FUNCTIONAL REQUIREMENTS FOR TIMEPUNCH

1 GENERAL

1.1 Project Description

We are to create a time tracking web app written on ASP.NET to record everyone's work on a per-project basis. The system differentiates between professors and students. The professor needs to be able to create and edit courses and projects. Students are allowed to, for each project, create groups or join existing groups, and potentially have a method of removing a group member. Everyone registered to a course is allowed to see the progress of others in the same course. The professor needs to have a system of approving students into courses and overriding group choices when problems arise.

1.1.1 Background

Prior solutions to this project were simple spreadsheet applications. They do not provide the organizational structure to support multiple users easily. A web app is desired that provides exactly the features for recording and visualizing time worked.

1.1.2 Interfaces to External Systems

- The application is written in .NET Core 2, specifically ASP.NET Core 2 MVC with Razor Pages framework.
- For ease of development and relocation, the database used is SQLite.

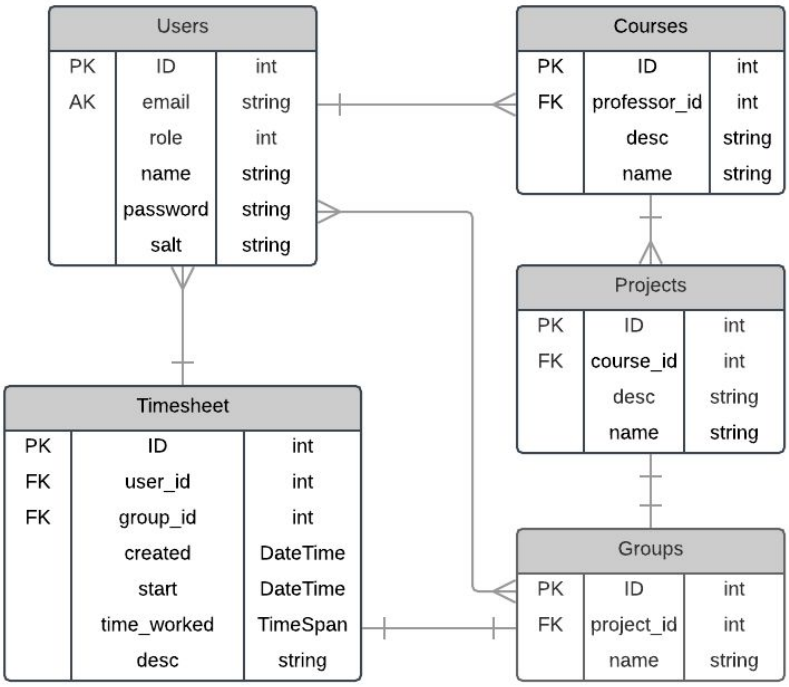
1.2 Original authors

- Daniel Salmond
- Rolando Collantes
- Kion Shamsa
- Matthew Bybee

1.3 Document References

Time punch outline, ERD, database, and the mockups for the web pages.

2 FUNCTIONAL REQUIREMENTS



2.1 Data Requirements

Entity definitions:

- Courses Table
  - To keep track of the courses and who the professor is for that course.
- Projects Table
  - Projects for courses. Projects know which course owns them. It has a name and a description.
- Groups Table
  - Groups for projects. Groups know which projects owns them. Besides a primary key, it just has a name.
- Users Table
  - Defines each individual user. Each user has a Role, as in Professor or Student. It's used for login.
- Timesheet Table
  - To keep track of the time students put into projects

Attribute definitions. Parentheses is (column name, .NET type). Please see SQLite notes.

- Courses Table (parentheses is column name, type):
  - Course ID (id, PK int)
  - Course Name (name, string)
  - Course Description (desc, string)
  - Professor (professor_id FK Users(id), int)
- Projects Table
  - Project ID (id, PK int)
  - Project Name (name, string)
  - Project Description (desc, string)
  - Course ID (course_id FK Courses(id), int)
- Groups Table
  - Group ID (id, PK int)
  - Group Name (name, string)
  - Project ID (project_id FK Projects(id), int)
- Users Table
  - User ID (id, PK int)
  - Name (name, string)
  - Role (role, enum (int)) : Unapproved, Student, Professor, SiteAdmin
  - Email (email, UNIQUE string)
  - Password Hash (password, string of base64 encoded hash)
  - Salt (salt, string of base64 encoded binary data)
    - Used to ensure each password hash is unique for each user, even if they happen to use the same real password. When a user logs in, the salt is combined with the password before it's hashed.
- Timesheet Table
  - Timesheet ID (id, PK int)
  - Date Created (created, DateTime)
  - User ID (user_id FK Users(id), int)
  - Group ID (group_id FK Groups(id), int)
  - Description (desc, string)
  - Start Time (start, DateTime)
  - Time Worked (time_worked, TimeSpan)

2.2 Functional Process Requirements

In the words of Professor Brad Peterson:

"FERPA requirement. I want this site to be a public site, and as such to comply with FERPA I can't have students auto register themselves, then select themselves into classes and groups. Instead, after auto registration, the student should select courses the instructor covers, and then the instructor needs to go in and approve those.

"Assume multiple instructors could use a single site. Also assume a single instructor could use this site for multiple courses.

"A good breakdown of the hierarchy is that a student can belong to a group, a group belongs to a

project, a project belongs to a course, a course belongs to an instructor, and an instructor belongs to the site. A student can only belong to one group within a project, and a student cannot belong to two or more groups within a project. A student can belong to multiple groups, just as long as each project is unique."

Although, we were told in person that a user may move between groups on their own. Please clarify this with the Professor. Our project and this documentation assumes students of a course can move around between groups on their own.

2.3 SQLite Notes

SQLite3 was chosen merely because it's portable.
It allows all the members of the development group to modify the entire project on their own computer.
It doesn't require the setup of a server.
It's also simple, sometimes to its detriment.
Please read the IMPORTANT note below about foreign keys.

SQLite3 only has 5 core/true types. They are:

- TEXT
- INTEGER
- REAL (Floating Point)
- BLOB (Binary data)
- NULL

There are other higher level "types", but SQLite seems to "decay" them into the above types.
https://www.sqlite.org/datatype3.html

It doesn't support stored procedures.

Dates and Time can be represented in different ways:

- TEXT ("YYYY-MM-DD HH:MM:SS.SSS")
- REAL (Apparently the fractional number of days since November 24, 4714 BC)
- INTEGER (Seconds since the Unix Epoch: Jan 1, 1970 00:00:00Z)

.NET DateTime and TimeSpan need to be converted to one of these types. We chose TEXT when creating the database.

The createdOn columns in all the tables use the INTEGER method, the number of seconds since the Unix Epoch.

IMPORTANT:
SQLite3's support for Foreign Keys depends on how it was compiled. Even when compiled in, it must be enabled at runtime, every time. I assume that the Nuget package implementing the SQLite interface uses a version of SQLite with Foreign Keys.

Execute the statement: PRAGMA foreign_keys=ON;
Before inserting or deleting to enforce those constraints.

Please look at the repository /databases/createdb.py Python script used to generate the tables for reference.

2.4 Asynchronicity/AJAX

In an effort to be as user-friendly and fault tolerant as possible, a user's input into the webpage should automatically save asynchronously through the use of AJAX, assuming it validates. We have not implemented any of this, but consider either JQuery, or writing your own using the Fetch API available to modern browsers.

SignalR, if available for ASP.NET Core 2.1 by the time you read this, can also support WebSockets with polling fallback to have bidirectional communication with the server and client.

3 OPERATIONAL REQUIREMENTS

3.1 Security

There are three roles of users: Unapproved, Student, Professor and a SiteAdmin.
.NET starts counting the first enum as 0, and each successive one is one more than the previous. Unapproved is 0, Student is 1, Professor is 2, and SiteAdmin is 3.

The unapproved user will not have access to anything until they are approved by the professor into a certain course. The student will have access to create groups for a project, join groups in a project, and input time logs spent on a project. The professor will be able to create projects for the course and approve unassigned users to the course as students. The site admin will be able to make modifications to the website and ad professors.

SiteAdmins can change the role of a user to Professor. This Role is intended for application management.

All passwords must be salted with random data and strongly hashed. All pages must be served over TLS. HTML links to external domains may be marked with no-referrer attributes. Refer to /Methods/PasswordUtils.cs

In an effort to avoid SQL Injections, use parameter binding and other kinds of sanitation.

3.2 Audit Trail

An audit system should be created that tracks all database changes, the time of the change, the location of the user that made the change (IP Address, for example), and anything specific required by FERPA.

3.3 Data Currency

The data needs to be current for the pie charts of groups and project to be accurate. In order for that to be accurate the time input for individuals needs to be accurate and up to date.

3.4 Reliability

The system needs to be reliable for the students grades to be accurate. If it goes down it needs to be back up pretty quickly so the students can input time spent on projects and the professor can see who is actually working on group projects within their groups. Consider changing away from SQLite to a scalable database when the project is nearing completion.

3.5 Recoverability

The system must be capable of multiple backups to ensure recoverability of the data. Test these backups to ensure they can be restored properly. Untested backups are not backups at all.

Some backups should also be stored off-site to ensure catastrophes in one location do not destroy the only source of data.

3.6 System Availability

System must be completed at the end of the summer semester ready to use for fall semester. It should be accessible by web browser over the open Internet.

4 General Handoff Notes

The project's current state can create users and allow them to log in. The Login and Create Account Razor Pages and their accompanying page models are functionally complete.

When not logged in, visiting any page besides /Login or /CreateAccount redirects the user to /Login.

When the user logs in at /Login, the page sets a cookie, and they're redirected to /Index (or just "/"). The cookie is set to expire after 2 hours. To test this system, you may need to manually delete your browser's cookies. A logout hasn't been implemented yet.

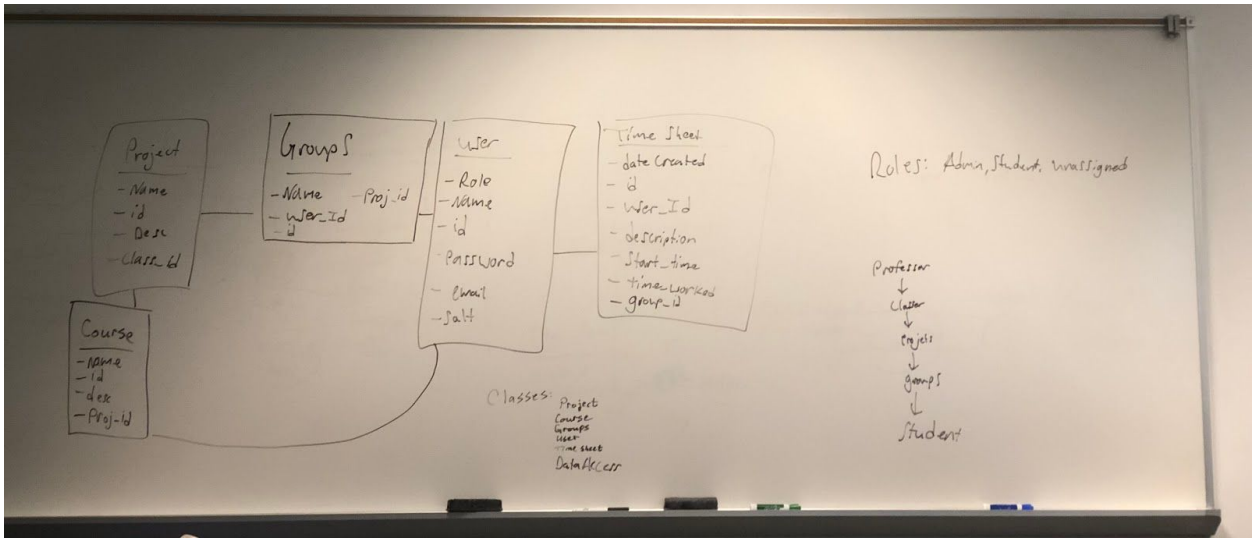
If you're inexperienced with the ASP.NET Core 2 Razor Pages framework, a good place to start is probably: https://jonhilton.net/razor-pages-or-mvc-a-quick-comparison/

Become accustomed to Dependency Injection (especially in configuring /Startup.cs)
Static files (such as .css and .js) are in /wwwroot/

The DataAccess class is a partial class. Its definition spans across the two files General.cs and Users.cs. The idea was to have one source code file for each of the tables. DataAccess is used to execute SQL queries in one logical location and to return objects to higher level C# code. Perhaps you can look into using the Entity Framework to abstract away having to deal with SQL. Changing to a different database with Entity Framework would probably be more seamless.

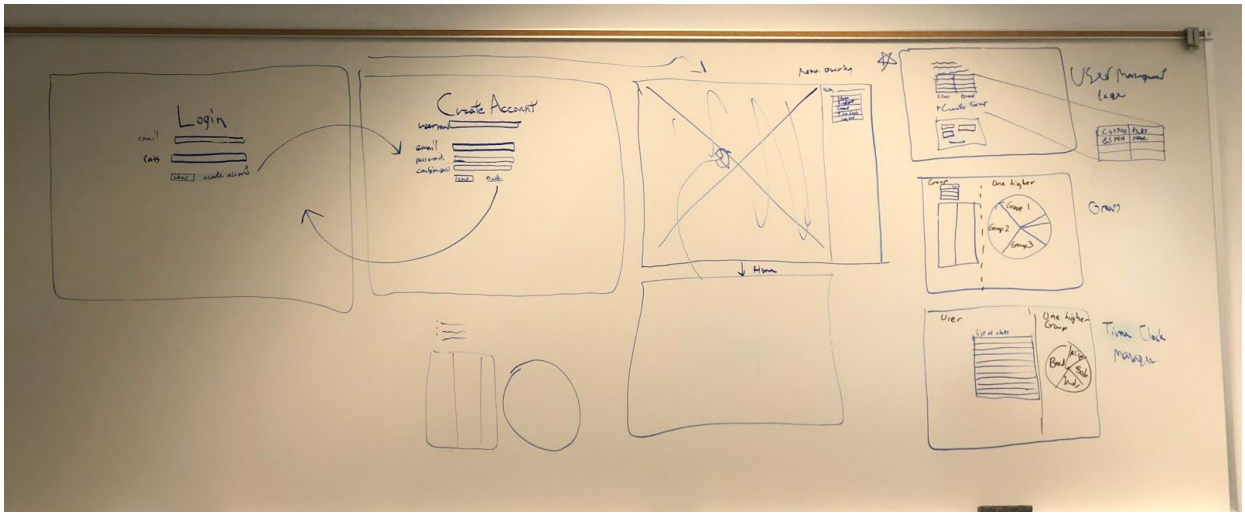
Link to the project repository on GitHub: https://github.com/collantesro/timepunch

These images were taken of the whiteboard after we discussed aspects of the project:

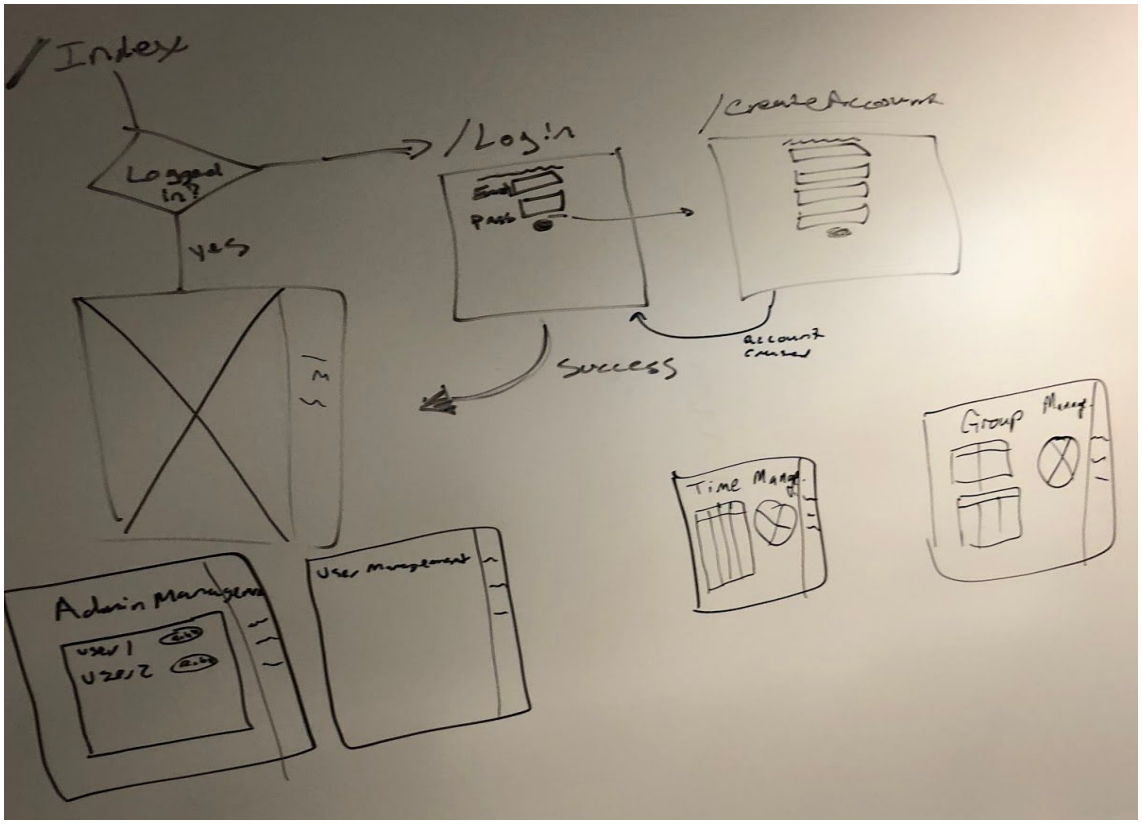


FUNCTIONAL REQUIREMENTS AND HANDOFF

This image was a brainstorming of the database. Hierarchy is listed in the bottom right: Professors own courses (renamed from “classes”), courses own projects, projects own groups, groups have multiple students.



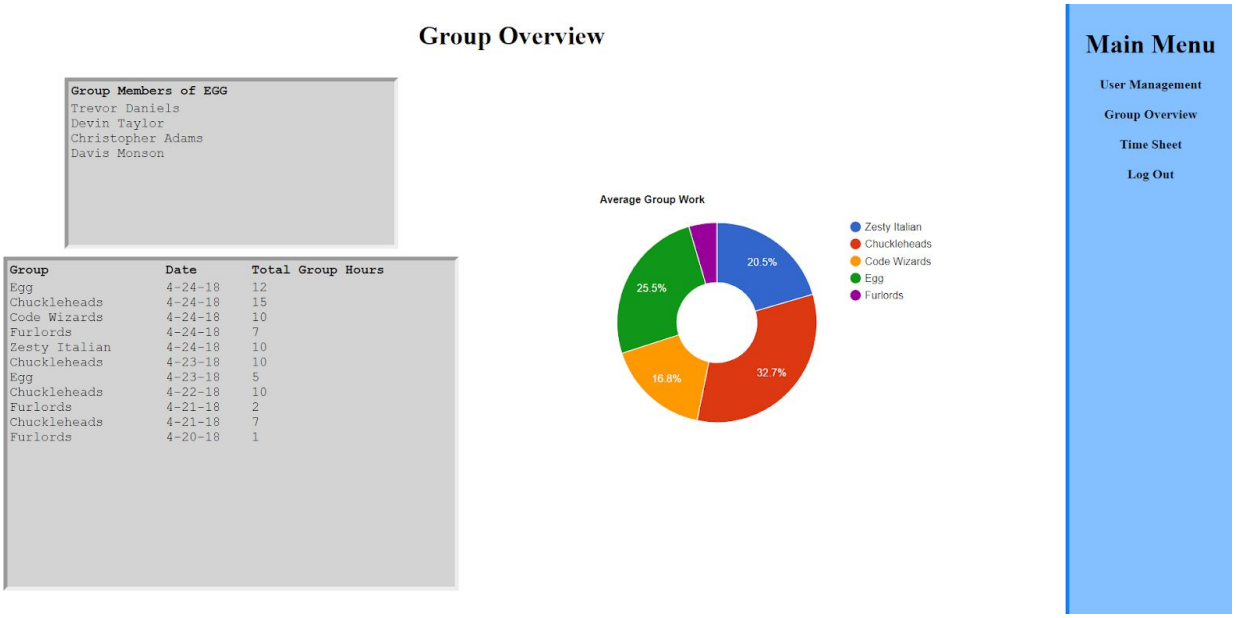
This image is a brainstorm of the major pages visible to the user. Pay attention to the reliance of pie charts. The pages on the right side of the board all have the navigation menu on the right-hand side.



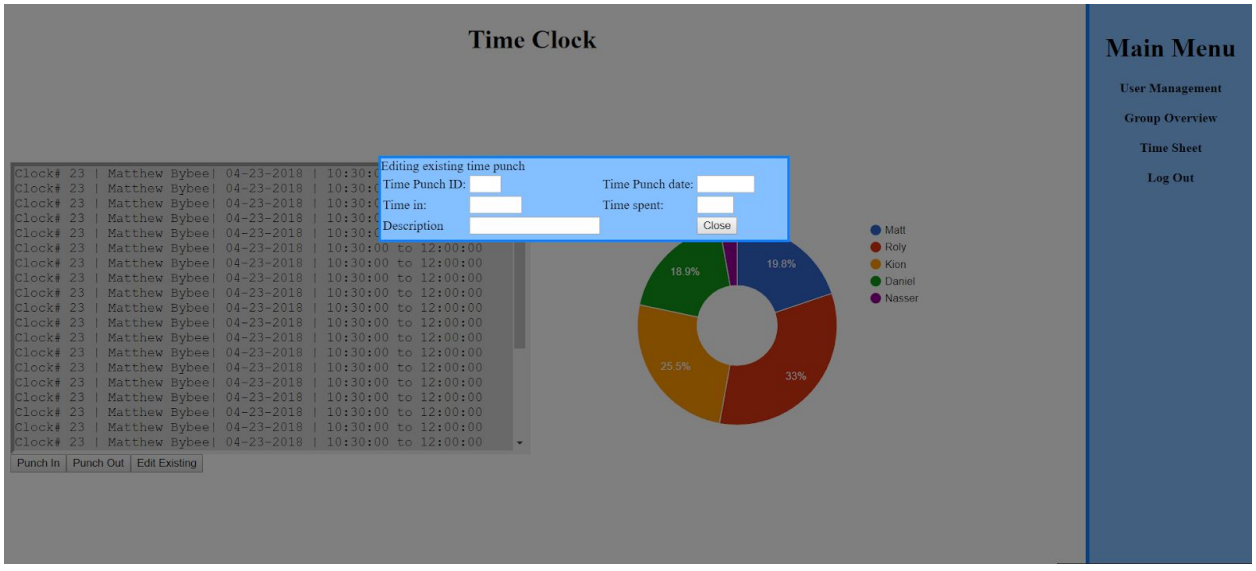
FUNCTIONAL REQUIREMENTS AND HANDOFF

This image tries to describe the flow-chart when navigating the site. Visiting /Index or any other page checks if the user is logged-in. If not, redirect to /Login. /Login has a link to /CreateAccount. If an account is created, the user is redirected to /Login. When a user logs in, they're sent to /Index, which displays either the user's home page or the professor's. This depends on their Role. From here, the user can visit any of the pages in the navigation menu.

Screenshots of mockups:

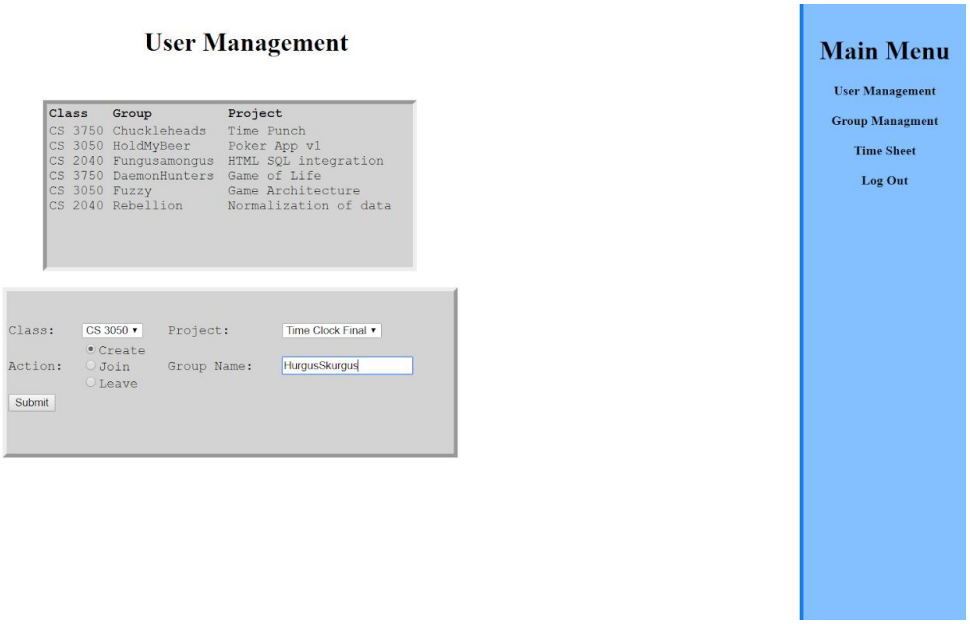


This is a screenshot of the Group Overview allows any of the users to see the progress of all the groups under the specific project. As long as a student is a member of the class, they can see the information of any of the other users in the same class. The chart on the right shows the comparison of time worked between different groups.



FUNCTIONAL REQUIREMENTS AND HANDOFF

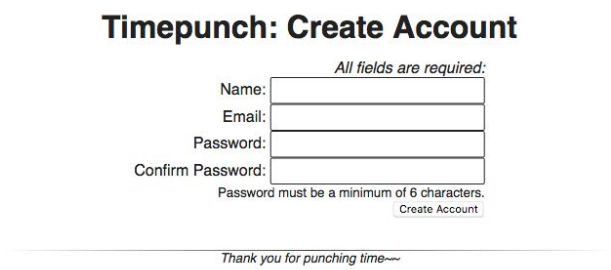
This is a screenshot of the Timeclock page. It allows the user to either automatically clock in and out (the system automatically generates the entry), or the user can manually enter their entry. Manual entry is for when a user forgets to clock in or out. Manual entries entered too long after the start or end are highlighted in red to hint at cheating, if any. The chart on this page shows individual contribution to the group.



This screenshot is of the User Management page. It allows the user to create a new group for a project, join an existing group, or leave their current group.

The source code used to render the above screenshots is included in the repository under /Resources/

Screenshots of /Pages/Login.cshtml and /Pages/CreateAccount.cshtml:


Timepunch: Create Account
All fields are required:
Name:
Email:
Password:
Confirm Password:
Password must be a minimum of 6 characters.
Create Account
Thank you for punching time~~