ECE 458 Evolution 1 Write Up

Oscar Wang, David Zhou, Andrew Gauthier, David Zhang

February 10, 2016

1 Design Choices

1.1 Languages and Technologies

We decided to use Ruby on Rails to create our web application. We chose Ruby on Rails because it does much of our work for us. The default setup uses a model-view-controller (MVC) framework that keeps the code very organized. Using Ruby on Rails also gives us access to many existing resources; there is plenty of documentation on the internet for connecting our code to external components, as well as gems for easily accomplishing various tasks. Most of us also previously had experience with Ruby on Rails, so the learning curve would not be as harsh. We came to the consensus that Ruby on Rails offered the highest return on effort, which made it our language of choice.

We decided to use PostgreSQL for our database and Heroku for deployment. We made these choices slowly for the sake of familiarity, and in retrospect, they were probably not the best choices. We ran into many problems with Heroku and PostgreSQL on Windows machines, which prevented half of our team from coding on our personal machines. We are in the process of setting up virtual machines through the Duke VM manager, and using those for our production and test servers in later evolutions. We learned in class that the virtual machines are also supposed to be faster than Heroku, so this will be beneficial to our project in the long run.

For the front end, we decided to experiment with React.js. This is something that none of us have experience with, but we have heard good things about it and decided it would be nice to learn something new. Hopefully, React will help us create a more impressive front end than what would be possible with Ruby on Rails alone. We also used the Devise gem for user authentication and Sendgrid for sending email notifications.

1.2 Program Organization

Our program uses a conventional Model-View-Controller pattern for organizing the code.

- Models: Our program uses four models to implement the functionality required for this evolution. The first is the user class, which is designed to work with the devise gem and allows for differentiation between basic users and admins. Some altercations from the devise gem had to be made in order to allow the admin the create users, as opposed to users just signing on. The second model is the resource class, which contains methods for manipulating its tags. The third model is the reservation class, which can check for overlapping reservations. Finally, there is tag class, which is used to organize the resources.

- Controllers: Our program uses three controllers, reservations, resources and users. The controllers are implemented using the CRUD actions, making it easy to create, update and delete the necessary objects. Most of the methods are pretty standard rails methods, and work with the parameters from the given forms from the view. There is no tags controller because the tags portion was a little more tricky, since all the work was done relative to the resources (tags are created when the resource is created). In addition to the CRUD methods, a removetag method was added to the resources controller, so that one can remove tags for specific resources.

- Views: Our front end is organized into html files for each page in the program. They are packaged based on which items in the back end they manipulate. In addition, forms are used to pass parameters from the front end to the backend. The same form is typically rendered multiple times so that the same code did not have to be repeated multiple times. While the front end is currently all html, we hope to add some styling to it via css and potentially bootstrap, as well as make it more interactive via javascript and react next evolution.

2 Evaluation

There is definitely a lot of work left to be done, but our current program is at least functional and satisfies most of the current design requirements. One of its strengths is its current simplicity; we designed it with the intent of making it as flexible as possible, and tried to anticipate extensions such as recurring reservations and multiple admins. The biggest weakness of the current design is the front-end, which is not as fleshed out as we would like it to be both because of our unfamiliarity with React and the problems we encountered with Postgres. Our two goals moving forward are getting the program fully operational on the virtual machines and making the user interface more polished.

3 Contributions

A lot of our work for this evolution was done as a group. We first met several times to plan and organize the project, and then got together several times to code together using a projector. As we discussed earlier, problems with Postgres prevented us from working on our personal machines.
Individual-specific contributions are listed below:

- Oscar Wang: Set up the Ruby on Rails, implemented Devise and Sendgrid, deployed on Heroku, set up React
- David Zhou: Wrote the write-up, worked on setting the program up on the Duke VM
- Andrew Gauthier: Prepared the presentation, worked on setting the program up on the Duke VM
- David Zhang: Worked on planning out models and relations to strategize how current design can be extended to implement future features