ECE 458: Evolution I
Resource Manager

Writen Analysis (25%) Along with every software deliverable, you will turn in a written document which will cover two main points:

~~1. A retrospective on how your previous design choices impacted your work to meet the current set of requirements. This section should analyze not only where your good design choices made things easy, but also where your bad design choices made things hard. For both of these points, you should analyze how/why these design choices helped or hindered you. For bad design choices, you should discuss what you might have done differently in the past to avoid the problem this time around.~~

2. An evaluation of your current design, with an analysis of its strengths and weaknesses going forwards. This section should justify your current design choices, explaining why you think they will be beneficial to you in the long run. If you recognize weaknesses in your current design, you should discuss them—including an explanation of why they are there, and how you plan to fix them in future submissions.

These documents should not only deep analysis of the strengths and weaknesses of your design choice, but also be well written. Ideally, the retrospective section of submission N would connect back to the forward-looking analysis of submission N-1 (i.e., Did things you think would be beneficial actually end up helping you? Did the weaknesses you identified come back to bite you? Did you fix your weaknesses this time around?).

Analysis of Design:

This application is built with MeteorJS, a full-stack javascript platform for client-server web applications. In choosing this, we took into account many of the considerations from the 1/20 class discussion on programming languages. First, it allows us to use one language (javascript) for all application code in the project. This is relatively rare for web applications, which usually have javascript client-side and some other language on the server. Second, the environment surrounding the language (meteor) is a full-featured framework with a widely used package system, allowing us to use 3rd party modules for common tasks. Finally, the framework also handles synchronization of data between the client and server on its own.

The design of our current application centers around the data model and the use of a few meteor packages for various pieces of functionality (searching, calendar display, etc). The data model is straight-forward, with MongoDB collections for resources and reservations. This simplicity in the data model may add work in later evolutions; for example, we chose to store *Tags* simply as a list of strings on each resource. If, later on, tags become a more important/complex piece of the requirements (e.g. tags have owners themselves, associated permissions), we may want a separate tags collection, and changes will need to be made in the model as well as in the application logic to accommodate.

We use two major packages to implement primary features: *filter-collections* for filtering, and *fullcalendar* for the calendar UI. While it was a major benefit in reducing development time and complexity of our application, our use of 3rd party packages can also be considered a vulnerability. Changes in the requirements in coming evolutions may call for functionality outside of that which can be provided with our current packages, and we *may* face the choice of re-implementing certain features or grafting more of our own external code onto the packages. However, the packages that we chose to use are relatively simple and self-contained, making it unlikely for us to have to alter our design regarding them too much. For example, there was a bug in the *filter-collections* module we used for resource search/sorting that caused it to be unreactive --searching only happened upon refresh, instead of asynchronously. We were able to make it asynchronously load with an external fix. Changes to 3rd party packages at this level are small and do not add any complexity at the level of design/programming in the large.

We use a 3rd party package called FullCalendar to display reservations to the user. While we hope this package overall improves development time, we could envision several issues which might come into play concerning the use of this 3rd party package. The first is that our reservation data must be converted into an event object that FullCalendar accepts. As reservation objects grow in complexity, we'll need to make sure the key mapping to the FullCalendar friendly event object is done efficiently in a central location or risk inconsistent calendar behavior. The second issue we could envision has to do with the lack of extensibility of this 3rd party package. This worry was lessened by the fact that we can hook into the HTML content of each calendar event, for example we were able to add a delete button to each event. We could still come up with scenarios which might require editing the source code of FullCalendar (allowing a user to copy and paste a reservation, custom event hooks), so we decided to fork the package to our GitHub organization and add it as a submodule to our project. This allows us the option of customization while still keeping the package up-to-date with the upstream project. Lastly, we worry about the fact that FullCalendar was not written with the design patterns and philosophy found with the Meteor Platform. Meteor is all about reactivity, or 3-way data binding. As soon as data updates in the database the local storage should update, as should the HTML content displayed to the user. We were able to achieve reactivity with FullCalendar by setting up a change listener for a specific database query of reservations. If the user is looking at reservations with a calendar date range of February 8-14, the database sends a notification whenever a reservation with this date range is added, changed, or removed. We think this listener system works well, but could see needing to revisit this system as requirements change and complexity grows.

Contributions to this evolution:

Sam:
- login system
- base meteor functionality
- admin creating/editing resources
- admin enrolling new users

Allan:
- Search/filter
- Tags filters
- Filter-collections bugfix

Emmanuel:
- search/filter

Davis:
- reservation calendar display, reservation detail view

- server methods for managing reservations