CSCI/CMPE 3340 Report – BattleFund.com

Project Summary

The purpose of this project was to create a web application where fighting game players could log in, create their player profile, and fund-raise to reach specific goals. These goals are going to be fighting game oriented, either for players to buy in in a certain tournament, or for them to fund their career. Profits will be made by subtracting a 10% fee to every donation made toward a certain goal.

Summary of Contributions

Luis Gutierrez and Jose Balanza Martinez where the developers and designers of the application.

Jose Balanza Martinez was responsible of the Ruby on Rails components, such as:

* Implemented devise authentication with customized parameters such as first name, last name, user\_name, and birthday. The user model was associated to the player model, and to the goal model through the player model.
* The player controller and model, which associated to the user model which allowed users to create and edit their player profile, and create goals.
* The goal controller and model, a model associated to the player and user models, which allows a user to create and edit their fund-raising campaign.
* Implemented protection, so users can’t modify another user’s goals and player.
* Implemented a profile controller that shows edits a user profile.
* Implemented a own\_goals controller that only shows the goals that belong to the player.

Luis Gutierrez was responsible for the front end development and was responsible for:

* Creator of the idea
* Integrated a third party Bootstrap template.
* Integrated a third party Bootstrap theme and integrated it to the the home\_index, the goals\_index, the player\_index, and the goals\_current, and the profile\_index pages.
* Integrated a live progress bar to show the progress of the goals.
* Edited the navigation bar to show links depending on whether the user is logged in or not.
* Created the tables in which all the goals and players load.

Requirements

BattleFund.com has the follow requirements:

* Any user that lands on the page can view goals and player profiles.
* A “goals” and “players” page will allow users to browse and chose their desire goal and player to support.
* A user can also click on a player profile and that players goals should load, they will also be able to click on “My Goals” and their goals should load.
* Users must be able to sign up and register an account.
* Users must be able to log in, log out of their accounts.
* Users need to create a player account with their player details and accomplishments in order to start creating goals.
* Users must create goals in order to start receiving funding.
* This web application should allow users to donate funds to their favorite goal.
* Players must pay 10% of the donation as a campaign hosting fee.

The system needs to support registered users and players and non registered users.

Users have the following privileges:

* Create player profile.
* Edit their goals and edit their player and user profile.
* Donate to their favorite goal.

Players have the following privileges:

* Create their fund-raising campaigns.
* Edit and manage their campaigns via goals.
* Raise funds towards their campaigns.
* Check the progress on their campaign.

Non registered users have the following privileges:

* Browse goals and players.
* Access other users’ player profiles.
* Create an account

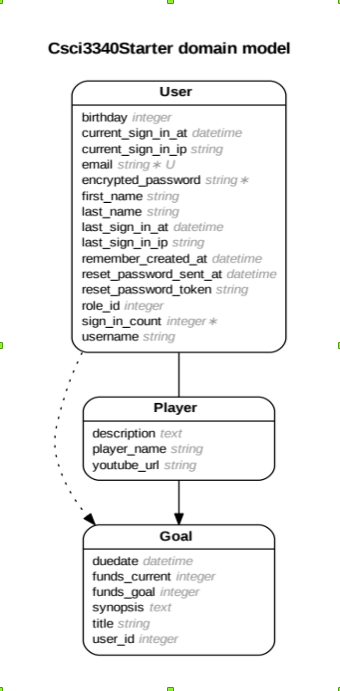
API’s and third party gems

This web application is heavily dependent on payments, which sadly are out of the scope of this project and will be added if time allows. We are planning on using Stripe for third party payment integration. This will give us the security that no credit card information will be saved.

Interaction Diagrams



Class/Model Diagrams

The user model is where a user creates their own personal user. We wanted to separate the user model from the player model just in case a user just wanted to sign in, but not raise any funds. This model stores all the information related to the user, their first name, last name, birthday, email, username, email and password. Users that have a user profile are allowed to donate to players’ goals.

The player model is related to the user model, which the user can make if he/she doesn’t have one, or edit it at any time. The player can include a Youtube video by copying and pasting a Youtube video link in their player profile. This has not yet been implemented as of writing this paper.

The goal model is related directly to the player model, and it is also directed to the user model through the player model. That way you can know what goals belong to what user and player. In the goal, a player stores relevant information such as the goal’s synopsis, the due-date, the desired funding goal, title and user and player ids.

System Architecture/Design

The system was developed using the Ruby language on the Ruby on Rails framework. The development database was implemented with Squlite3, and later changed to the more Heroku friendly PostgreSQL. The fronted was developed using th ERB templating, delivering HTML to the browser; CSS was the language used for the styling.

The Model-View-Controller is as follows.

The models are:

* The User Model ( explained above)
* The Player Model ( explained above)
* The Goal Model ( explained above)

The controllers are:

* Administrator controller (not implement yet)
* Home controller
* Profile controller
* Player controller
* Goals controller
* Goals\_Own controller

The Administrator controller and views allows the moderators to delete unfinished, and inappropriate goals.

The Home controller and views allow users to view the homepage and browse the highlighted goals.

The Profile controller and views allow users to edit their user information, and link to their player.

The Player controller and views allow users to create and edit their player and links to the “create a goal page” and to their goals page if they have any.

The Goals controller and views allows users to create, show, and edit their goals.

The Rails router takes web addresses and maps them to actions int the controller and displays the view associated with that action.

The application is deployed and hosted to Heroku, this site provides hosting and database services for Ruby on Rails and other applications.

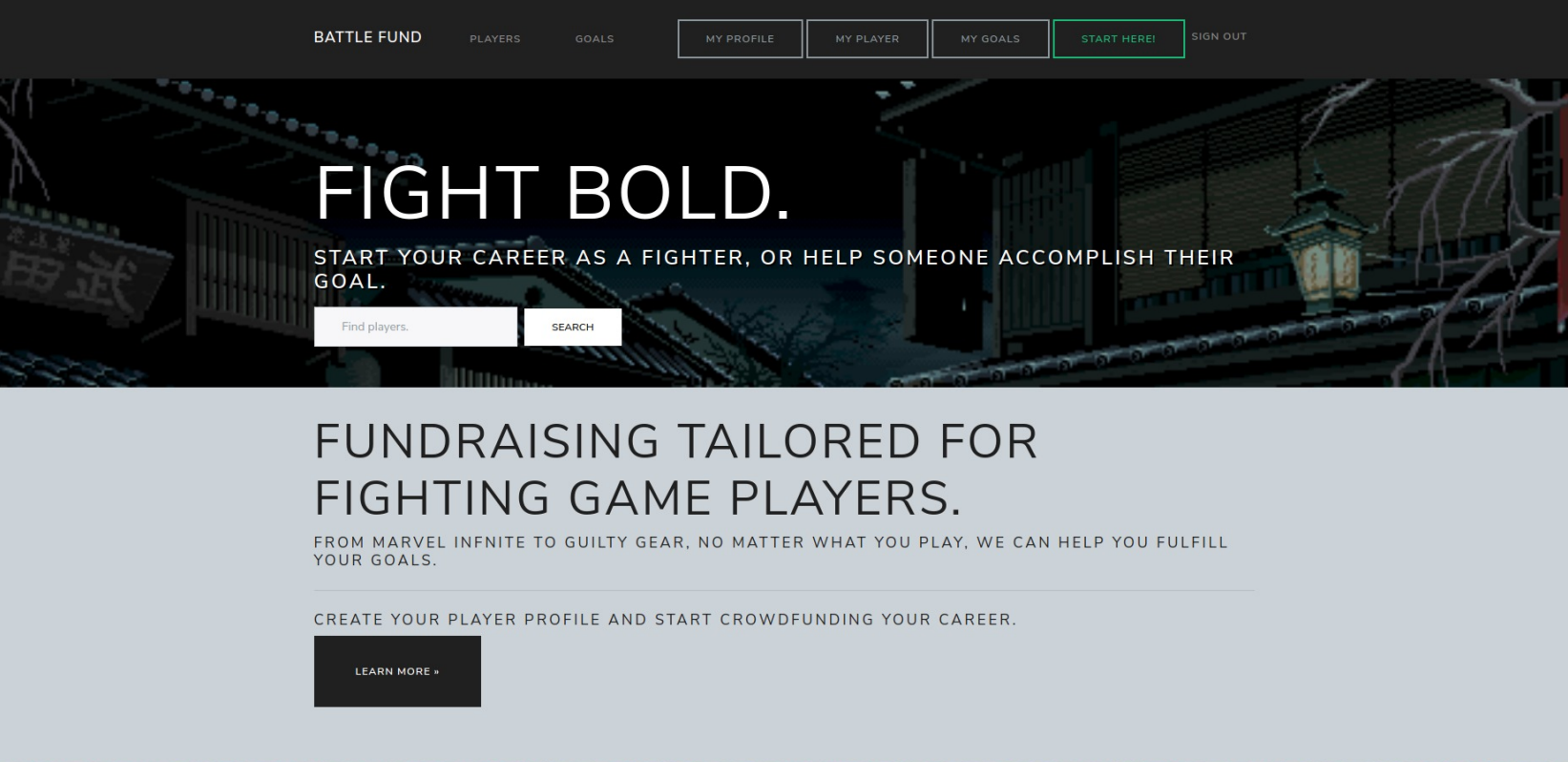
Algorithms & Data Structures

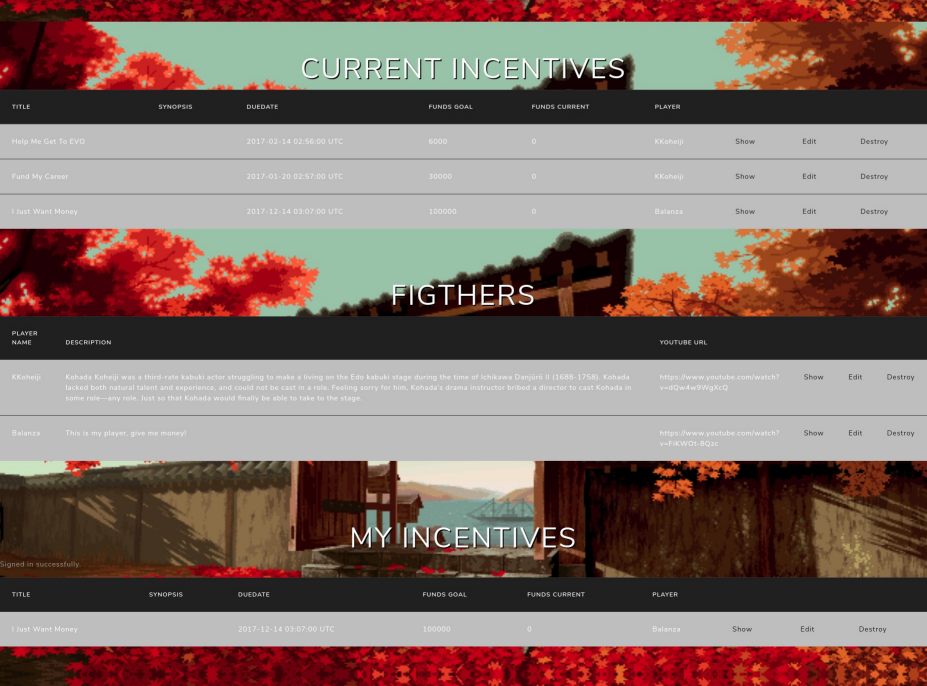
There were no special data structures or algorithms used for this application other than the ones already provided by the Ruby on Rails framework and the installed gems.

User Interface Design & Implementation

The user interface was designed using a combination of HTML, CSS, and Javascript, although Javascript was not used heavily. This was because we felt that HTML, CSS were our strongest languages, and thus, the coding and maintainability was going to be smother and faster. We also want to expand to a cross-platform experience in future, and this languages are easily implementable.

The UI was developed with the Bootstrap framework, which greatly helps in the coding, since we won’t have to reinvent the wheel. Additionally, we used a third party theme called LUX, by Bootswatch. We used this theme because it conveyed the spirit we want to convey in our website.





Current State & Future Work

The application is already functional without payments. Fake payments, were implemented instead of real payments with the Stripe third party application. All the requirements for the scope of this class however, were met. This application is not currently supported on mobile, since we have only adapted it for PC browsers.

For future works we wanted to implement payments and the administrator page for the future. We also want to add the milestones model, the campaign model, the FAQ model, the Comments model, Updates model and the Community model.