# **Neutrino**

# **Purpose**

The purpose of this document is to provide evidence for the different roles of the team members.

# **Introduction**

The objective of this project was to build a fully functional calendar management system that is controlled by a single user. The single user can have multiple calendars set up as per their requirement and manage them through a webpage. The operations that a user can do are create, delete and edit calendars, schedule and delete meetings with individuals.

The user can create and delete the calendars and add and remove additional days from the calendar. They can also close time slots as per their requirement. Either an entire date, a timeslot from a day in the week for every week, a timeslot from a date and a timeslot from all the available days.

The primary objective of this application was to develop an app that makes time management and scheduling convenient for users.

# **Team organization, members, and responsibilities**

|  |  |
| --- | --- |
| Frontend Group | Ashish Gurung |
| Backend Group | Ke Shao |
| Junfeng Guo |

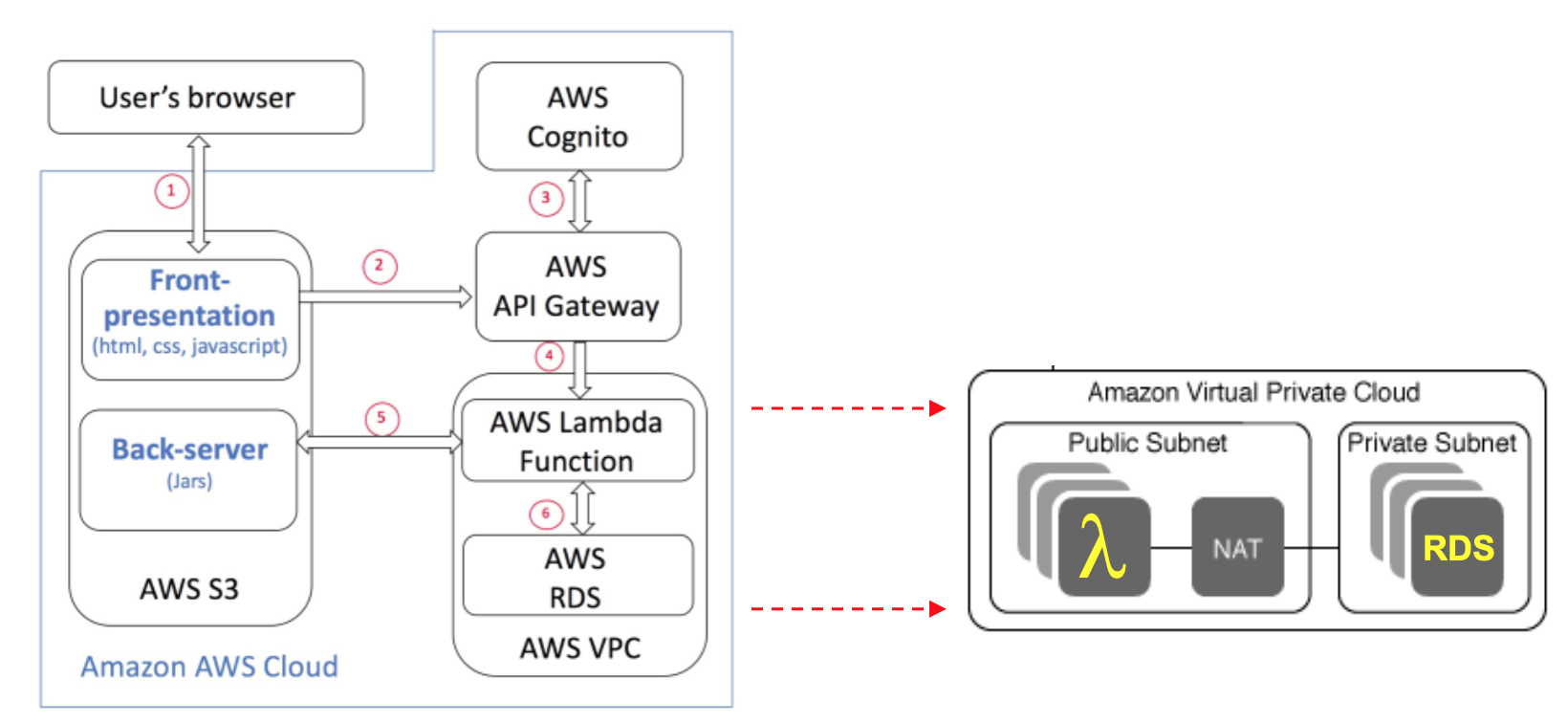
The team was divided into two major halves with two members working on the backend, one member working on designing the front end and testing the application. The testing ranged from UI testing to API testing. The backend project was hosted in a github page where as the designing and front end was individually done. The frontend is work entailed designing the UI and constructing the website using HTML, JS and CSS where as the backend work dealt with setting up the AWS lambda functions, database and the API that helps the frontend and backend interact with each other.

# **Process**

# This project was based off of the first assignment project that we had individually developed. We sat together and designed the database for the project. As the database design of Ke and Junfeng were similar to each others we decided to build our system on that.

The database consists of two tables, Calendars, and TimeSlots. Originally our model from the first project has some additional tables, Month, and Date, for easier searching when implementing the daily view and the monthly view. MySQL database gives us the ability handle the date variable easily using the date type, so we modified our initial design, abandon the Month and Date tables.

Our Calendars table has variable id, name, duration, start\_time, and end\_time. Id is assigned automatically by the database using auto increment. name is varchar and is marked as unique. It will ensure that when user try to create a new calendar with a exist name, it will not be inserted into the database. Duration is a integer representing how many minutes a meeting will last. Initially we think about using timestamp to represent start\_time and end\_time. However timestamp contains both date and time, which is very hard to maintain. So we decided to use int to store start\_time and end\_time, which represent how many minutes from 0:00am. TimeSlots table follows the same design principle. Id is integer assigned by the database automatically using auto increment. Status is a integer representing the status of the timeslot, which could be -1(closed), 0(open), and 1(scheduled) respectively. When the status of a time slot is open, the person and location should be an empty string.

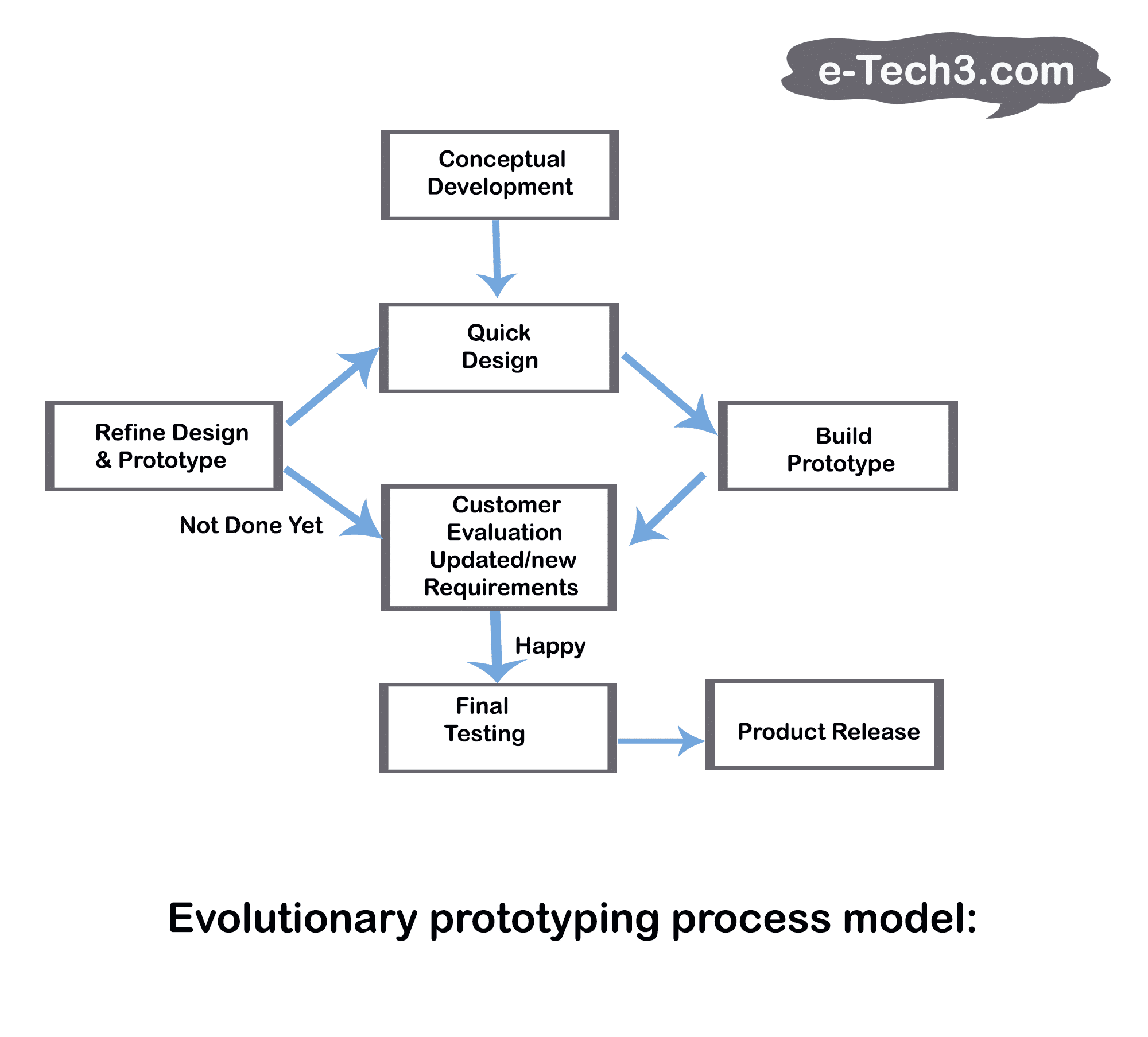


[figure : aws architechture breakdown of the project ]

Once the database design was completed we decided on how the front end and back end would interact with each other. We setup a list APIs with their URLs and described their functionality along with the expected inputs and outputs for the APIs. This API list was the base for our system as the frontend and backend team branched off into their respective tasks depending on the requirements of the application. We met twice a week, once every friday and once during the class meetings to brainstorm and get status updates on each other. These meetings were also an excellent opportunity to review our progress and give feedback on each others progress and find how everyone is progressing.

For the development of this application we found Prototype Development Model to be the most suited for our requirements. During our development we developed the application in stages. First we developed the create and delete functionalities and made a fully functioning prototype that met the expected requirements. We divided the tasks into five iterations which had the following goals at the end of each iteration:

1. Implement the create and delete calendar features
2. Implement schedule and delete meeting features
3. Implement edit calendar features: add and remove days
4. Implement edit calendar features: remove time slots
5. Final testing, UI polishing and thorough JUnit testing for all the lambda function



We did code reviews and testing at the end of each iteration to ensure that we could follow each others’ progress and ensure that the code is readable. We made 100 percentage test coverage that dealing on HandleRequest function for each lambda functions. However, the plan of running everything on a single html page and the end of term pressure meant that the code reviews was overlooked for the last two iterations which affected the overall readability of our code. Especially the javascript file and the JUnit test codes.

The primary digital communication communication medium used was Facebook messenger app as it allows us to set reminders and create events to keep track of our progress. We also maintained a separate Google doc file that kept track of our progress with respect to the use cases provided by our stakeholder(Prof. Heineman).

As we met in person at least twice a week and the relatively small size of our team meant that keeping track of each others progress was not a huge challenge so we didn’t use any extra medium apart from the messenger features and google doc.

# **Tools**

1. Eclipse:  
   We programmed the lambda functionalities for AWS using eclipse. One of the primary reason for selecting eclipse was the stability of an Eclipse IDE and the excellent support it receives from the community and Amazon for AWS related bugs.
2. Atom:  
   For coding the front end of the project we used Atom. It was just a matter of choice. Although it did complicate things when we had to debug the code and reupload it to S3.
3. MySql Workbench:

MySql Workbench gives us an easy and straightforward way to check the content in the database. It has the ability to create and modify tables in the database using GUI operation, and automatically generate Sql command accordingly, which is very helpful.

1. github:  
   As two people were primarily working on the back end of the project we decided to setup the java project for lambda functions, which allowed push/pull interaction between Eclipse and github.
2. Bitbucket:  
   We did give bitbucket a consideration as it is private and it also has additional features to help with project management. However given the size of our team and the project we decided not to explore bitbucket as it would only further add to the list of things to keep track, which would hinder our progress.
3. Slack:  
   Initially we decided to use slack to keep track of all our tasks, documentation and communications but we soon realized that we were getting easily distracted by it rich set of features and rather engaging interface. So we decided to simplify the whole situation and stick to Facebook messenger for communication and planning meetings and Google doc to track our progress.
4. Facebook Messenger:  
   We used messenger to communicate, set up meetings, events and reminders and create pools to help with design decision.
5. Google Document:  
   We used google document to keep track of the progress of the application and ensure that all the use cases were met.

# **Accomplishments**

The following table hold the list of primary requirements that we implemented on our application:

|  |  |
| --- | --- |
| **S.N.** | **Use cases implemented** |
| 1 | Create new calendar with:   * start and end date * starting and end hour * duration of each timeslot |
| 2 | Delete Calendar |
| 3 | Load Calendar |
| 4 | Add/Remove day to calendar |
| 5 | Remove timeslots:   * time slots from date * time slots from a day of the week * time slot at a given time * all time slots from a day |
| 6 | Schedule/Cancel meeting |
| 7 | Daily schedule |
| 8 | Monthly schedule |

The additional requirements that we explored was the dynamic loading and consumption of events using ajax to enhance the UX. We also tried to improve the UI to ensure a smooth user experience.

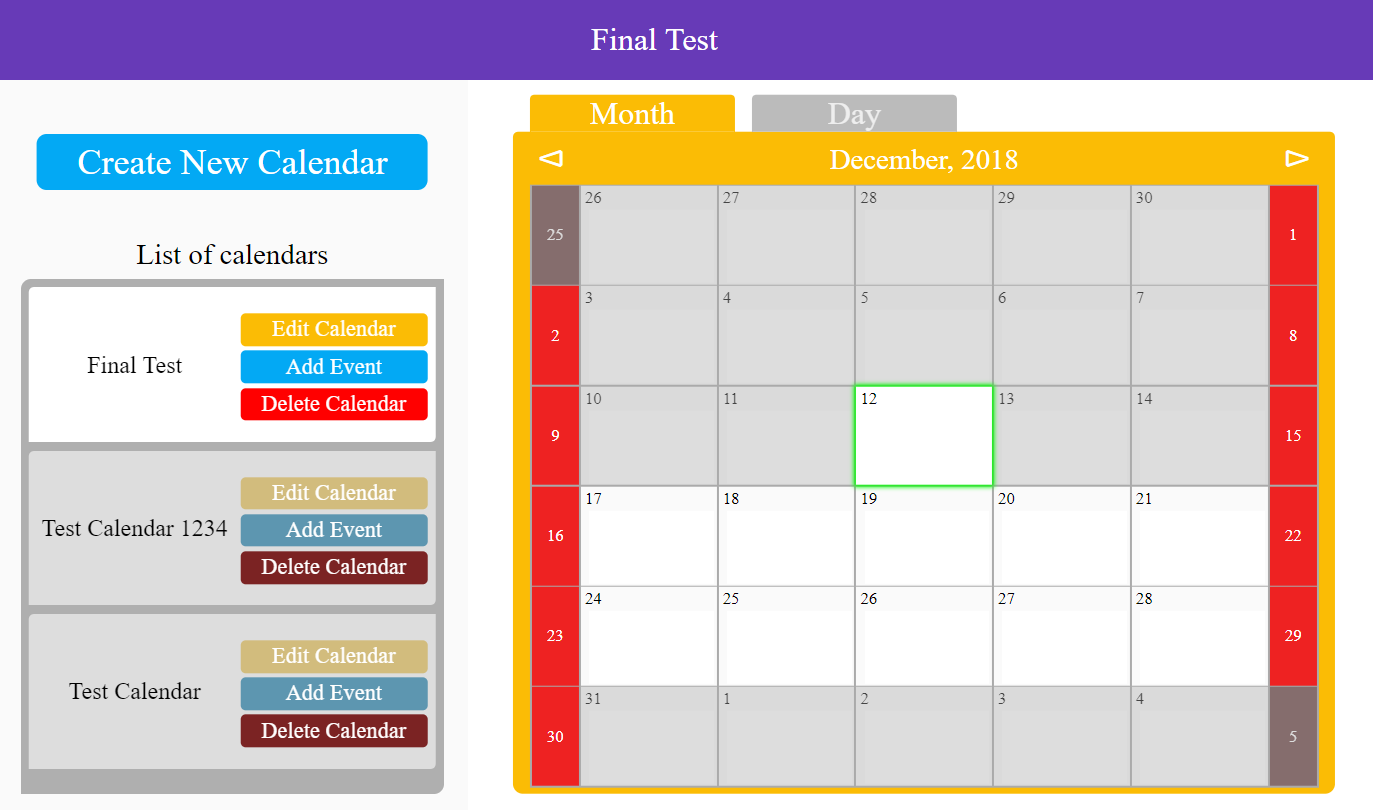


Fig: screenshot of the application with 3 calendars(Final Test is the active calendar), the monthly view with the 12th being the active date.

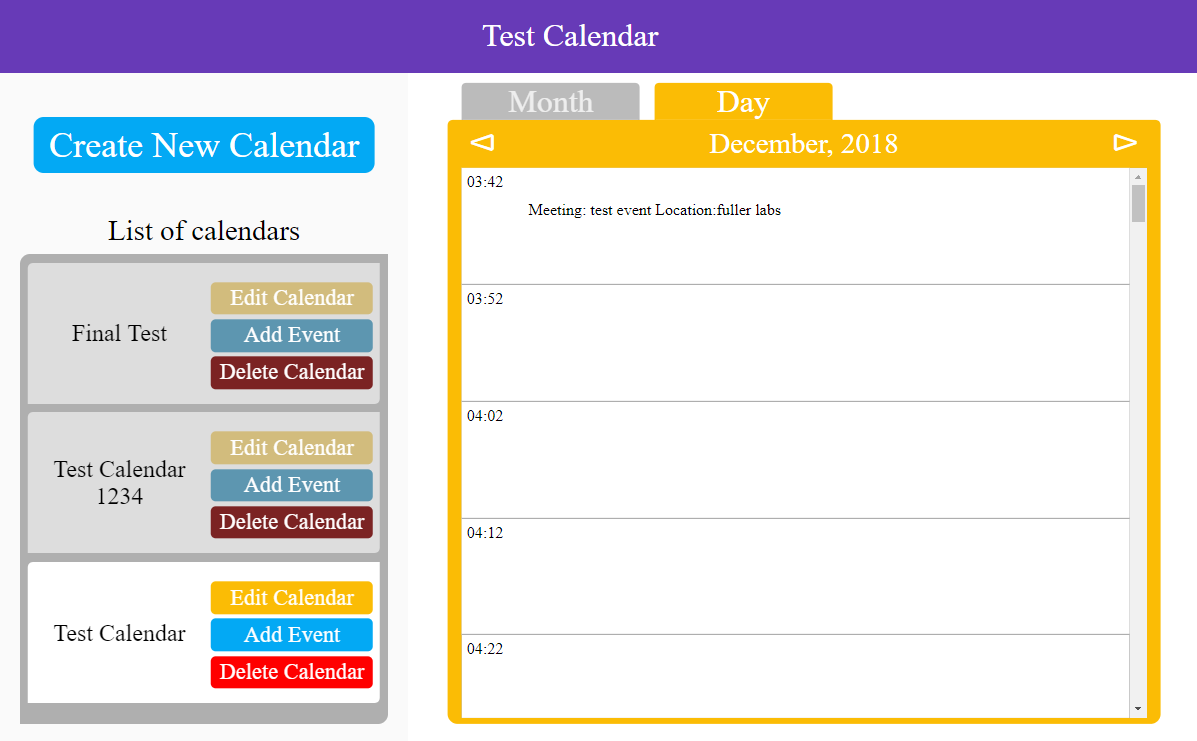


Fig: screenshot of the application with Test calendar as the active calendar and the daily view. There is a meeting “test event” in Fuller labs at 3:42 am.

We did try to explore the addition of push request and SMTP to further add to the features of the project. Although we wanted to implement the SMTP features we eventually decided to opt out of these features as we were running into too many configuration bugs and time was against us as the pressure of the finals was fast approaching.

## ***Deliverables***

For deliverables we only have the product and the introductory guide for it. The primary reason we didn’t write a user manual is our belief that we have designed the product well enough to ensure that the user can navigate through the application purely on intuition, if they know the requirements of the application.

# **Reflection**

## ***What worked, what didn’t work***

As is the case we ran into a few problems during the development and deployment of the product.

1. AWS   
   As none of us had worked with AWS before, we ran into some minor issue during the deployment. Especially with the CORS related issues. Which we resolved by opening up the API to all requesters, which was not the most intelligent thing to do security wise.
2. Minor issues during the design of JSON files  
   As JSON was the primary means of communication between the front end and the backend and we had set up the core architecture(urls for API and JSON messages for each API) very early on in the project. Having the opinion that the core did not need an in depth reevaluation and revision during the project was a mistake in judgement that we regretted when we approached the end of the project while during in depth testing and review of our code.
3. Separating the team into two halves  
   We felt that separating our team into two halves really helped our coding skills as it kept us in check while writing the code as we were more concerned with ensuring the readability of the code. It also helped us learn and develop our coding skills as we learned each other’s coding patterns and approaches.
4. github  
   Setting up github and managing .git files and .gitignore files was an interesting challenge during the development of our application as not all of were comfortable setting up these files so it was both a learning and teaching experience for fellow team members.

## ***Our biggest mistake***

Our biggest mistake was made during the initial phase of our project. Initially we had thought that meeting after class every week would suffice. However due to lack of proper communication and clarity we ran into minor issues where all of us were working on the project individually. The lack of cohesion within the team hindered our pace and we had to reevaluate our approach during the midterm break. We did find the flaws in our approach and changed our approach after mid-term and made sure that everyone was on the same page when it came to the project. It was an interesting learning point in our project as we all came to actually realize how having strong communication determines the success or failure of a project over the individual competence of team members. A case of “The whole is greater than the sum of its parts”. We concluded that the amount of time required to communicate a message across a team is completely dependant on the team and not the other way around.

## ***Changes we would make***

As a team we would suggest that there should have been a major milestone before the midterm break after the completion of either the second or third iteration; where each team would be expected to present their work in class. In our opinion, this would have made us more accountable and having one’s work reviewed by their peers is always a nice motivator to improve the quality of one’s work.

# **Lessons learned**

The major lesson we learned was in regards to team communication. Initially we just divided the work among us and went about working on the project individually. Although everyone in the team were equally competent, the progress we made during this time did not reflect the amount of effort we had put in. The primary lesson we learned during this project was the importance of communication and how the term “The whole is greater than the sum of its parts” always holds true in a team.

## ***Things we learned***

The most beneficial topic that we learned from this class was the development of application using AWS. As a sound knowledge of AWS is a very important skill to have in the professional sphere this class had the perk of being very challenging, up to date with the industry and an excellent opportunity to gain experience while mimicking actual real world team environment.

## ***Advice to future teams***

The advice we would have for future teams is to be proactive and do as much as you possibly can. Especially in CS509 the more you do the more you learn. So aim to make something significant and take the professor’s requirement of the final product as simply the baseline and not the final product of the application.