

Student Time Tracker

Summary and Hand-Off Documentation

This application uses a MongoDB 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. While MongoDB, a non-relational database, has worked for the application thus far, there are difficulties in storing and interacting with data in it. Due to the fundamental structure of noSQL, we had to design the collections and models to imitate that of a relational database, which isn't really in line with how MongoDB is supposed to work though it is good experience to see how this can be accomplished. It might be worth considering moving to a relational database in the end however.

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. 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.

The Front End

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.

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 follow by some dummy data to allow a user to see what a page should look like. Because we weren't able to test all the endpoints with real data from the database, there are likely to be a few little things that need to be ironed out once the endpoints are defined and uncommented from the Angular controllers.

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

Access to the Mongo Database is accomplished in two part. The MongoGateway class, which at the time of instantiation establishes a connection to the database (review collections and documents with MongoDB) and the static DataAccess class which instantiates the MongoGateway class has defined methods used to pull data from the collections and return the desired queries. Collections can be retrieved using a generic Bson Document type, or mapped to a user defined model which was done for this project. This allows one to query a collection that gets returned directly into an object or list. Using this structure we are then able to easily store and retrieve objects in the database.

DATABASE STRUCTURE

- **DetailedCourses**
 - courseID
 - courseName
 - professorID(Professor)
 - isActive
 - Projects**
 - ProjectID
 - Groups**
 - groupID
 - Users**
 - userID
 - TimeCards**
 - timeslotID
- **Courses**
 - courseID

- courseName
- professorID(Professor)
- isActive

- **Projects**

- ProjectID
- ProjectName
- isActive

- **Groups**

- groupID
- groupName
- isActive

- **Users**

- userID
- username
- password
- firstName
- lastName
- isInstructor

- **TimeCards**

- timeslotID
- dateIN
- dateOut
- Hours
- IsEdited
- CreatedOn

MONGOGATEWAY

- A connection string is defined to connect to a cloud database. This can easily be substituted with the parameters for a local database.
- Client settings are defined that are required to be in place by the hosted server. These need not be defined if the database is to be accessed locally as mongo will use the default settings automatically configured.
- A database accessor uses the connection string and defined settings to establish a connection to the database.
- Collection mappings are defined in MongoGateway class as they are directly correlated with the database structure.

DATAACCESS

- Static class housing methods to return objects and lists of objects. Structured to return data in a similar fashion to that of a relational database.

#UserRequest Region

- GetUserList
- GetUser
- AddUser
-

#CourseRequestRegion

- AddCourse
- GetCourses
- GetDetailedCourses
- GetDetailedCourseList
- UpdateCourse

#ProjectRequestRegion

- AddProject
- UpdateProject
- GetCourseProjects
- GetProject
- GetUserProjectsList

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

This leaves quite a bit unfinished:

- 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
- Getting Group information and all the Users and their Time information
- Allowing Users to add and edit time information, and changing the time to mark that it has been edited if it's been edited over a week after it was made.
- 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
- Probably some things that I've forgotten

Models

User

- _id
- username
- password
- firstName
- lastName
- isInstructor
- isActive

Group

- _id
- name
- isActive
- users[]
 - _id
 - time[]
 - _id

Project

- _id
- name
- isActive
- groups[]
 - _id

Time (might actually be the TimeCard model)

- _id
- hours
- timeIn
- timeOut
- isEdited
- createdOn

Course

- _id
- name
- instructorID
- isActive
- projects[]
 - _id
- users[]
 - _id
 - isAccepted

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 by 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

Notes

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 against 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

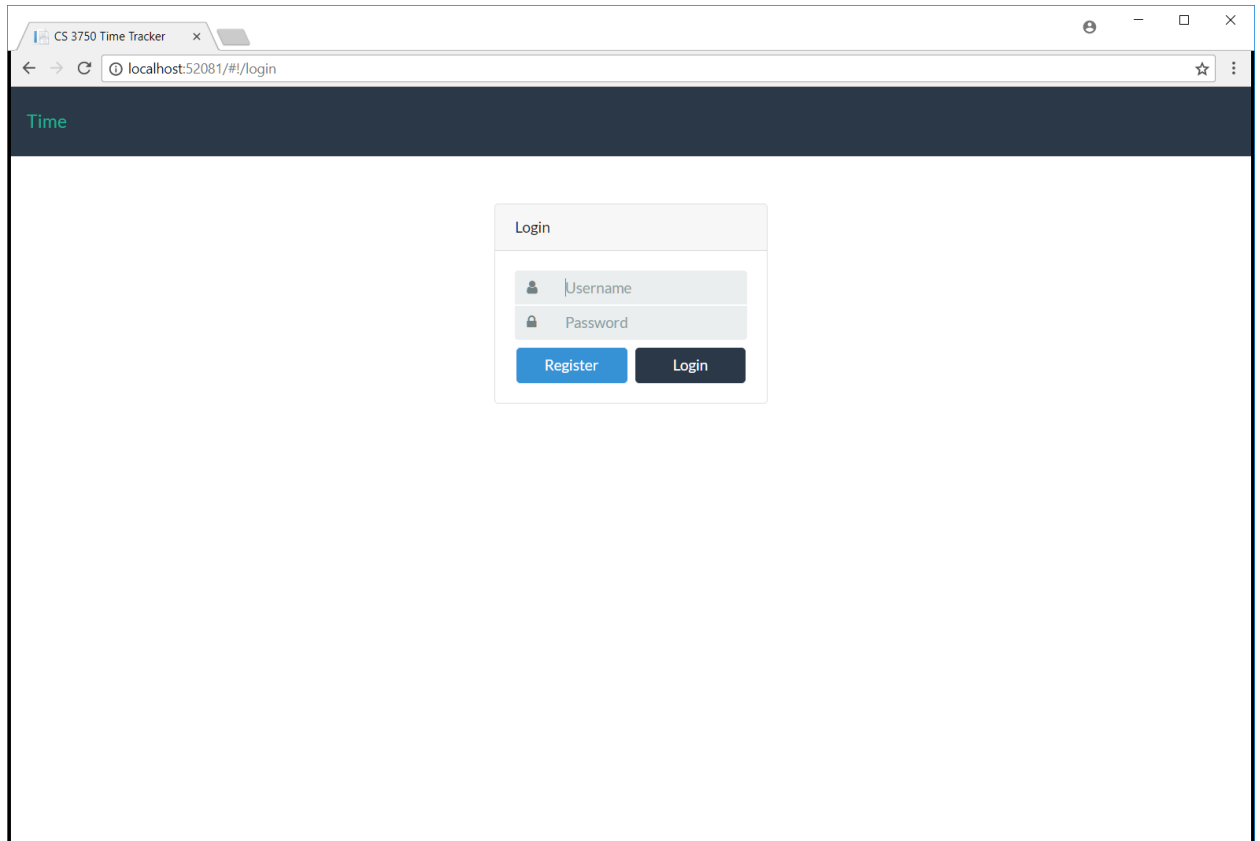
- /Home/CheckSession
 - If there is someone logged in, return the User with
 - _id
 - firstName
 - lastName
 - isInstructor
- /Home/CheckInstructor
 - If there is someone logged in (Assumes there is), return a bool indicating if they are an instructor
 - True
 - False
- /Home/RegisterUser
 - Will receive 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

Screenshots

These are screenshots taken of the front end with dummy data, and represent the anticipated final product functionality, not the current functionality.

Login



The screenshot shows a web browser window with a single tab titled "CS 3750 Time Tracker". The address bar displays "localhost:52081/#/login". The page has a dark blue header with the word "Time" in green. The main content area is white and contains a centered login form. The form has a title "Login" and two input fields: "Username" with a person icon and "Password" with a lock icon. Below the fields are two buttons: a blue "Register" button and a dark blue "Login" button.

CS 3750 Time Tracker

localhost:52081/#/login

Time

Login

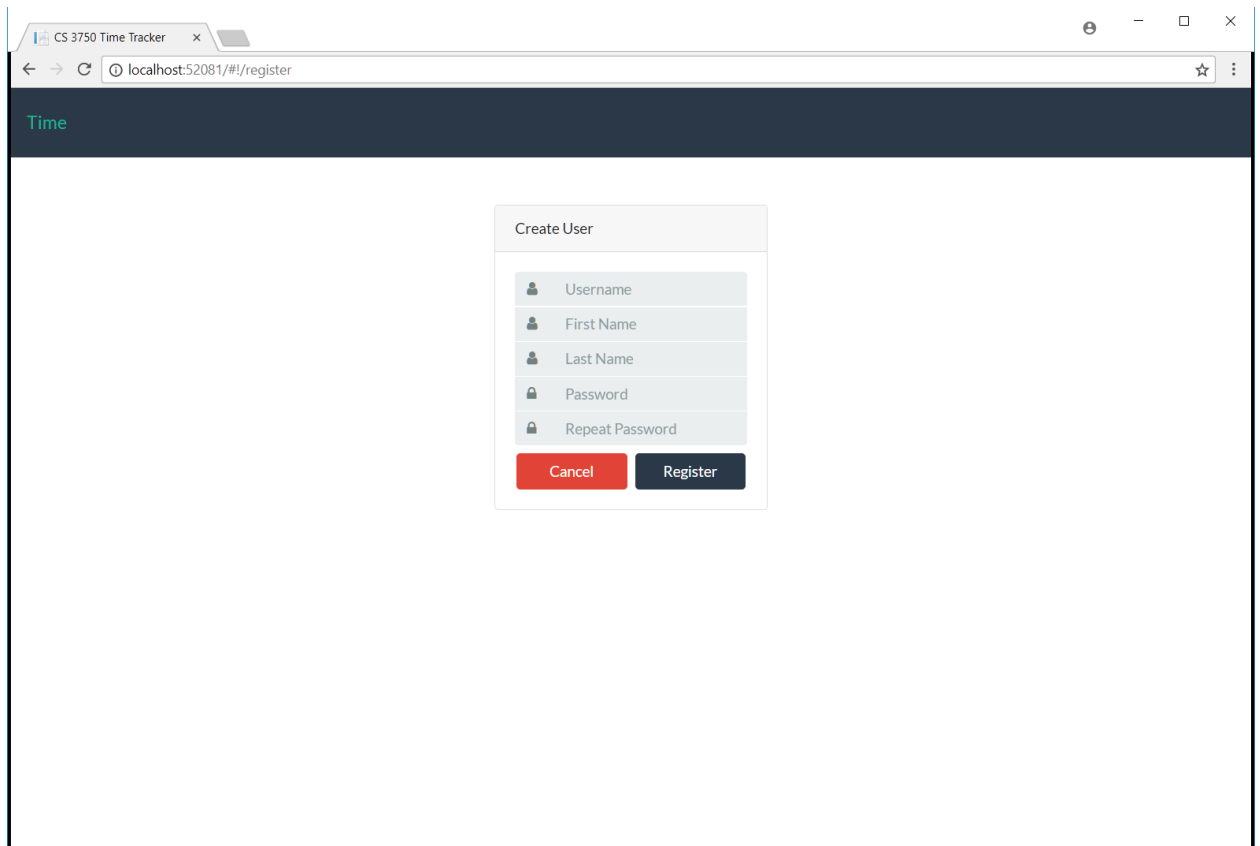
Username

Password

Register

Login

Register



The screenshot shows a web browser window with a single tab titled "CS 3750 Time Tracker". The address bar displays "localhost:52081/#!/register". A dark blue header bar at the top of the page contains the word "Time" in green. The main content area is white and features a centered "Create User" form. The form has a light gray header with the title "Create User". Below the header, there are five input fields, each with a user icon and a label: "Username", "First Name", "Last Name", "Password", and "Repeat Password". At the bottom of the form, there are two buttons: a red "Cancel" button and a dark blue "Register" button.

CS 3750 Time Tracker x

localhost:52081/#!/register

Time

Create User

Username

First Name

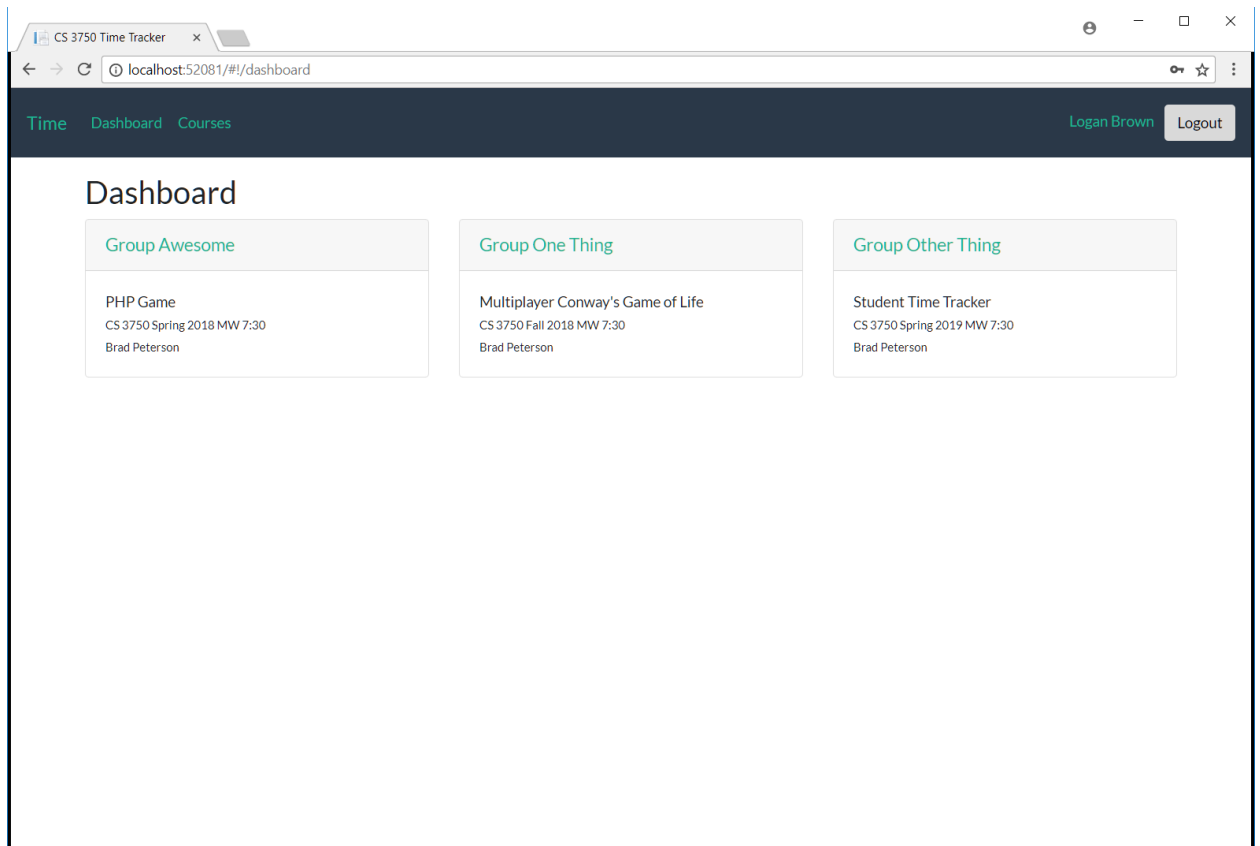
Last Name

Password

Repeat Password

Cancel Register

Dashboard



Courses – Student

CS 3750 Time Tracker x

localhost:52081/#/courses

Time Dashboard Courses Logan Brown Logout

Courses

Course Name	Instructor
CS 3750 Spring 2018 MW 7:30	Brad Peterson
CS 3750 Fall 2018 MW 7:30	Brad Peterson

Course – Student

The screenshot shows a web browser window with a single tab titled "CS 3750 Time Tracker". The address bar displays "localhost:52081/#!/course/12". The application has a dark blue header bar with navigation links "Time", "Dashboard", and "Courses" on the left, and a user profile "Logan Brown" with a "Logout" button on the right. The main content area is titled "CS 3750 Spring 2018 MW 7:30". It is divided into two columns. The left column, labeled "Projects", contains a list with two items: "PHP Game" and "Multiplayer Conway's Game of Life". The right column, labeled "Students", contains a list with three items: "Logan Brown", "Rizwan Mohammed", and "Skylar Olsen". A "Join" button is located at the top right of the "Students" list.

CS 3750 Time Tracker x

localhost:52081/#!/course/12

Time Dashboard Courses

Logan Brown Logout

CS 3750 Spring 2018 MW 7:30

Projects

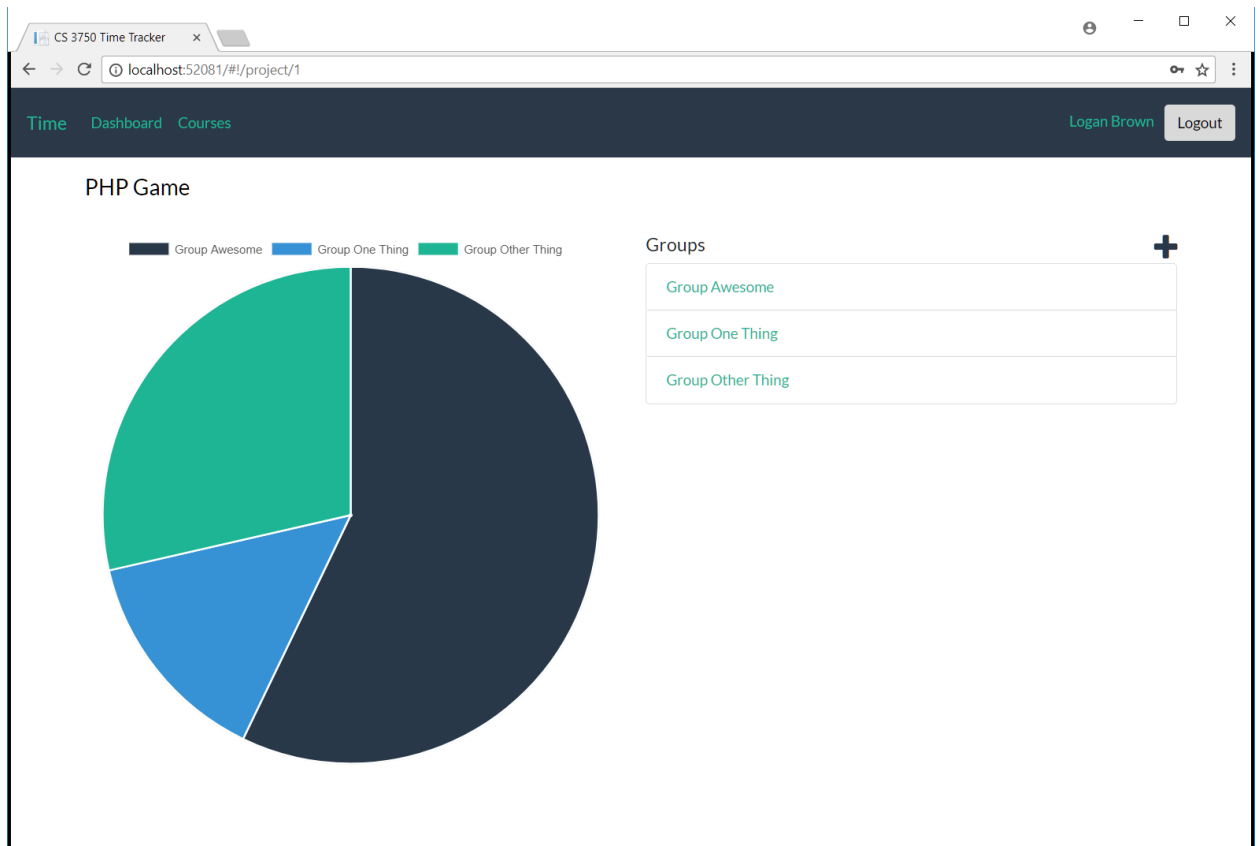
- PHP Game
- Multiplayer Conway's Game of Life

Students

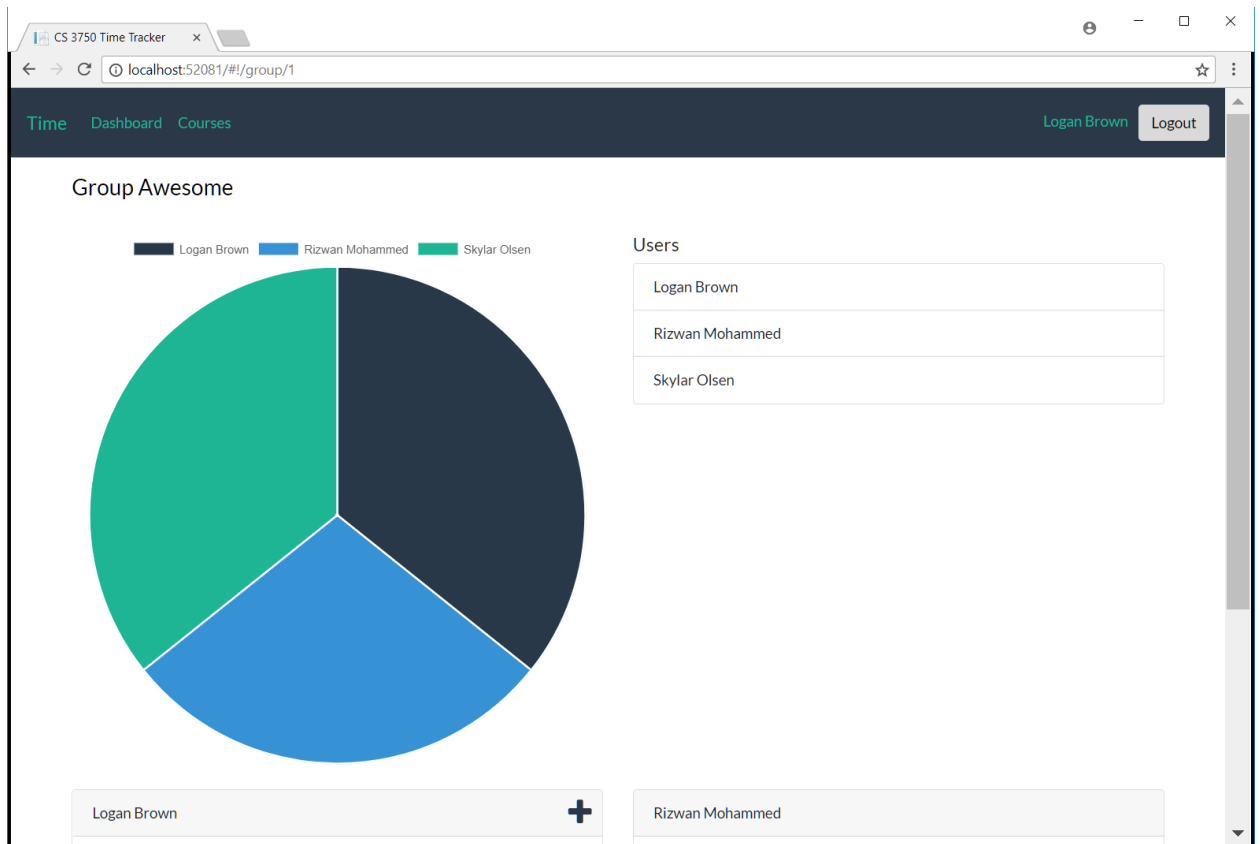
- Logan Brown
- Rizwan Mohammed
- Skylar Olsen

Join

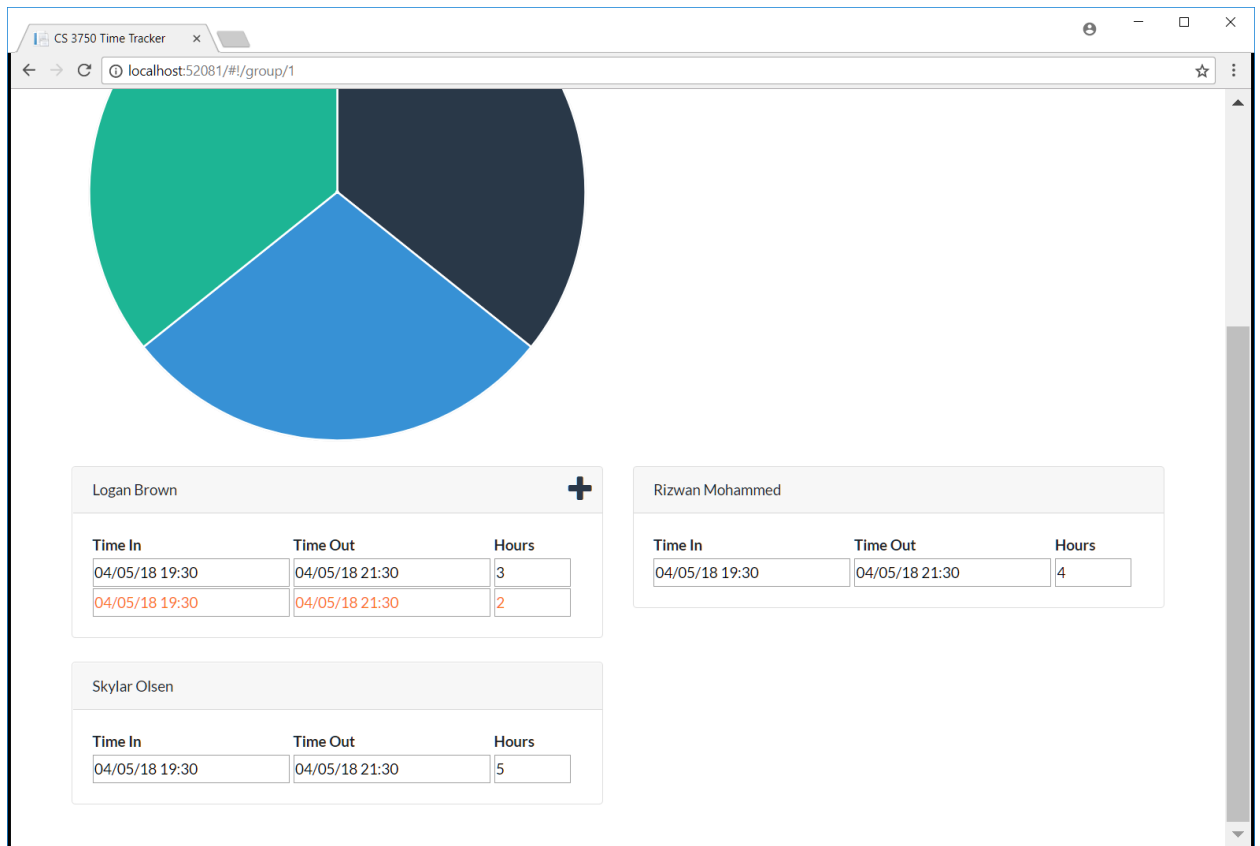
Project – Student



Group – Member



Group - Member 2



Courses – Instructor

The screenshot shows a web browser window with the title "CS 3750 Time Tracker". The address bar displays "localhost:52081/#!/courses". The application has a dark blue header with navigation links: "Time", "Dashboard", "Courses", and "Users". On the right side of the header, it shows the user "Brad Peterson" and a "Logout" button. The main content area is titled "Courses" and includes a toggle switch labeled "Show Inactive" and a plus icon. Below this is a table with two columns: "Course Name" and "Instructor".

Course Name	Instructor
CS 3750 Spring 2018 MW 7:30	Brad Peterson
CS 3750 Fall 2018 MW 7:30	Brad Peterson

Course – Instructor

CS 3750 Time Tracker

localhost:52081/#!/course/12

Time

Dashboard

Courses

Users

Brad Peterson

Logout

CS 3750 Spring 2018 MW 7:30

Active

Projects

Show Inactive

+

Students

Join

Accepted

PHP Game

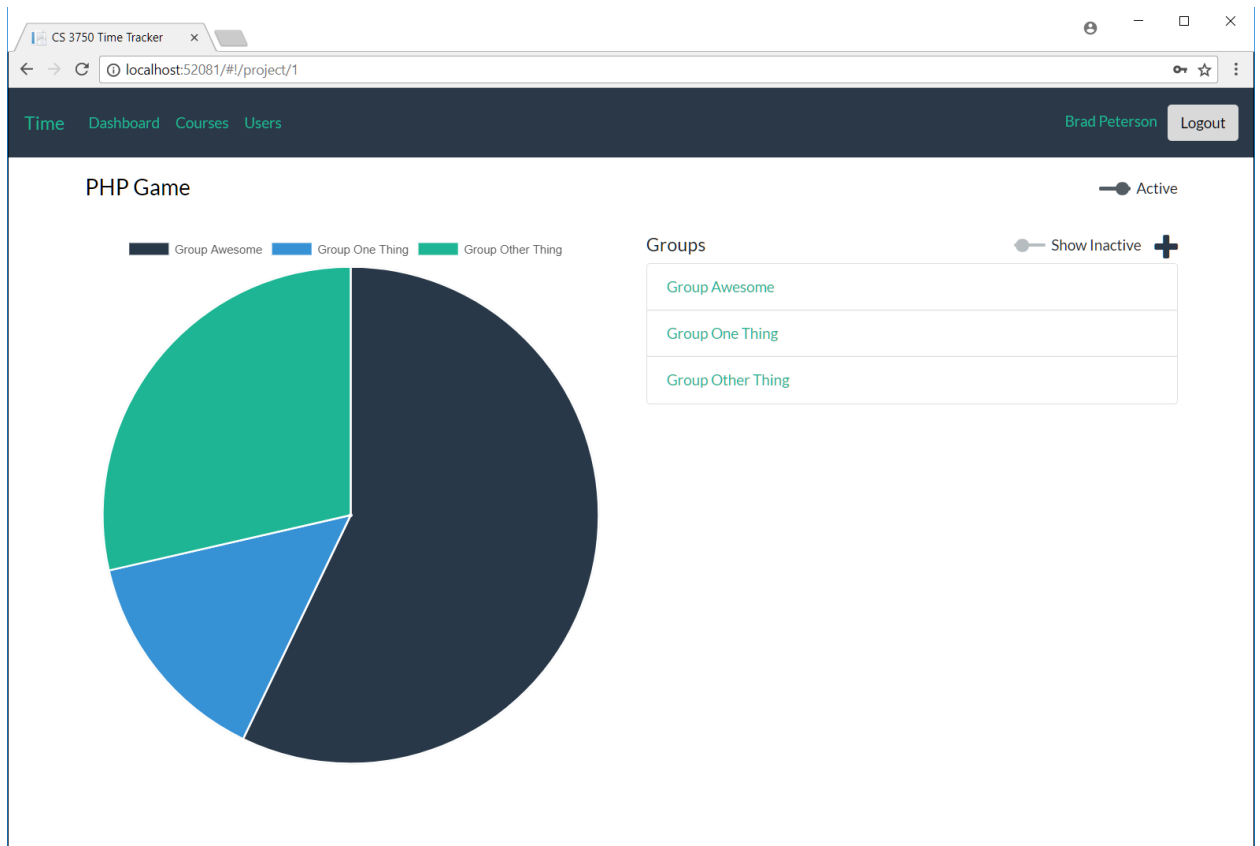
Multiplayer Conway's Game of Life

Logan Brown

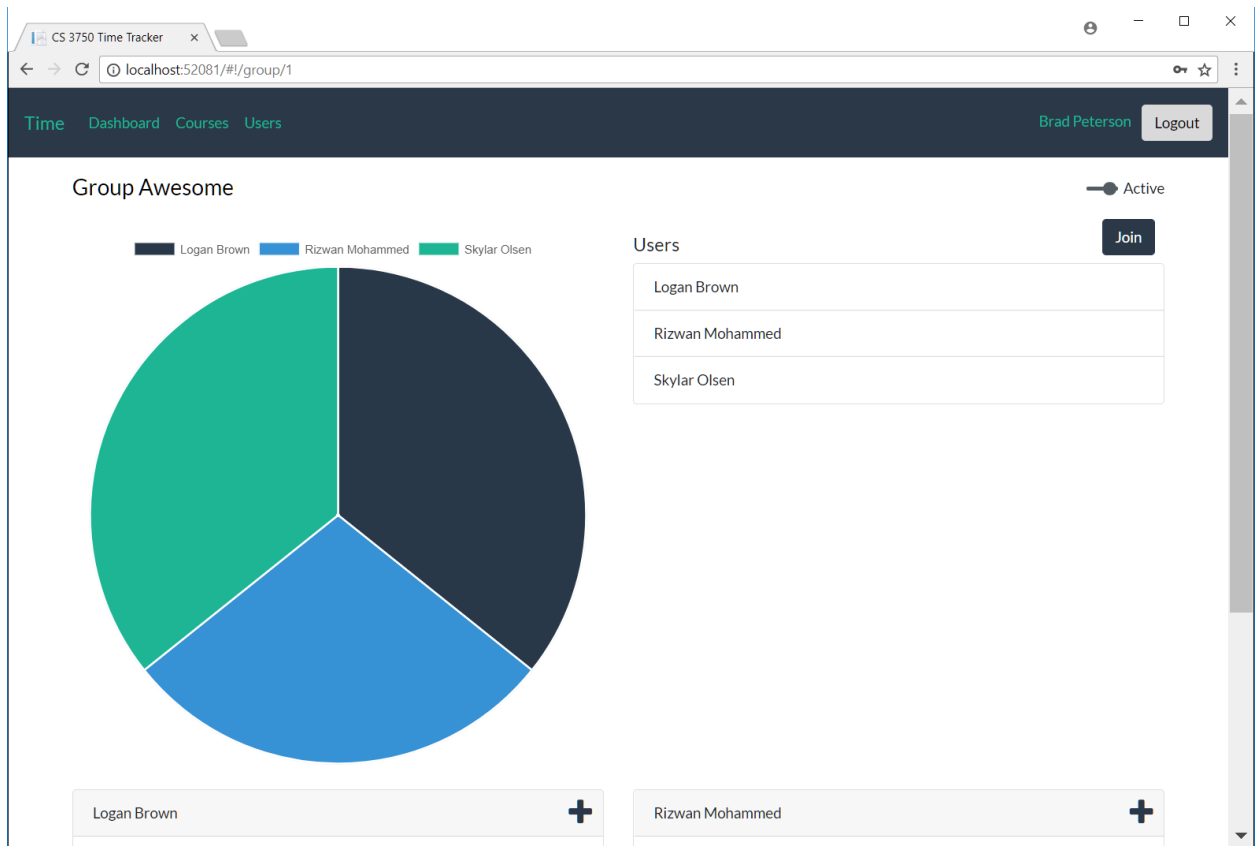
Rizwan Mohammed

Skylar Olsen

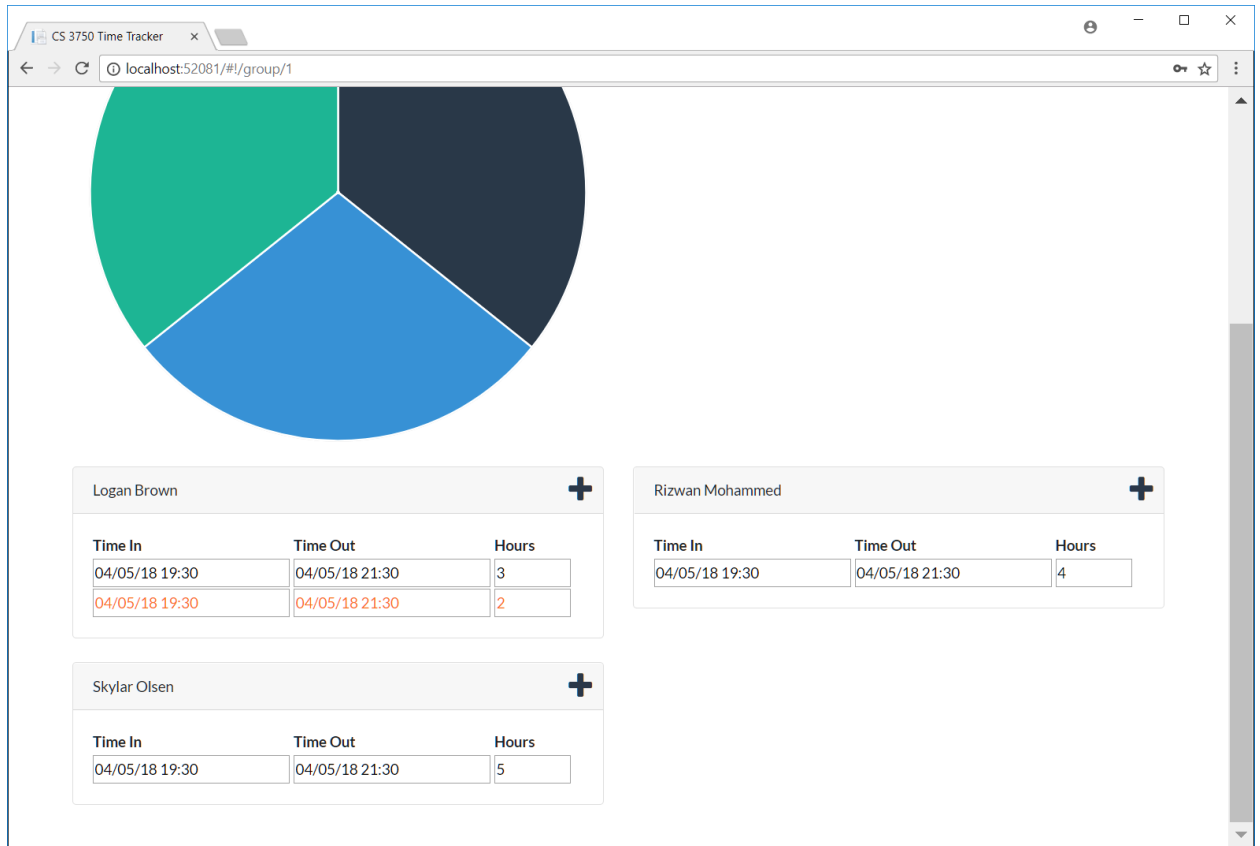
Project – Instructor



Group – Instructor



Group - Instructor 2



Users – Instructor

CS 3750 Time Tracker

localhost:52081/#!/users

Time Dashboard Courses Users

Brad Peterson Logout

Users

Name	Username	Active	Instructor
Logan Brown	logan	<input checked="" type="checkbox"/>	<input type="checkbox"/>
Rizwan Mohammed	riz	<input checked="" type="checkbox"/>	<input type="checkbox"/>
Skylar Olsen	skylar	<input type="checkbox"/>	<input type="checkbox"/>
Brad Peterson	brad	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>

User – Instructor

CS 3750 Time Tracker x

localhost:52081/#/user/5ad54b26fb49e95bf06a1c92

Time Dashboard Courses Users Brad Peterson Logout

Logan Brown

Username	<input type="text" value="logan"/>	Current Password	<input type="password"/>
First Name	<input type="text" value="Logan"/>	New Password	<input type="password"/>
Last Name	<input type="text" value="Brown"/>	Repeat Password	<input type="password"/>
<input checked="" type="checkbox"/> Active <input type="checkbox"/> Instructor		<button>Change Password</button>	