SoCal Evolution 1 Write Up

Peter Yom, Parker Hegstrom, Brandon Chao, Wayne You
ECE 458

SoCal

1 Back-end

The MEAN stack gave what was believed to be the best tools to develop a web application. A few team members were familiar with tabular based SQL databases, but the NoSQL MongoDB appeared to be much more powerful and more appropriate for the calendar application. Using Node.js with the Express framework forced us to use and learn Javascript, but it also allowed our front-end and back-end to be unified under just 1 language. Communication between our view and back end was very simple as we were able to directly pass JSON objects back and forth. Express allowed us to route to various HTTP requests very easily. App.js is the main hub for our entire back-end. All requests initially go through the app.js before being routed to the appropriate file. The majority of our work on the back-end was spent creating different route files in the routes folder. Each route file is prefixed with a unique endpoint. For example, the calendarRoutes.js file is routed to with the end point /calendar/. Within calendarRoutes.js, we would add the code for GETs, POSTs, PUTs, and DELETEs as well as any functions.

2 Database and Route Design

The first steps done were to design the database schemas and the API. The User schema holds a users?s login information, salt information, a list of calendars the user owns, a list of calendars the user can modify, a list of calendars the user can view and a list of calendars the user can view as busy. User authentication was done through Passport: a user authentication middleware. It automatically creates secure, salted authentication, tracks the current logged in user in a session variable, and automatically logs a user out after a period of inactivity. Calendars hold a list of its events, a list of users who can modify it, and a list of its rules. Event sharing was made to be on a Calendar granularity. When a user wants to share an event or a series of events with other users or usergroups, the user actually shares an entire calendar. The user is able to create multiple calendars and create events in any of the calendars that it is able to modify. Rules are given priority based on the position that they are in a calendar?s list of rules. A newly created rule is by default at the highest priority. When a rule is deleted, the effects of that rule are removed from the users and the list of rules are run through again to propagate the effects.

Alert information was initially a part of the Event?s schema, but were then separated into their own collection. It became much easier to query on a collection of alerts as opposed to querying and searching through Events for the appropriate Alerts and then populating.

3 Web Client

The web client was designed using Bootstrap, AngularJS, and Jade. Jade is a templating language to create cleaner HTML. It came with Node?s Express framework, and was adopted for a more readable page template. While it provided a less cluttered view during development, the HTML it produced contained no whitespace, so debugging improper tags became much more difficult. Bootstrap was chosen for a better looking website and access to many CSS classes. Bootstrap also contains JQuery, but we decided that AngularJS provided us with more options in development.

The client itself is composed of essentially one page with three major areas in a single AngularJS application. The first major area is the calendar itself. This calendar was pulled from Serhioromano?s Bootstrap Calendar. The calendar can store events and display them in four different, navigateable views. The second major area is the sidebar where events, calendars, usergroups, etc. are listed. All lists the user can access are populated into the sidebar where the user can navigate for more details. The last major area is the lower detail pane. This area is populated with details pertinent to the user?s actions in the sidebar. As of now, it holds event details as well as the event creation form.

4 Web Client Design

The client is held in a single Angular application with two controllers handling the sections for the detail pane and the sidebar. The main application is in charge of handling event data, event requests, and calendar requests. The sidebar is in charge of changing the lists which are populated using Angular directives. The Angular directives tie Javascript data to HTML elements to dynamically populate them. The detail pane controller manages the event forms and event details. However, due to the storage of data, the root application controller tends to format event data and handle the view status of the detail pane. For the sidebar and detail pane, the views are handled by changing values for hiding certain elements, which may be changed to using AngularJS?s UI Router if or when the project requres it.

The calendar also requires a set of data that is much more limited than the information that we store, and it requests the information in a different format. For this reason, we are required to re-format the data we receive and the data we send for each transaction. The calendar also lacks a proper means of updating itself when new events are added. For this reason, we will most likely change to using an AngularJS extension made by Mattlewis92 for this calendar. The new calendar will support AngularJS data binding for events and reduce the amount of formatting necessary.

The only lingering design concerns stem from the amount of logic run by the client and the frequency of HTTP requests made. Both are high, but determining exactly how much we want to limit in the client is important.

5 Designing and Conducting Experiments

5.1 Parker

One experiment I designed came about when I realized I did not understand the behavior of a certain Mongoose API call. Specifically, I needed to delete a document from the database

and decided to use a combined function provided by Mongoose: findByIDAndRemove(¡query¿, [args], callback). The experimental process I derived to help elucidate the return values of this function was composed of executing the method multiple times with different arguments and comparing those permutation with my interpretation of the documentation. I then proceeded to console.log() the return values. This helped not only visualize the return values by seeing the values, but also get a sense for the structure of the return object. Was the returned objected executable? JSON? Able to service further queries? I was able to answer these questions by my iterative and extensive methods.

5.2 Peter

Working with the back-end, I had to deal with a lot of JSON data. In order to test my various HTTP GET, POST, PUT, and DELETE requests in our API, I heavily utilized Postman. For example, in order to test whether a Calendar DELETE request has been successful, I had to make several other calls to set up the environment. First, I would create a few new Users with POSTs and then log in. I would then POST a new calendar with a calendar name, and I would check in mongo whether my calendar was created with the correct name and with the correct owner. I would then grab the new calendar?s ObjectId and use it to POST new Events and Rules. I would then have to check in mongo that the appropriate Users have been correctly modified and affected by the Rules. Once satisfied, I would then make a DELETE request. I would then have to make sure that the right Calendar has been deleted from the database, that any references to the calendar in any of the associated Users? modCalId, myCalId, canView, or canViewBusy are gone, that the Events tied to the calendar are removed, that the Rules associated with the calendar are removed, and that the affects of those Rules are all gone.

5.3 Brandon

I worked on the client side code for our calendar application and as such worked closely with Angular.js, Jade, and CSS. A large part of this included writing the AJAX calls to the Node API so our front-end Angular code could interact with Node and MongoDB. To achieve this, the AJAX calls needed to match with the API defined by our back-end team in Node. One experiment I designed was to debug an HTTP error resulting from a nonfunctional call to Angular?s $http service. I wanted to compare the JSON object printed out in the browser console with the object inside our database to help determine whether the issue was on the front-end or back-end. Thus, I was able to view the object by logging it out in the console and viewing it when I clicked the button that triggered the call. On the other end, I accessed mongoDB directly through my terminal and queried the database for the current state of the object in question. I found that they were different thus indicating that the back-end was not sending the correct object through the Node API for us to use on the front-end. After knowing this, it was easier to look through the relevant routes on the back-end with my team members and isolate the problem.

5.4 Wayne

The program design and testing was relatively easy after having defined an API for use between the front and back-end. This design

6 Analyzing and Interpreting Data

6.1 Parker

This answer follows from my previous experiment discussion. The data in this experiment with findByIDandRemove() method were the logged return values. By simply seeing the value print out in the console, I was able to interpret the form of the return value. A special feature of these query type functions in mongoose, though, is that they sometimes return a query mongoose object that allows you to invoke further query criteria before eventually arriving at a JSON object. Further analysis was done by attempting various queries and other mongoose method calls on the return objects to devise the type of the value/object returned by findByIdAndRemove().

6.2 Peter

Javascript?s callback functions are asynchronous and made life quite difficult for us at times. When I was testing a function in the Rule schema that returns all of the associated Users, I noticed that the function functioned perfectly fine. However, it did not return the correct results. Instead, It would only return a part of the answer or none of the answer, while a console log of the result would display the desired result. I realized that this meant that the callback functions in the function actually happened asynchronously and would complete after the function returned.

6.3 Brandon

Our group defined a very strong API at the beginning of the project and as such for the most part we did not run into too many problems with front-end and back-end integration. However, there were instances where I got blocked because of the API documentation being defined incorrectly. As a result, many times I debugged by logging the JSON objects sent over into the console to make sure it matched up with the specifications in the API. Because Javascript is dynamically typed, fields can be added in objects at any point and as such, we had to make sure all of the fields in our objects we were sending over JSON were the same. I was able to debug many API issues by analyzing the data printed out from the objects being sent over to Angular.

6.4 Wayne

The initial choice of tools was mostly due to convenience and an interest in learning to use AngularJS. The tools worked fine and did not require much changes in design to operate, but some issues have arisen during development. Jade is unforgiving with slight indentation issues. Like Python, it uses indents in order to determine scope, but many times the error is not apparent and works its way into the HTML and becomes impossible to locate properly. Javascript has many deferred value objects as Promises when making HTTP requests. This leads to race conditions within the same function sometimes causing data to not return properly for use, e.g. event information pulled from calendars not populating because of an undefined value until the HTTP request returns. Javascript also does not have strong typing, which leads to many problems from trying to determine if an input is failing due to a bad value or a bad type as many things resolve to Object.

7 Designing System Components

7.1 Parker

I designed the email alert component for my group. In summary, this component allows a user to bind an alert to a given event on the calendar. The server then checks the outstanding alerts in the system and sends an email to the user on file for that event at the correct time.

Before coding anything, however, I conducted some research on the availability of different timers in Node.js as well as what the general practice was for controlling a repeating database query. Fortunately for me, Node exposes a global interval timer.

Knowing the mechanism I would use, I then had to decide how to store the actual Alert type in the database to allow for easy access from the intervalCounter. The decision was made to store an Alert as its own model in the database, giving it its own collection?a collection I could query based on time parameters.

The final design decision had to do with determining the optimal interval time for the timer. To often, and I would slow down the server with frivolous requests, not often enough, and I could miss the alert time all together. Due to the fact that the smallest unit of time we offer to the user for setting an Alert time is the minute, I set the interval time to a minute

The remaining step involved ?plugging? my alert module into the app as a whole, which was done so by exposing it to the Express app variable in our main executable. Tests were carried out to ensure that the module worked as advertised and that the client could rely on the alert being sent within 30 seconds of the set alert time.

7.2 Peter

The problem above was to create a function in the Rule schema that would return all of the associated Users (the Users in its assocUsers array and the Users from its assocUserGroups array). This was first done by looping through the two arrays, populating the assocUserGroups with Users, removing duplicates, and returning the result. However, my use of callback functions to populate the UserGroups made my function work asynchronously. I fixed this problem by learning about javascript promises and passing in a callback function. After finishing my calculations, instead of returning the array of Users, I would pass the array in the passed in callback function.

7.3 Brandon

I designed the UI of the sidebar as well as the Angular controller for the sidebar that tied our Angular functions to the HTML that is rendered by the website. I wanted the user experience to be as intuitive as possible while still keeping the website clean and free of clutter. For the sidebar, we have buttons to allow the user to see and modify their user groups, the calendars they own and are added to, and the events on their calendars. In each of these sections, we display the main info, and then when the user clicks on a specific item (user group, calendar, or event), the view will render to display the details and relevant information and allow them to perform actions such as creating new calendars, managing the users that can modify their calendars, and add rules to their calendars. The whole sidebar is also tied to a sidebarController in Angular. At this time, we feel this controller is a little bloated with too much functionality, and in the future

we would like to split it up into multiple smaller files by possibly using UI-router in Angular to do the routing.

7.4 Wayne

I mainly constructed the detail pane and sidebar views for usergroups and events. Most sidebar views are simply lists which populate new views, such as usergroups. Upon clicking a specific usergroup, the sidebar will then display a list of users in that group. The title is updated using an AngularJS scope variable. If the list can be added to it contains a text input which sends the data to the server for database storage. If the list can be removed from, then each element will have an [X] by its element which sends a delete request to the server. The detail pane runs some logic on determining what to display, and the form required heavy re-formatting for proper sending to the server.

8 Dealing with Realistic Constraints

8.1 Parker

The wonderful, yet confusing nature of JavaScript is that it is asynchronous, and Node.js is non-blocking. This means that queries to the database, which we know have some latency greater than zero, present us with possible race conditions where code executes that uses the value to be returned from the database before it has actually been returned. To deal with this asynchronous aspect, we had to rely heavily on JavaScript?s callback functions. This worked, but once callbacks became nested within callback functions, things became a little difficult to debug.

8.2 Peter

Parker and I discovered that Mongoose does not have native deep population. We found a mongoose plugin, which supposedly gives us the ability to deep populate, but could not get it to work after much avail. This limited us is how we would be populating JSON objects and sending them to the front-end. We worked around this by populating the1 level that we could, and then manually populating the JSON objects to return by querying the database.

8.3 Brandon

Angular has a built in router called ng-router. However, this is lacking various functionality including the ability to create multiple views and nested views. As a way around this, there is a very popular UI-router library that many developers use that support these features. However, since it was not part of core Angular, we weren?t aware of its existence at the beginning of the project. As such we are currently managing our routing using selectors to hide and show the relevant views as people interact with our website. We would like to change this to using UI-router as soon as possible to allow for more flexibility and functionality

8.4 Wayne

While the calendar has the ability to populate modal pop-ups and link to pages, neither of these were useful in changing page details in a meaningful manner for use. I attempted to use JQuery to modify the buttons created by the calendar to run javascript functions to change the page, but the calendar modified its own HTML too often without running its callbacks to keep the buttons properly set. A few other minor designs needed to be changed as the AngularJS directives did not support the intended use. For the most part, the only impactful constraint came from the calendar design.

9 Teamwork and Team Member Interaction

9.1 Parker

Most of my interaction with the team was with Peter, the other person working on the back-end with me. However, as the due date for evolution approached, I made an effort to meet as an entire group to more efficiently and effectively solve problems that arose when connecting front end to back end. Over half of the time, the connection issue was a communication mishap that was solved very quickly.

9.2 Peter

Parker and I were partners on the back-end. We communicated very often and met daily to discuss our API, teach each other about Node, and pair program together. We kept in contact and communication with Brandon and Wayne through Facebook in order to coordinate meetings and to share updates and information. Parker and I also managed a design and API google doc for the front-end team. It made it very easy for the front-end team to know which routes were available and how to use them. We all had different working styles and habits. However, towards the end, we strove to emphasize meeting together as a group.

9.3 Brandon

I was partners with Wayne for the front-end. For the beginning of the project, we tended to meet briefly to discuss our plans and requirements and then spent the time implementing separately. As we approached the deadline, we met more frequently with Peter and Parker as well to make sure everything went smoothly when tieing the front-end and back-end together. This worked well for us mostly, but in the future I would like to meet more often in person as I work better when able to bounce ideas of others and discuss them outloud.

9.4 Wayne

Brandon and I worked on the client and worked mostly separately from Peter and Parker though we met up many times to work out details with the interface and any changes we needed to make. Aside from that, Facebook was the primary means of communication.