Evolution Four

ECE 458

Parker Hegstrom (eph4)
Peter Yom (pky3)
Wayne You (wxy)
Brandon Chao (bc105)

Abstract

Contents

1 Overall Design

The overarching design principle we wanted to achieve was modularity. In doing so, we believed we would be able to work separately (with occasional meetings to work through minor problems with the API) and refactor without the worry of breaking another member's code. Figure 1 below shows a high level diagram of how we decided to design our calendar web application.

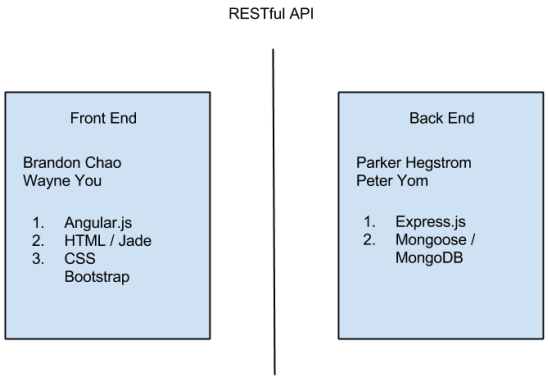

Figure 1: Diagram of our Large Scale Design

Essentially, our back end team provides an exhaustive RESTful API service to our front end. As we received the new requirements for evolution two, the benefits of our modular design came to light as we met to discuss both the refactorings from evolution one that needed to be done and the edits to each modules system design in order to account for the added calendar functionality–event requests and persistent until done events.

The following sections further discuss design choices and implications of those design choices for both our front end and back end teams.

2 Back End Design and Analysis

2.1 New Features and Developments

2.2 Benefits of Our Previous Design

2.3 Drawbacks of Our Previous Design

3 Front End Design and Analysis

3.1 New Features and Developments

3.1.1 PUD Events

The extra features to PUD events including an expiry time and priority escalation were handled on the front-end by simply adding fields to the PUD creation form and sending this data to

the back-end. Both of these features took front-end input but the functionality was handled seperately in the back-end.

3.1.2 Preference-based Sign Up

3.1.3 En Masse Event Creation

We implemented en masse event creation using simple text parsing. Since we have the user input the event with a Name, Description, Start Time, and End Time on separate lines, we simply split the input string based on the new line tokens and create separate events for each series of 4 lines of text.

3.1.4 Schedule by Email

Our front-end implementation of this feature was fairly easy. For text schedule, we simply parse through all of the events within the current display frame and send them to the back-end. For image schedule, we take a screenshot of the calendar in the current display using Javascript's html2canvas() function and send it to the back-end to email.

3.2 Benefits of Our Previous Design

The implementation of the features in this evolution were fairly straightforward due to our design coming into it. Many of the features seemed to be more back-end oriented which also made our lives a lot easier in this evolution. The only completely new feature was getting the schedule by email, which was a matter of parsing through our list of events we already store for text-based, and used a Javascript function to accomplish the image-based one. Even mass event creation was not hard to create due to the way our events were being created in the past. We merely had a for each loop to create an object with the data for each event we parsed through, and then called our previous event creation route on each event. We really saw the benefits of a flexible design coming into this evolution based on the requested features which allowed us to complete this evolution quickly.

3.3 Drawbacks of Our Previous Design

4 Individual Portion

Parker

a) **Designing and Conducting Experiments**

.

b) **Analyzing and Interpreting Data**

.

c) **Designing System Components**

.

d) **Dealing with Realistic Constraints**

.

e) **Teamwork and Team Member Interaction**

.

Peter

a) **Designing and Conducting Experiments**

.

b) **Analyzing and Interpreting Data**

.

c) **Designing System Components**

.

d) **Dealing with Realistic Constraints**

.

e) **Teamwork and Team Member Interaction**

.

Brandon

a) **Designing and Conducting Experiments**

- Wayne initially implemented getting the schedule in image format by using the html2canvas() function to take screenshots of the page based on the DOM. However, the limitation to this is that we can only get the image of the time range that is currently being displayed. As such, I experimented with taking and storing images of multiple weeks/months and sending them all at once to the user. I did this by changing the view of the calendar (and the DOM) and taking a screenshot at each requested month using the same function. However, this didn't seem to work because I was not able to change the view of the calendar without it rerendering on the front-end which caused it to keep changing the display. Also, I was having trouble sending an array of image data URLs to the back-end to represent each week or month that was requested. Because of these issues, we ultimately decided to limit the functionality of getting the schedule in image format to only send the current time frame to the user.

b) **Analyzing and Interpreting Data**

- When I was implementing reordering of Slot Sign Ups to allow the owner to decide slots based on their preference, I ran into the issue of the slots not updating their reorder in real-time. However, when I printed them out after grabbing them from the back-end, they were in the expected order. This led me to believe that whichever scope variable we were using to display them on the front-end wasn't being updated correctly. I confirmed this by printing out the array of objects for the ordered slots from the Angular code and saw that they were indeed incorrect. This was solved by updating the scope variable upon successful execution of the HTTP PUT. We have run into numerous similar issues in the past which made debugging this fairly easy. However, if I had time to continue working on the calendar, one thing I would refactor is cleaning up our controllers and which scope the variables were placed in by creating a better hierarchy of controllers.

c) **Designing System Components**

- One component we had to design was the detail display for preference-based signups. We decided that we wanted the owner to view the Slot Sign Up details modal unless all requested users have signed up for slots already and it was time for the owner to resolve the conflicts. In that case, we bring the owner to the Slot Sign Up resolution modal. After resolution, subsequent clicks of that particular slot sign up will once again take the user to the details modal. We achieved this by maintaining two flags stored within the scope for the particular slot sign up. We maintain one flag to check if the preferences for slot sign ups are complete, as in all requested users have signed up, and one flag to check if the preferences for slot sign ups are final, as in the owner has resolved any conflicts. Each time the slot sign up is clicked, we check these flags to determine which modal to display. This currently does not allow the owner to edit a slot resolution after they submit it, but adding this would simply require a button that would toggle the final flag for this particular slot sign up which would allow the owner to make changes after submission.

d) **Dealing with Realistic Constraints**

- One realistic constraint that we had to consider was when we were implementing the graphical schedule by email. We did this by taking a screenshot of the current time frame of the calendar. While this wasn't very difficult, the nature of the html2canvas() Javascript function we used to do this made it difficult to get a graphical display of calendars that you were not currently on. For example, to get a graphical schedule of the next month, you would need to manually click next month and then send the image file. We experimented with using various other functions to achieve this, but the html2canvas() function provided the easiest implementation.

e) **Teamwork and Team Member Interaction**

- We made an effort to start implementing the features of this evolution a lot earlier this time. As a result, by the night before it was due, we had a lot fewer pressing errors to resolve last minute. I felt that teamwork and team member interaction went a lot more smoothly because of this and because we met up in person earlier to work out bugs.

Wayne

a) **Designing and Conducting Experiments**

.

b) **Analyzing and Interpreting Data**

.

c) **Designing System Components**

.

d) **Dealing with Realistic Constraints**

.

e) **Teamwork and Team Member Interaction**

.