
    - isActive
    - Groups associated with that project with
      * groupID
      * name
      * isActive
      * All users associated with that group with
        + userID
        + All time associated with that User with

Hours

* Group(/Home/Group - will receive groupID, ensure logged in user is active in the associated course)
  + Individual group with
    - groupID
    - name
    - isActive
    - All users associated with that group with
      * userID
      * firstName
      * lastName
      * All time associated with that User with
        + userID
        + timeID
        + timeIn
        + timeOut
        + Hours
        + isEdited
* Users (bonus gravy - not high priority) (/Home/Users)
  + All users in the system with
    - userID
    - username
    - firstName
    - lastName
    - isInstructor
* User (bonus gravy - not high priority) (/Home/User)

Additional Endpoints - Located in HomeController.cs

* /Home/CheckSession
  + If there is someone logged in, return the User with
    - \_id
    - firstName
    - lastName
    - isInstructor
* /Home/CheckInstructor
  + If there is someone logged in (Assumes there is), return a bool indicating if they are an instructor
    - True
    - False
* /Home/RegisterUser
  + Will receive user information and create a user in the database, doesn’t need to return anything more than success or failure HTTP code (separate login call is made afterward to /Home/Login)
  + Will be sent:
    - username
    - firstName
    - lastName
    - password (hashed value, hash it again on the backend before saving it)
* /Home/SaveCourse
  + Will receive a course object as JSON, should save the course in the database according to courseID
  + Should verify that logged in user is an instructor
* /Home/SaveProject
  + Will receive a project object as JSON, should save the project in the database according to projectID
  + Should verify that logged in user is an instructor
* /Home/SaveGroup
  + Will receive a group object as JSON, should save the group in the database according to groupID
  + Should verify that logged in user is part of the group
* /Home/SaveTime
  + Will receive time object as JSON, should save the time object in the database according to timeID (should already be created), ensure that the userID of the time object in the database matches the logged in userID, or that the logged in user is an instructor)
* /Home/CreateCourse
  + Should create a new course in the database and return its courseID
  + Set a default value for course name like “New Course” and isActive should be true, and the associated instructor for the course should be the currently logged in user
  + Should verify that logged in user is an instructor
* /Home/CreateProject
  + Receives a courseID and should create a new project in the database as part of that group and return its projectID
  + Set a default value for project name like “New Project” and isActive should be true
  + Should verify that logged in user is an instructor
* /Home/CreateGroup
  + Receives a projectID and should create a new group in the database as part of that project and return its groupID
  + Set a default value for group name like “New Group” and isActive should be true
* /Home/CreateTime
  + Receives a userID and groupID, and should then create a new time object in the database with that userID, and return its timeID
* /Home/JoinCourse
  + Will receive a courseID, should add the logged in user to that course, with an isActive value set to false
* /Home/JoinGroup
  + Will receive a groupID, should add the logged in user to that group
* /Home/SaveUser (bonus gravy)
  + Will receive a user object to update in the database
* /Home/ChangePassword (bonus gravy)
  + Will receive user object with old and new password, old password empty if it’s an instructor and they don’t need to match the old password. If not an instructor, must match the old password and then update to the new password

## 

## 

## Production - Hosted Project

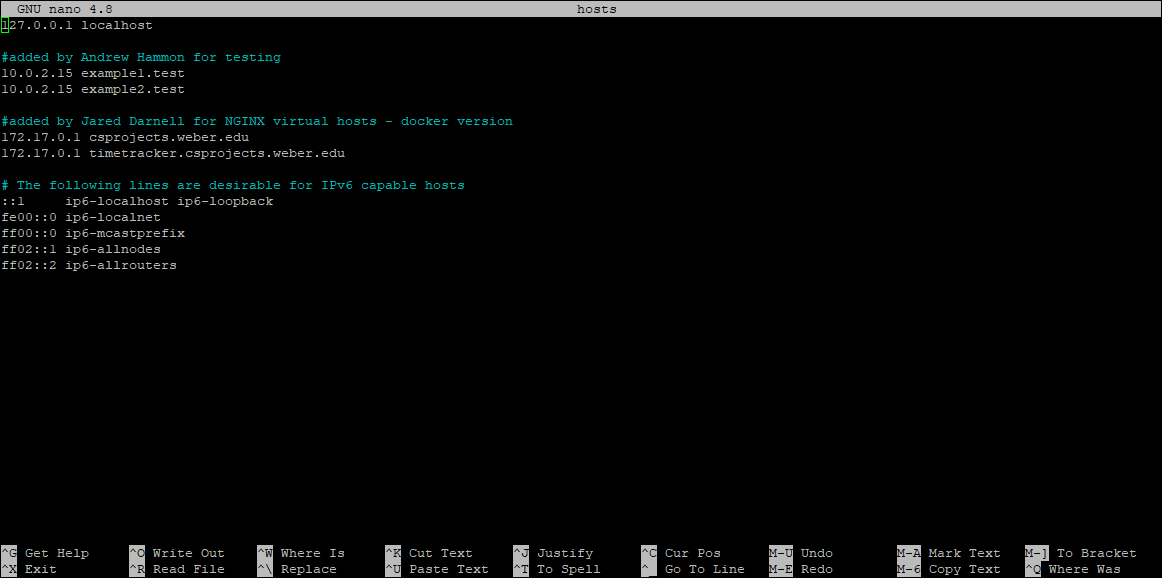
The server is hosted on the Weber Icarus network at csprojects.weber.edu, you can ssh to this by doing ssh [firstnamelastname@csprojects.weber.edu](mailto:firstnamelastname@csprojects.weber.edu) when connected to Icarus. You can connect to Icarus via VPN or ssh. You will need to talk to Bradley Peterson to get access to the server.

Currently the project uses a Docker container to contain a PostgreSQL database and a ASP.NET Core web application. The server utilizes a docker-based NGINX reverse proxy to route incoming requests like csprojects.weber.edu/timetracker to the timetracker container. At the moment it is not routing traffic properly to be visible from a remote IP address so using csprojects.weber.edu/timetracker will bring you to a 503 page. This could be caused by the non-docker NGINX being installed on the machine and being disabled.

The commented out stuff in the Dockerfile about a reverse proxy is not needed as that is running in a separate container and is linked to projects on the net docker network. To view the networks use sudo docker network ls. To view running containers use sudo docker ps. To view all containers use sudo docker ps -a. For images it’s sudo docker images. Also, nano is a very user-friendly terminal-based text editor.

I was following this guide <https://linuxhandbook.com/nginx-reverse-proxy-docker/> to set up the NGINX reverse proxy. For this you want to scroll down and look at step 5, as that is where I left off. I’ve included the VIRTUAL\_HOST and VIRTUAL\_PORT parameters in the docker-compose.yml file.

As shown in that guide you need to route your hostname to the docker IP address.



I’ve already done this under the virtual hosts section of the hosts file (located in /etc/), but you may need to make changes.

At /etc/nginx/ the notable files are nginx.conf, /conf.d/default.conf, /sites-available/default, and /sites-available/projects\_server.conf. You may need to configure these as I’m not certain that the docker-based nginx proxy utilizes these or not. It’s possible that the non docker-based nginx needs to be uninstalled.

The reverse proxy image that is being used is jwilder/nginx-proxy located <https://hub.docker.com/r/jwilder/nginx-proxy> there’s a decent amount of documentation that may help you out in setup.

I’ve included both the docker-compose.yml file for the reverse proxy and a zip of the files I’ve been using on the server (the reverse-proxy folder and the production-timetracker.zip on the github.)

The timetracker configs also need to be changed over to be more applicable for a production environment. Change the POSTGRES\_PASSWORD in .env to something much more suitable than “example”. Also change the ASPNETCORE\_ENVIRONMENT to Production. The docker-compose file should also be changed to pull from these settings. There are probably some more things that can be done as the image is currently over 1gb.

A migration needs to be created to transition the database from a dev-like environment to a production environment. This means removing default users and creating an admin account that Bradley Peterson knows the credentials for.

If you run into issues being unable to access the database or migrations not applying (tables not creating) then try messing with the Host parameter connection string in appsettings.json or the entrypoint.sh file may not be running the dotnet ef database update properly.

The Database property in appsettings.json is the actual name of the database, you might want to change this to something like timetracker-db or something at some point. The value host.docker.internal is the IP address of where the docker container is, it is very different from localhost.

Getting this hosted is a lot more work than it appears to be and don’t get discouraged if you are not making a whole lot of progress.

## Screenshots

These are screenshots taken of the front end with dummy data, and represent the anticipated final product functionality, not the current functionality. Some of the ideas for what the functionality looks like isn’t exactly the same as what the website is currently running.

#### Login - Done

#### Register - Done

#### Dashboard - Done

#### Courses - Student - Done

#### Course - Student - Done

#### Project - Student - Not Done (Pie Chart not populating)

#### Group - Member - Done

##### Group - Member 2 - Done

#### Courses - Instructor - Done

#### Course - Instructor - Done

#### Project - Instructor - Not Done

#### Group - Instructor - Done

##### Group - Instructor 2 - Not Done (Editing Time)

#### Users - Instructor - Needs to be redone

#### User - Instructor - Done