Rogelio Sergio Ramirez III

Austin Karingada

CSCI/CMPE 3340 Report- Unit Creator

# Project Summary

The goal of this application was to create a web application that would allow BattleTech players to create their own military unit for campaign play. Users would be charged a reoccurring monthly payment for access to the tool and the various military units they had created. Non-paid users would have no access to the application.

# Summary of Individual Contributions

Rogelio Sergio Ramirez III and Austin Karingada are the designers and developers of the application.

The initial project that provided basic routes, templates, and user integration was provided by Eric Martinez for the class using Ruby on Rails and Devise.

Rogelio Ramirez was the primary developer for the backend and UI of the web application.

Specifics:

* Created the mock ups for the signed in and signed out pages.
* Designed the Unit model and implemented create / edit / and destroy options using the DashboardController.
* Integrated the Unit model to the database using Active Record. (Scaffolding was not used in order learn how it worked, and to implement it)
* Implemented the routing structure to separate logged in and logged out users

Austin Karingada

# Requirements

The Unit Creator has the following requirements:

* Users must be able to register
* Users must be able to log in and log out
* Users that are not logged in, should not be able to access the unit creation portion.
* Users should be able to add an existing Unit (on paper) to their account, and edit or destroy as needed.
* Users should be able to use a wizard to create a Unit using the rules found in Campaign Operations.
* Dashboard should be reactive to the Unit that is selected on the table.

The system needs to support Admin users, Paid users, and Non-paid users.

Administrators should have the following:

* Edit User accounts.
* Add/ Modify/ Delete Units
* View all Units, no matter the creator

Paid members will be able to use the unit creator and access their Units.

Non-paid members will have access to no information.

# Third Party Integration

The system should allow users to purchase a monthly membership with a credit card. The chosen platform is Stripe because of how common it is, and that it reduces the risk to the system by not keeping sensitive financial data on the server.

# Interaction Diagram

No

No

Yes

Yes

Pay

Cancel

Profile

Add/Edit/Delete Unit

Unit Creation

Wizard

Paid?

Log in

Home Page

Logged in?

Register

Dashboard

# Class Model

|  |  |  |
| --- | --- | --- |
| User | Unit | SubUnit |
| Email: String  Encrypted\_password: String  Id: Integer  Role\_id: Integer  Last\_paid: Date  Paid: Boolean | User\_id: Integer  Unit\_name: String  Unit\_type: String  Admin\_people: Integer  Tech\_people: Integer  Rating: Integer  Balance: Double  Ocost: Double  SubUnits []: SubUnit | Unit\_id: Integer  SubUnit\_Type: String  Ammo\_cost: Double  Spare\_cost: Double  Fuel\_cost: Double  Crew: Integer  Officers: Integer  Rating: Integer  Num\_of\_machines: Integer |
| Admin\_only()  Paid()  Current() | Operating\_cost()  Avg\_rating()  Total\_people() | Num\_people()  Total\_cost() |

The User is created by Devise and includes all of its standard attributes and methods. The Admin\_only() and Paid() methods return a true or false. Current() also returns true and false by checking if the last paid date falls in the month time frame.

The Unit model keeps track of the overall statistics for the military unit in BattleTech. All of these attributes can be set by the user using a create/ edit interface or by using the Unit Wizard. The SubUnits array keeps track of the actual military forces associated with the unit, vs support personnel such as Administrators and Technicians. The methods allow for basic numbers to be extracted from saved information.

The SubUnit model keeps track of the operating costs and personnel for that military force. It can also accommodate multiples of the same unit type.( IE 107 HUMMVEES) The User will input most of this type of information.

# System Architecture and Design

The web application was developed using Ruby as the primary programming language for the back end and Rails for the web application framework. The production database is PostgreSQL, while development uses the light weight Sqlite3. The frontend was developed using Embedded RuBy (ERB) templating language that delivers the HTML to the browser.

The Model-View-Controller pattern was adopted and used to try to divide the responsibilities of the code.

The models used are:

* User
* Unit
* SubUnit

The controllers used are:

* Application Controller
* Admin Controller
* Home Controller
* Dashboard Controller

The Application Controller is part of the standard Rails application creation, did have one nonstandard change. This is the controller that automatically redirects people after logging in to the dashboard.

The Admin Controller is just used to check to see if the user is an Admin.

The Home Controller and views display the landing page that shows the benefits of using the Unit Creator.

The Dashboard Controller and views are the primary components to the web application. Its primary purpose is to get the current information of the user and do all the creation / editing/ destroying for the units the user creates. The unit creation wizard is also part of this controller and view.

The Rails router maps all the different actions of the controllers to web addresses that are used when linking the webpages for the application.

The models are Ruby classes that correspond to database tables that the gem ActiveRecord manages. The attributes are typically loaded from the database.

The application is being hosted on Heroku due to widespread use and ease of use. Heroku provides the hosting and database services needed.

# Algorithms and Data Structures

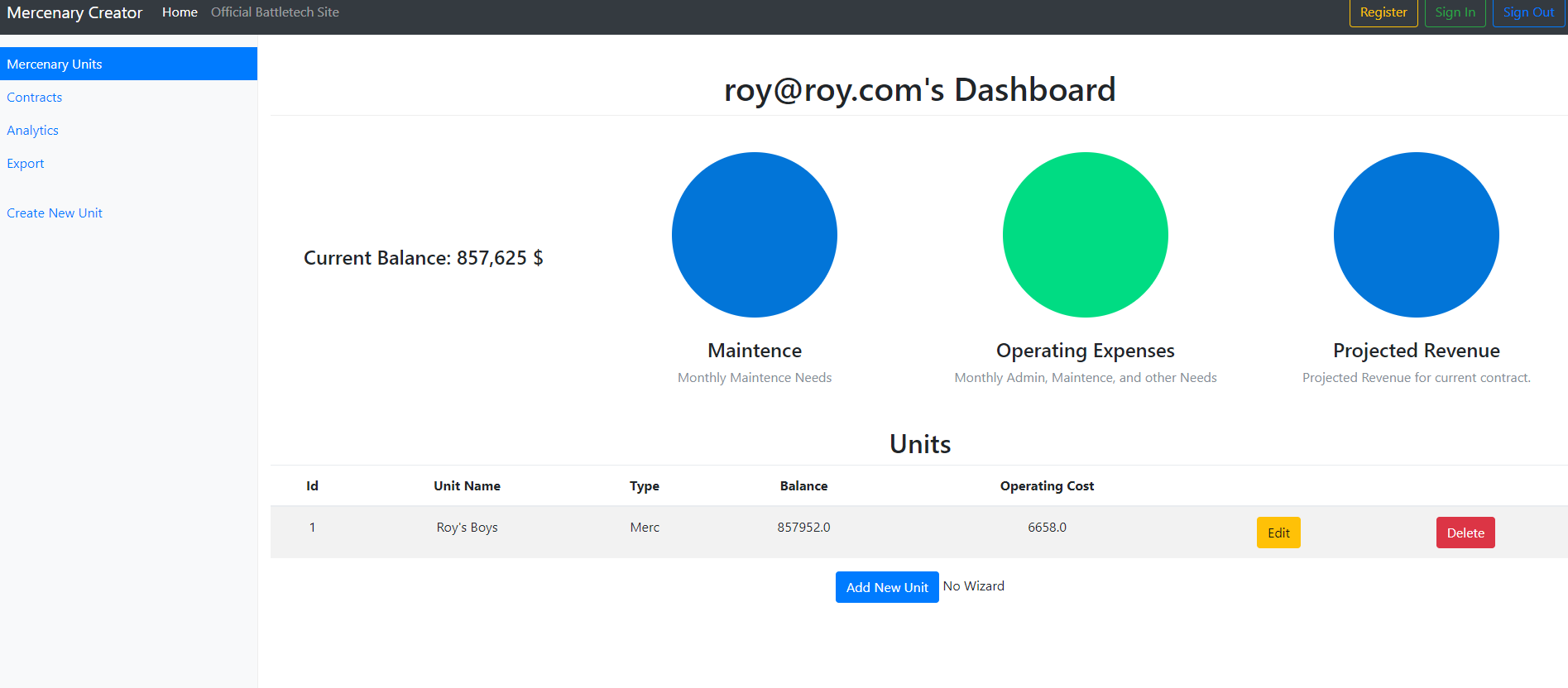
No noteworthy algorithms or data structures are used in Unit Creator.

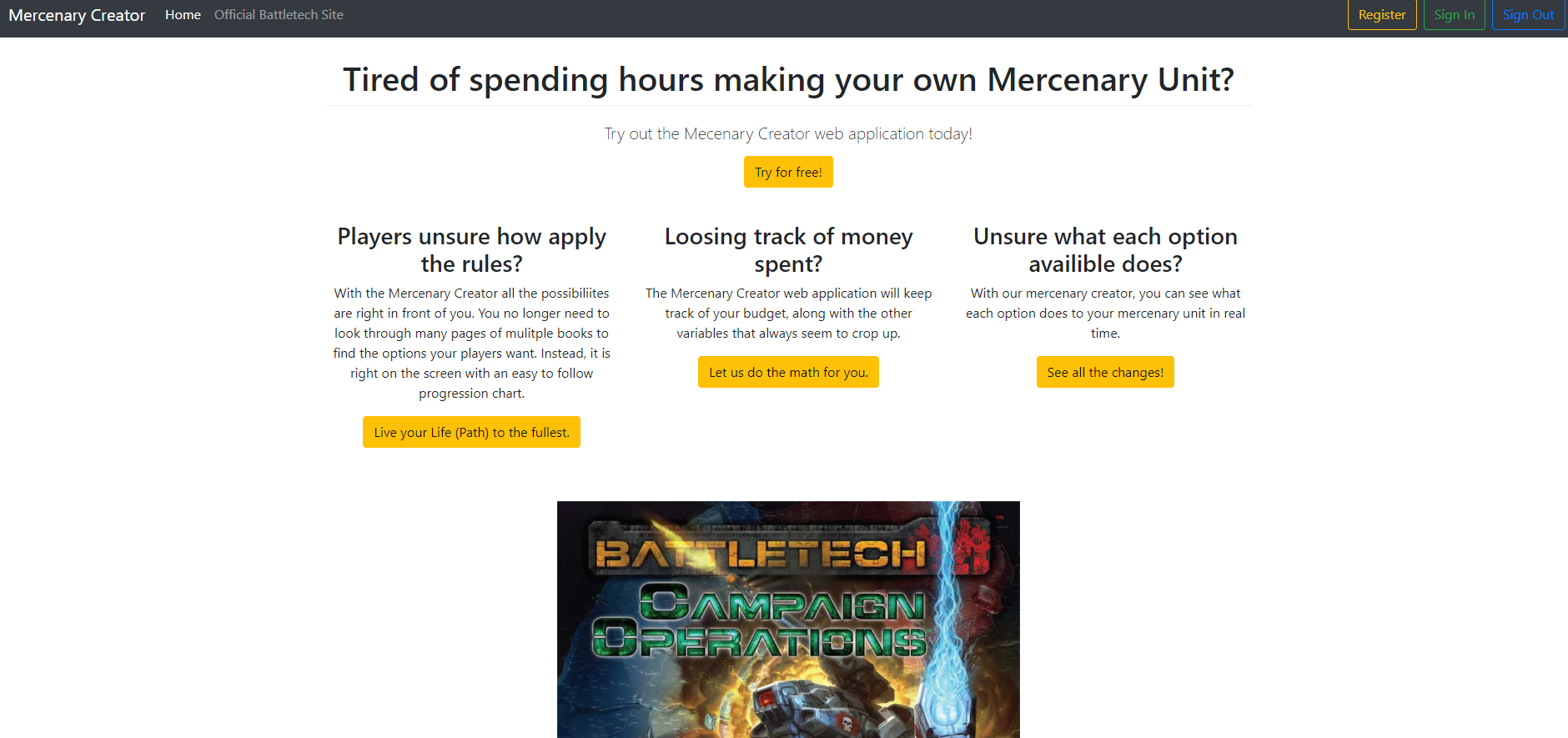
# User Interface Design and Implementation

The user interface uses common browser technologies to display the web application. It is mostly customized CSS and basic HTML. The expanded functionality is added by Bootstrap. Bootstrap comes with its own custom CSS and JavaScript files, which are loaded from their recommended content delivery network.

Bootstrap was primarily chosen due to the fact that this technology was the one we were most versed in. Mobile viewing was not on the forefront during the development, but with leaning on bootstrap for most of the styling, it is hoped that a transition to a mobile experience will be relatively easy, should the need ever arise.

The UI was developed over two free Bootstrap templates that were available. The starter template and the Dashboard template available at: <https://getbootstrap.com/docs/4.0/examples/>.





# Current Status / Future Work

The web application is partially functional and most of the features are not yet implemented. The major sections that need to be completed are:

* SubUnit integration
* Administrator rights
* Billing integration
* UI customization
* Dashboard Responsiveness

There are many potential additions that can be added, but there are two that should be considered above all. The first and most important one would be to somehow integrate all the military units found at <http://www.masterunitlist.info/>. This would allow the user to select the units instead of entering it by hand, vastly reducing creation time. It appears that the site had no API, so most likely, web scraping will have to be done. The other feature that should be added is the ability to share units with friends, and maybe make the Unit information available to everyone. The unit creation would still be a paid feature.