SWEN30006
Assignment 3

Members
Avnish Jain: 607518
James McLaren: 524772
James McMahon: 513250
Jake Naughton: 538700

Summary of Changes

After being granted the request to combine two groups into one, our final submission would incorporate the 'better' parts of both groups' previous projects. We first had to distinguish the subtle differences in our MVC component diagram from project 2, for example, which models were necessary and which functions different controllers would implement. While these components were in a sense 'set in stone', our views components from Project 1 were left open and our designs challenged by our ability to implement HTML and CSS. A summary of changes between our initial projects and our final submission is stated below:

- Initially, one of our groups did not include a tickets model, however the model was included in our final design. The easy configuration of models in rails (a simple 'belonged to events and users') allowed easier control of keeping track of which club issued tickets and users who had reserved them.

- Both groups had implemented the concept of a subclass of user named Club Admin, however we decided late in our programming to utilize joint tables to track admins of particular clubs and removed the model. This made both the programming easier and the design simpler.

- Some additional models with joint tables were event_follows, comments and club_events. Aside from 'club_events', none of the listed joint tables were part of both our initial design. The event_follows model helps users track updates (i.e comments) of events that are subsequently added to the Announcements on the Home Page.

- Our final design looked very different from our mock-up wire-frame designs we constructed for Part 1. For example, our final homepage did not include the 'recent events' feature that allowed user to see descriptions and photos of past events. This was far too challenging and time consuming to program in the short time frame given. Similarly, other designs issues were the club admins ability to invite other clubs to an event, send and receive notices and manage and complete tasks. In place of notices, we used the event_follows and comments joint tables as explained above, a more efficient implementation with the same functionality as notices.

- Though we lost some of the potential functionality of our wire frames, we certainly improved the overall look, adding colour (unfortunately purple), shadows and better-designed, concise tables with headings such as tickets available, start times and web-links. This was a huge improvement on making the website more user friendly and easier for different end-users to access the required information.

During our exposure to Ruby on Rails, we found it surprisingly easy to add tables, inputs, change layouts etc. However, given the limited time, an executive decision was made to focus on improving particular sections rather than 'creating more' and hence potentially losing functionality and usability. That being said, our designs and diagrams in Part 2 provided us a firm structure to guide us through the implementation and definitely made the overall process much easier. We found for the most part that we stuck to and rather improved on our initial design rather than implement a new one.

FINAL REFLECTION

Our general approach to implementing this project was to dissect the series of issues the would arise from our lack of familiarity with Ruby on Rails. Important to all software design is not just being efficient with time but completing such in a given time-frame. Obviously not all barriers would be foreseeable but it was important to limit our programming to a feasible goal, which we could accomplish in the time given. From this we deduced it necessary to focus solely on the basics, that is the requirements of the system outlined in the project guidelines, and neglect any extensions until these were finished. Further we prioritized reliability and layout over functionality, to keep our programming in check and avoid hazardous risks involved with larger systems.

In lectures we learnt how the Model – View – Controller software architecture and its features makes the overall design and implementation procedures more efficient and easier. In combination with the use of GitHub, these 'teachings' were almost immediately realized as we had different members of our group simultaneously working on different aspects of the project such as the controller logic, data migrations, and the HTML and CSS of the views with minimal conflicting errors. However while rails matches the MVC design pattern (which we had chosen) it was a lot more work to implement our user experience design, as none of the design team had any experience with HTML or CSS before this project.

Our inexperience with the rails framework led to us having a poorly laid-out design plan. Rails can automatically generate scaffolds as well as individual models, controllers or views but many elements of our implementation required significant work, including ongoing tinkering in order to compensate for obstacles found.

The hardest part of the design process was the fact that there were a few core functions that the application depended on, so if these were not implemented properly it could halt the implementation process. For example, in our events controller, we have a function that determines whether the current user has the necessary privileges to delete an event. While this is a relatively small function in the scale of things, if one person were to implement that function partially and then another person were to come along and add additional functionality to event deletion they would be unable to test their implementation until the initial function was working correctly. While it is true that this issue could have been overcome by using git branching (or a similar version control function) this was an area in which we were unfamiliar. Another reflection on our design process was the fact that it would have been greatly improved had we used something like agile development. There were often times when we were a bit unsure as to what needed to be worked on, and spending a bit more time at the start sorting this out would have greatly improved efficiency.

Regarding administration rights for clubs, our design plan was to have a specific ClubAdmin class inheriting from the more general User class that would be associated with one particular club, administrators of the club would be able to log in using the ClubAdmin username and password. unfortunately by having two Devise (the authentication gem) classes became very complicated very quickly, and on advice of the tutors we decided to implement a join table between users and clubs that would imply admin rights.

Though Ruby and Rails were both unfamiliar to the group initially, we were glad with the choice due to the robustness of the language and framework. A lot of the code was generated through the command line with a majority of the complex aspects, handled by gems. For example, we did not have to worry about image, log in and registration, and dynamic browser design management because of gems like Paperclip, Devise, and Bootstrap.

Another positive was how our design in Part 2 readily fit in with the Rails framework, which meant that the implementation part of the project was simplified and far more structured. As mentioned in our 'Summary of Changes', our group for the most part rather improved on our initial design by adding functions, models and joint tables, rather than abandoning it and implementing a new one.

Clearly as we developed our project the recurring problems revolved round our inexperience which was counterbalanced by the usability of Ruby on Rails. Outlined in our summary of changes we see a vast divergence from our initial mock up wire-frames to the final implementation. Given more time our design would have refocused on usability and functionality, making the website more user friendly and practical. An example would be introducing a 'recent events' scroll bar on the homepage. This would have made the website far more attractive and generated a lot of activity by allowing users to experience other clubs outside of their own, also encouraging new members to sign up. It would have been nice if clubs could invite other clubs to host and event, a very practical function for club admins and a necessary addition if the website was to be used.

As we developed our project one of the ideas we came across which was not in our initial design and a facet we did not end up implementing was the 'Facebook way' of sharing, liking and following clubs, events and users. Seeing as Twitter was constructed using Ruby on Rails the were many online tutorials and advise about generating such a functionality. As we approached the final due date however, we had a series of unfortunate instances involving bugs and errors which at times took hours to resolve. Our primary goal as stated in the introduction was to make the website reliable so we backtracked and fixed the existing bugs rather than taking the risk of creating more, so eventually skipping the 'Facebook way'.

It is also worth noting that we definitely improved the overall layout of our website, since in our original projects we were unsure of complications involved in such programming. As we discovered editing and improving views proved fertile, additionally beneficial they proved not to generate bugs. This came from the easy learning curve involved with html and vast availability of resources found for it online. Some examples of where we exemplified our previous designed was displayed in the shadowing behind our tables and the colouring of the borders and titles. Further we expanded the number of headings in our tables, we did not initial ignore these because we thought it would be to hard to program, but because we would be limited for space, which proved to not be a restriction.

A last point with respect to performance, given a larger time-frame we would have liked to create user profiles such that other user could plan their social lives around their friends. This would have included a display of events a user is attending and clubs they have joined. Ultimately with improvements stated above our website may have even proved a realistic website for the actual unimelb clubs to use, unfortunately we would need further exposure to using Ruby on Rails before tackling such a task.

For a final reflection on our project, it was quite admirable to come so far with no exposure to this method of programming. Impressively our project resembled alot of our design and architecture. It was also very interesting to combine two groups and elaborate on a final design involving the better parts of our previous projects. This had the downfall of consuming a lot of the time but we benefited from a better design than we could not have hoped for from just one group. Delegating tasks was also tricky as our original plan was no longer applicable for four people. This had to be extended and re-accessed based on each individuals ability and skill. Overall The project would never have come together as smoothly as it had without our MCV architecture and our UML's. This project taught us that the key behind achieving a great implementation is the effort and time put into the design. Hopefully we will be able to take this skills with us and apply them outside of university where they are needed!