Milestone 3 Report (EinsteinBoys) - Gemini

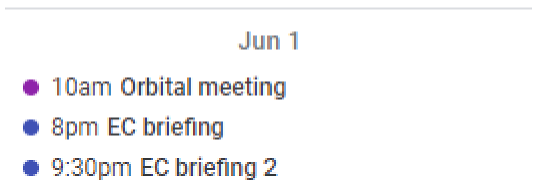
- Repo (with instructions) at https://github.com/dlimyh98/EinsteinBoys

Development process

Aim

To create a web application that enables users to intuitively visualise and plan their events and tasks in a single scheduling application

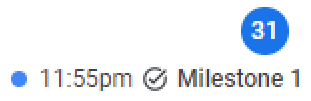
Existing scheduling apps do not make a clear distinction between events and tasks. This is especially problematic for tasks, because there is no way to visualise how long one has before the task's deadline. Take for example Google calendar:



Jun 1

- 10am Orbital meeting
- 8pm EC briefing
- 9:30pm EC briefing 2

Events are represented very clearly here. We know we have an Orbital meeting at 10am on June 1, and we can set reminders to alert us as the meeting draws closer



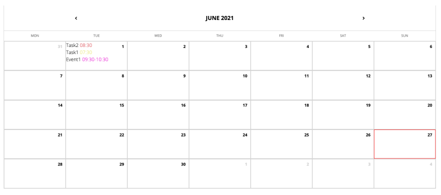
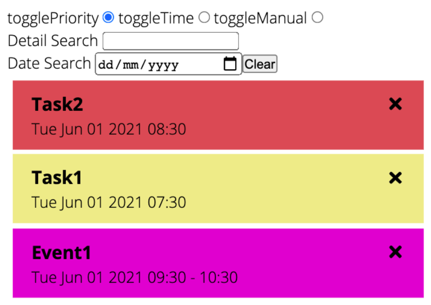
31

- 11:55pm Milestone 1

Unfortunately, the calendar entry to complete Milestone 1 by 1155 on May 31 is ambiguous. Firstly, there is no way to set reminders to alert of impending deadlines (the only notification you get is when the deadline has passed). Next, tasks look the same as events: there is no clear visual distinction between something that will happen for a set duration at a certain date and time (event) and something to be completed by a predetermined deadline (task). This makes it difficult to visualise how much time one has until the deadline.

User requirements

The application must have two distinct interfaces serving different purposes: the **Calendar** and the **Todo-list**. The table below shows the differences between the two types of interfaces:

	Calendar	**Todo-list**
Display	 ‹ JUNE 2021 › MON TUE WED THU FRI SAT SUN Task2 08:30 Task1 07:30 Event1 09:30-10:30	 togglePriority ● toggleTime ○ toggleManual ○ Detail Search Date Search dd/mm/yyyy Clear **Task2** ✕ Tue Jun 01 2021 08:30 **Task1** ✕ Tue Jun 01 2021 07:30 **Event1** ✕ Tue Jun 01 2021 09:30 - 10:30
Purpose	To serve as a continuous display for the user to have a macro view of their upcoming plans throughout the month	To aid users with prioritising upcoming plans by showing more important entries right at the top

Any entry that the user adds will be seamlessly shown on both the calendar and todo-list. Both interfaces are also displayed in the same page, allowing users to more easily visualise upcoming plans.

Calendar entries need to be separated into 2 categories: **Events** and **tasks**. The table below shows the differences between the two types of entries:

Events	**Tasks**
Description of entry *e.g. Vaccination*	Description of entry *e.g. Milestone 2 submission*
Start date and time, end date and time *e.g. Thu, 31 June 2021, 1700 - 1800*	Deadline *e.g. Mon, 28 June 2021, 2359*

As shown above, the key difference between **events** and **tasks** is their time entry and therefore how they are displayed on the calendar and todo-list.

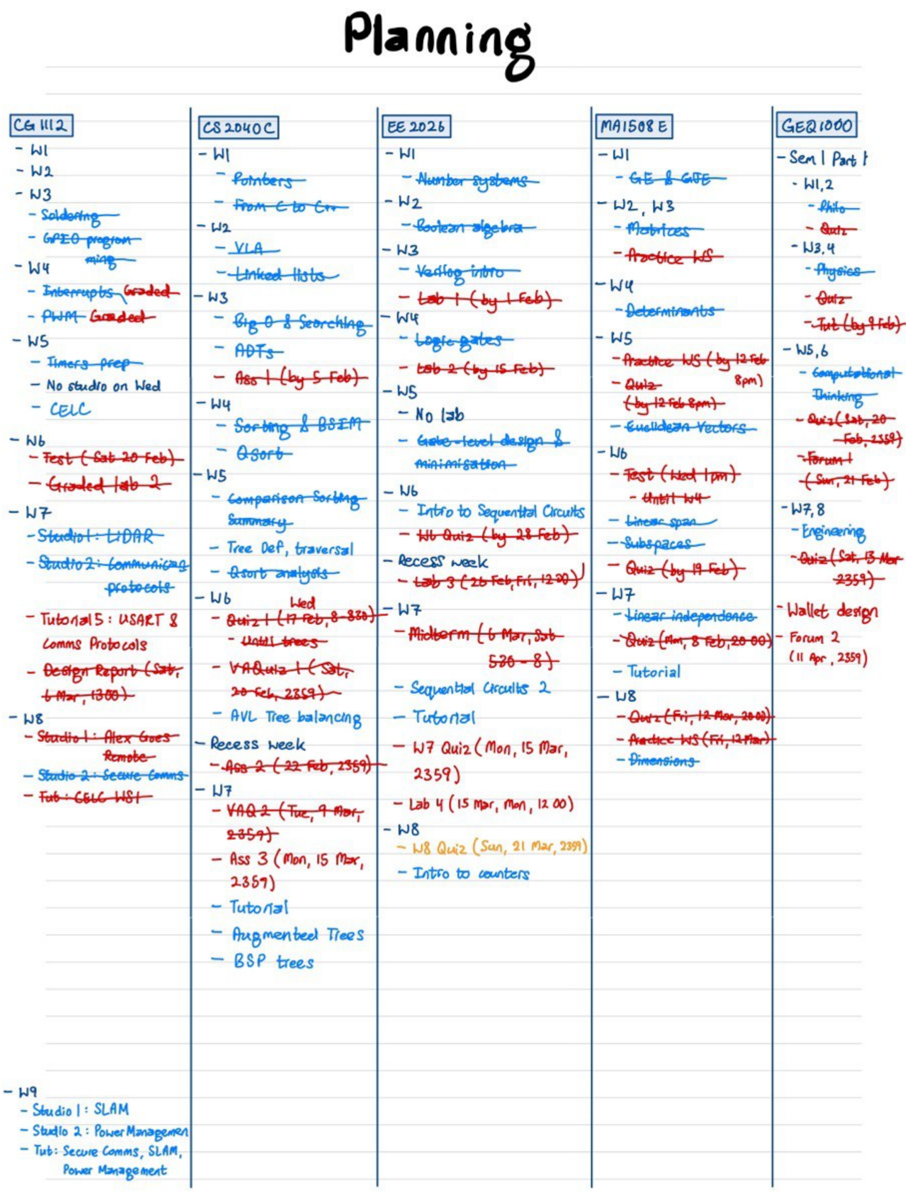
On top of displaying all the users' upcoming plans in an intuitive interface, our program must be able to send the user timely reminders to alert them of impending events and deadlines. We think the best way to implement this is through a Telegram bot. Considering how our target audience is mainly university students, Telegram is an appropriate platform to send users alerts because the majority of our user base would already be active on the app. This is in contrast to email alerts sent by other calendar applications, which users are less likely to check as often as Telegram notifications.

Through the Telegram bot, users can customise how often they want to receive reminders as upcoming plans draw closer. This allows users to receive timely alerts for when they need to make preparations for upcoming plans, for instance, leaving the house an hour earlier for a date or finishing their part for a group project the day before. It also ensures that notifications reach the user when they are most likely to be active and not asleep, so they are not bombarded by a whole string of reminders which they are then more likely to ignore, undermining the efficacy of our app.

Plan

Thought process

In order to come up with a product that would be effective in tackling the scheduling problem, we looked at how we have tried to solve this problem before in other ways. Personally for me, on top of Google calendar, which does not effectively display tasks in a way that shows the importance of said task, I list down all the assignments that I have to complete in chronological order. The below is an example of this approach:



Planning

CG1112
- W1
- W2
- W3
  - ~~Soldering~~
  - ~~GPIO programming~~
- W4
  - ~~Interrupts~~ ~~Graded~~
  - ~~PWM~~ ~~Graded~~
- W5
  - ~~Timers prep~~
  - No studio on Wed
  - CELC
- W6
  - ~~Test (Sat 20 Feb)~~
  - ~~Graded lab 2~~
- W7
  - ~~Studio1: LIDAR~~
  - ~~Studio2: Communicating protocols~~
  - Tutorial5: USART & Comms Protocols
  - ~~Design Report (Sat, 6 Mar, 1300)~~
- W8
  - ~~Studio1: Alex Goes Remote~~
  - ~~Studio 2: Secure Comms~~
  - ~~Tut: CELC WS1~~
- W9
  - Studio 1: SLAM
  - Studio 2: Power Managemen
  - Tut: Secure Comms, SLAM, Power Management

CS2040C
- W1
  - ~~Pointers~~
  - ~~From C to C++~~
- W2
  - ~~VLA~~
  - ~~Linked lists~~
- W3
  - ~~Big O & Searching~~
  - ~~ADTs~~
  - ~~Ass 1 (by 5 Feb)~~
- W4
  - ~~Sorting & BSEM~~
  - ~~Qsort~~
- W5
  - ~~Comparison Sorting Summary~~
  - Tree Def, traversal
  - ~~Qsort analysis~~
- W6
  - ~~Quiz 1 (17 Feb, 8-830)~~ Wed
    - ~~Until trees~~
  - ~~VAQuiz 1 (Sat, 20 Feb, 2359)~~
  - AVL Tree balancing
- Recess week
  - ~~Ass 2 (22 Feb, 2359)~~
- W7
  - ~~VAQ 2 (Tue, 9 Mar, 2359)~~
  - Ass 3 (Mon, 15 Mar, 2359)
  - Tutorial
  - Augmented Trees
  - BSP trees

EE2026
- W1
  - ~~Number systems~~
- W2
  - ~~Boolean algebra~~
- W3
  - ~~Verilog intro~~
  - ~~Lab 1 (by 1 Feb)~~
- W4
  - ~~Logic gates~~
  - ~~Lab 2 (by 15 Feb)~~
- W5
  - No lab
  - ~~Gate-level design & minimisation~~
- W6
  - Intro to Sequential Circuits
  - ~~W6 Quiz (by 28 Feb)~~
- Recess week
  - ~~Lab 3 (26 Feb, Fri, 1200)~~
- W7
  - ~~Midterm (6 Mar, Sat 530-8)~~
  - Sequential Circuits 2
  - Tutorial
  - W7 Quiz (Mon, 15 Mar, 2359)
  - Lab 4 (15 Mar, Mon, 12 00)
- W8
  - W8 Quiz (Sun, 21 Mar, 2359)
  - Intro to counters

MA1508E
- W1
  - ~~GE & GJE~~
- W2, W3
  - ~~Matrices~~
  - ~~Practice WS~~
- W4
  - ~~Determinants~~
- W5
  - ~~Practice WS (by 12 Feb 8pm)~~
  - ~~Quiz (by 12 Feb 8pm)~~
  - ~~Euclidean Vectors~~
- W6
  - ~~Test (Wed 1pm)~~
    - ~~Until W4~~
  - ~~Linear span~~
  - ~~Subspaces~~
  - ~~Quiz (by 19 Feb)~~
- W7
  - ~~Linear independence~~
  - ~~Quiz (Mon, 8 Feb, 20 00)~~
  - Tutorial
- W8
  - ~~Quiz (Fri, 12 Mar, 2000)~~
  - ~~Practice WS (Fri, 12 Mar)~~
  - ~~Dimensions~~

GEQ1000
- Sem 1 Part I
  - W1,2
    - ~~Philo~~
    - ~~Quiz~~
  - W3,4
    - ~~Physics~~
    - ~~Quiz~~
    - ~~Tut (by 9 Feb)~~
  - W5,6
    - ~~Computational Thinking~~
    - ~~Quiz (Sat, 20 Feb, 2359)~~
    - ~~Forum 1 (Sun, 21 Feb)~~
  - W7,8
    - Engineering
    - ~~Quiz (Sat, 13 Mar 2359)~~
- Wallet design
- Forum 2 (11 Apr, 2359)

As shown above, I separated school assignments by module and the week they have to be completed by. Each module takes up one column, and the workload for that module will be listed each week starting from Week 1 (W1).

For the first few weeks, this approach was very effective in helping me keep up with assignment deadlines as the sheet was still relatively empty and there were not many entries yet. However, as the semester progressed, the assignments for some mods were lower in priority than others. This could be because they had longer deadlines, or because there was not much preparation required for them. Unfortunately, the sheet was quickly inundated with entries and it was getting harder to see which assignments were higher in priority and had to be completed first. There were even some instances where I missed deadlines because the assignment entry was from a few weeks before and therefore was harder to spot in the list.

We realised that from this, our app needed to implement a way to prioritise tasks and events by displaying them in order of importance. We also decided that we would leave the user to define the importance of any entry and not solely base it off of its deadline because the importance of a task could also be determined by its workload.

From there, we decided on a few essential features that our app must have:

1) Clear distinction between **Events** and **Tasks**
2) A macro view of upcoming plans with the **Calendar** interface
3) A priority-focused display of events and tasks with the **Todo-list** interface
4) Customisable timely reminders with our **Telegram bot**

Approach

Now that we had a clear objective, we came up with a detailed plan to build the product. We decided to choose React as the environment to build our application in with advice from our advisor. Being computer engineering students, we had little to no knowledge of web development, so our priority was to get up to speed with the most effective web development techniques. Listed below is all the topics we had to learn on our own:

1) HTML
2) CSS
3) JavaScript
4) Git
5) MongoDB
6) React

We spent the first few weeks self-learning the above topics. However, a few problems soon arose. Closer to Milestone 1, it was clear that we were both learning at different rates and we had to delegate the workload in other ways. Another was that we realised that some features could not be implemented or were difficult to implement, and we had to forgo such features as they were not essential to solving the overarching issue our application aims to solve.

User stories

To gain a deeper insight into the effectiveness of our application as well as how we can improve, we got family members and friends to try our application for a week and thereafter interviewed them. Here are the more notable insights that we gathered:

1) Emir Ilyas, 20, NSF
    a) As an officer in the army, Emir has to juggle between meeting mission planning deadlines and minimising clashes between his work commitments and personal life. He especially likes how both the calendar and todo-list are shown on the same page. This makes it easy for him to prioritise more important tasks. This is because an important deadline might seem to be far away, but require days of preparation. This is where the priority listing feature of the todo-list comes in handy. He cites an example where he was planning to catch a movie with his friends, but upon checking his todo-list, he realised that he had a company physical training exercise to plan for due in four days. In the calendar, it seemed like he had plenty of time to complete the task. However, since it was at the top of his todo-list, he realised he had to start planning days before. As a result, he decided to forgo his plans with his friends to complete the task. Fortunately, because he prepared for the task well in advance, he completed it one day before the deadline and was then able to make up for his missed plans with his friends.
    b) However he laments that the UI could be messy at times and he would like to have an option to toggle between the calendar and todo-list whenever he only wants to focus on one
2) June Lee, 20, university undergraduate
    a) Considering how this trial was conducted during her summer break, June has not had the opportunity to use this app when her schedule is at its busiest. However, during the one week she has used the app during her break, she has seen how it can potentially be useful when her school term resumes because she still has hall orientation commitments on top of her Orbital workload.
    b) Although not yet implemented, she is looking forward to the integration of the Telegram bot notification function that will eventually be incorporated into our app. She thinks it will be especially useful because she uses Telegram for messaging very often and will usually check the app several times a day. This will ensure that she will not accidentally miss any alert, and therefore, assist her in starting on assignments on time well before their deadlines.
3) Irmawati Jantan, 51, teacher
    a) As a mother of 3 and a teacher, Irma found the clear distinction between tasks and events very helpful in managing her day-to-day activities in attending to her family while teaching online. An example that she cited happened last week, when a distant relative came to visit. In preparation for the visit, she wanted to cook up something special which had to be prepared days in advance. For that, she added a task entry to remind her to purchase the required ingredients closer to the date, and another calendar entry to remind her when the relative would be visiting and for how long.

Tech Stack / Libraries Used / Database

Tech Stack

- MongoDB
    - MongoDB Atlas supports Cloud Storage, and is relatively easy to deploy. It also has a document-based data model which is attractive for the purposes for App, as it allows for fast retrievals.

- Express.js & Node.js
    - The norm for coding Backend/Server logic seems to be this, and a significant amount of documentation already exists for it.

- React
    - An easier learning curve compared to Angular/Vue, and the unidirectional data-flow keeps things easier for our first foray into programming Web Applications. Existing documentation is rich and diverse as well.

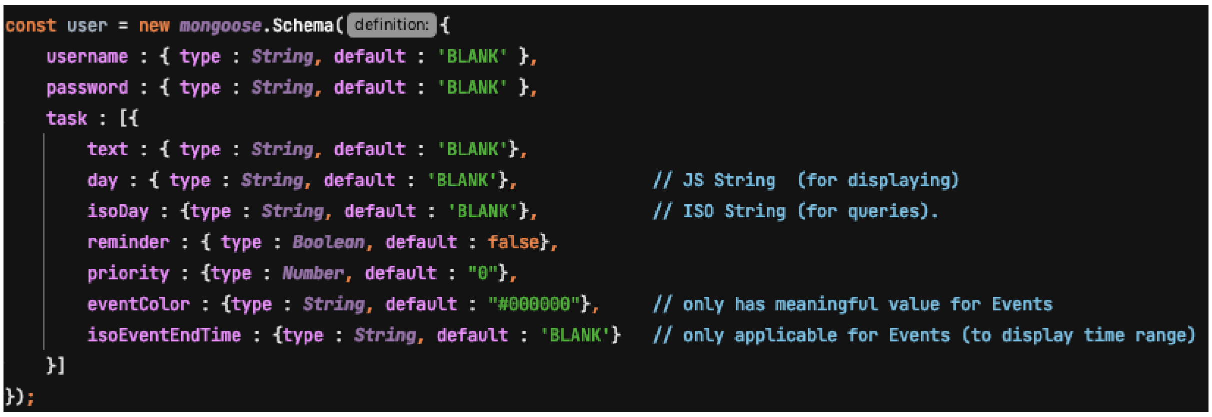
Libraries and Framework

- Passport.js
    - Has powerful Authentication features that can support what our App needs. It is flexible and modular as well, making it easy to add as middleware to Express.

- Moment.js & Date-fns.js
    - Both have powerful features which allow us to query and manipulate data relating to Date/Time, without the headache that comes with it. They complement each other well, as each library may have some functions that the other does not.

- Axios
    - Helps us to implement Promised-based asynchronous Javascript HTTP requests. Integrates with Node.

Database Organisation



```
const user = new mongoose.Schema( definition: {
    username : { type : String, default : 'BLANK' },
    password : { type : String, default : 'BLANK' },
    task : [{
        text : { type : String, default : 'BLANK'},
        day : { type : String, default : 'BLANK'},             // JS String  (for displaying)
        isoDay : {type : String, default : 'BLANK'},           // ISO String (for queries).
        reminder : { type : Boolean, default : false},
        priority : {type : Number, default : "0"},
        eventColor : {type : String, default : "#000000"},     // only has meaningful value for Events
        isoEventEndTime : {type : String, default : 'BLANK'}  // only applicable for Events (to display time range)
    }]
});
```

Mongoose was chosen, as it's Object Data Modelling library made it easier to work with MongoDB.

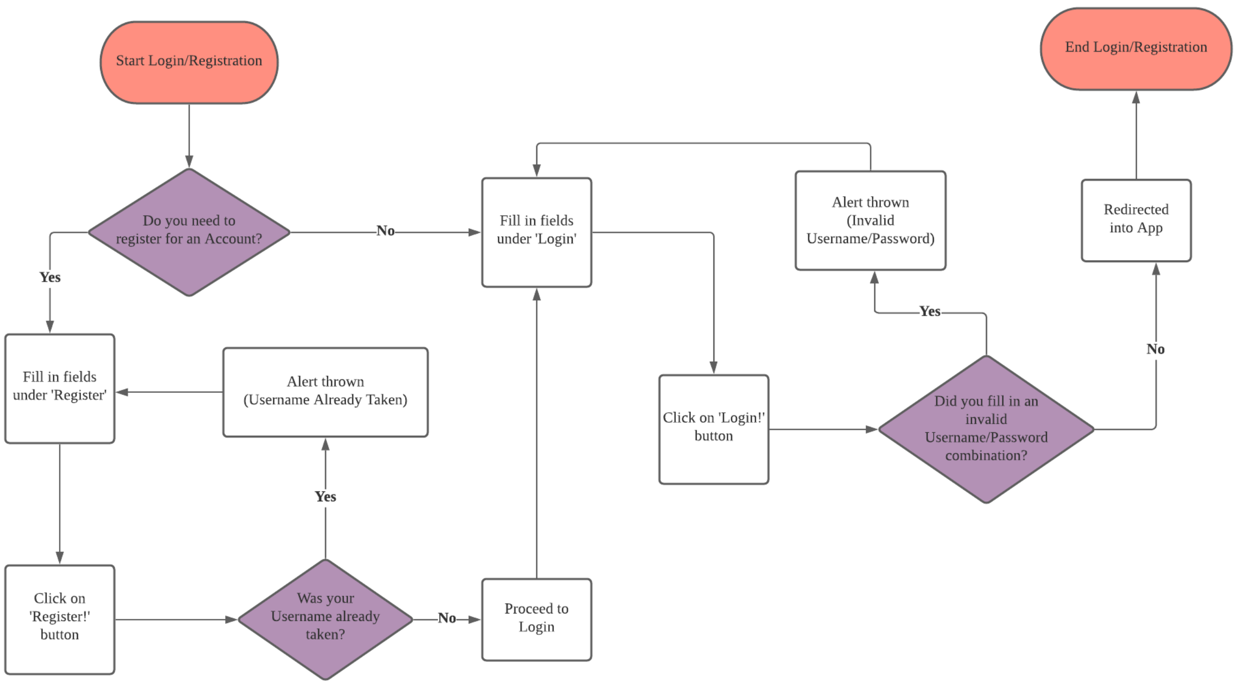
Each User has the following fields attached to it

- Username (for Authentication purposes)
- Password (for Authentication purposes)
- Task Array (each index contains information on a **specific** task)
  - Text
    - Details about Task/Event
  - Day
    - Start date/time of Event, or deadline of Task. Used for UI display.
  - isoDay
    - Start date/time of Event, or deadline of task. Saved in ISO8601 format, used for querying/manipulating/comparing.
  - Reminder
    - Represents whether Reminders are enabled or not
  - Priority
    - 0-3 scale representing the priority of a Task/Event.
  - eventColor
    - Represents the User selected color for Event
  - isoEventEndTime
    - End date/time of Event, saved in ISO8601 format. Used for querying/manipulating/comparing.

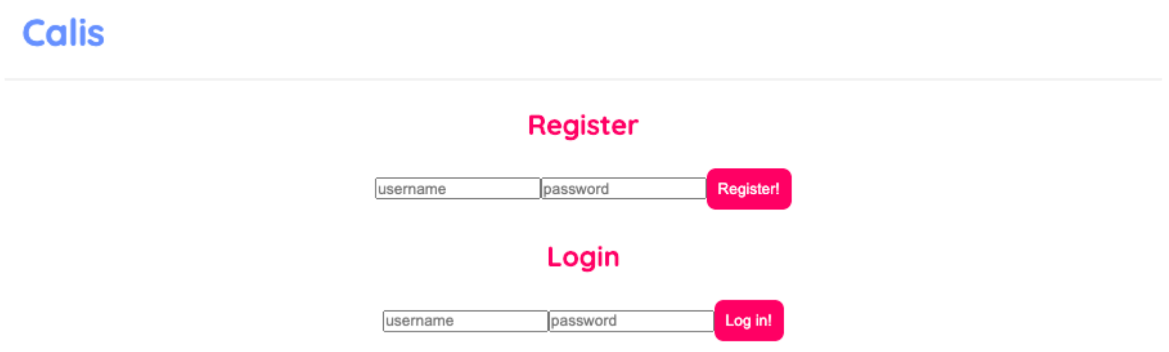
User flow, Design , Implementation of Features

Registration & Login (User Flow)

Registration and Login is an important core feature of our App, as it allows for Users to save their information on our databases.

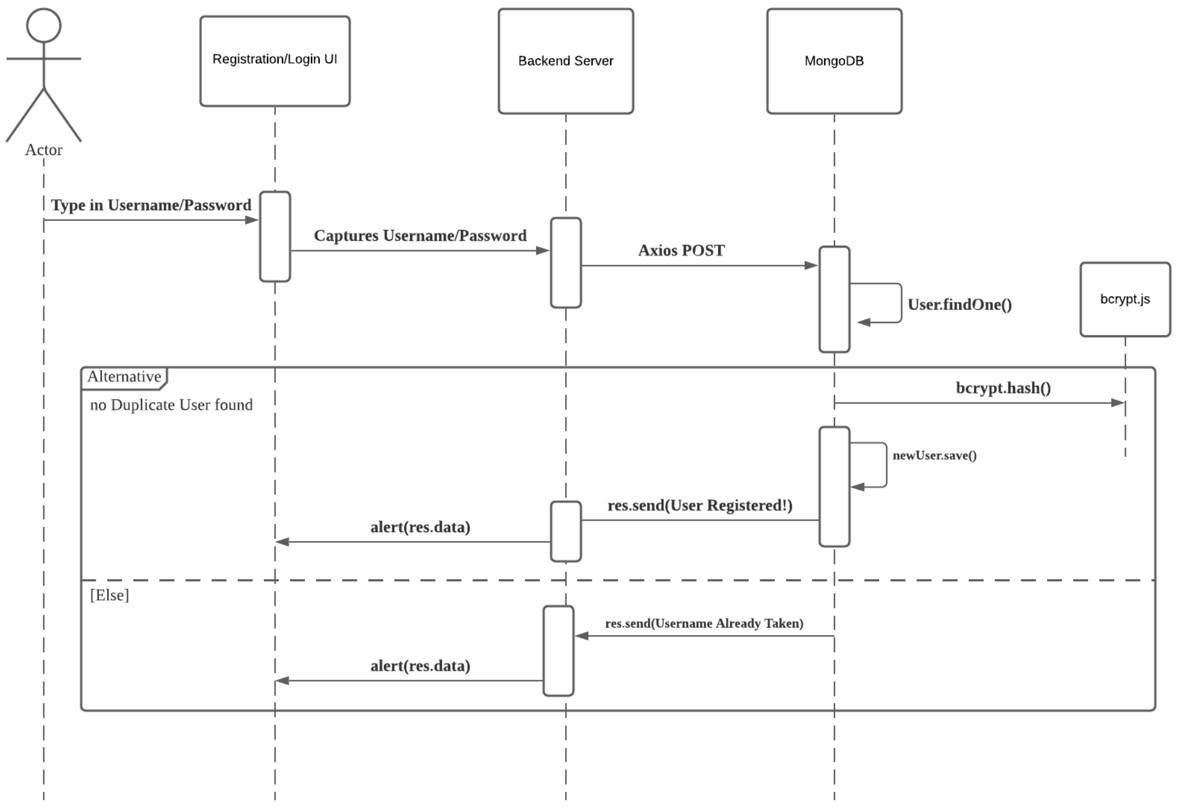

Start Login/Registration
Do you need to register for an Account?
Yes
No
Fill in fields under 'Register'
Alert thrown (Username Already Taken)
Click on 'Register!' button
Was your Username already taken?
Yes
No
Proceed to Login
Fill in fields under 'Login'
Click on 'Login!' button
Did you fill in an invalid Username/Password combination?
Yes
No
Alert thrown (Invalid Username/Password)
Redirected into App
End Login/Registration

Registration & Login (Design)



A more modern ‘Quicksand’ font was utilized, in order to appeal to our younger target audience. Taking feedback from the other Groups, the Password field will also not be shown as plaintext.

Registration (Implementation)



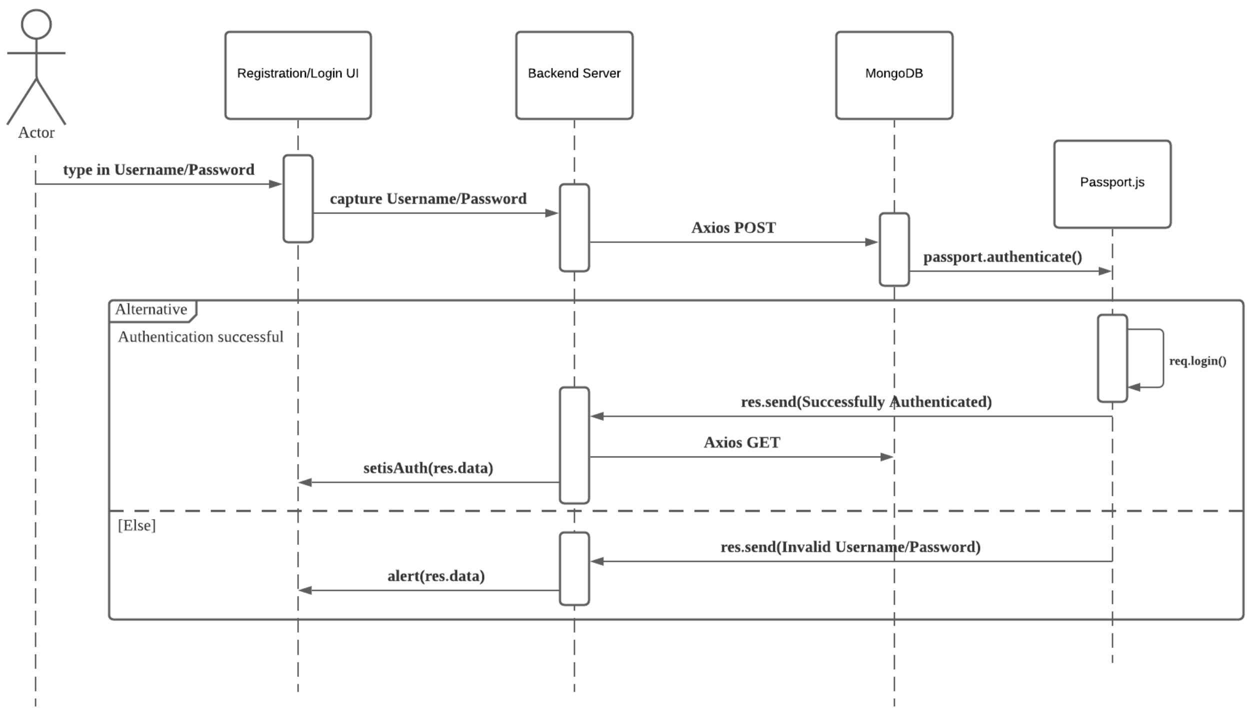
Registration UI captures the Username/Password as the User fills in those fields. It passes this to the Backend Server, which does an Axios POST request to MongoDB, carrying the Username/Password as data.

MongoDB will search through it's database for any existing User with a similar Username (*User.findOne()*).

If a duplicate User is found, MongoDB sends back a response detailing that the Username is already taken. This message is shown as an alert on the UI.

If no duplicate User is found, MongoDB first hashes the password using bcrypt.js. After which, the Username and hashed Password is saved into the database (*newUser.save()*), and a response is sent back detailing that the Registration was successful. This message is then shown as an alert on the UI.

Login (Implementation)



Login UI will capture the Username/Password as the User fills in those fields. It passes this to the Backend Server, which does an Axios POST request to MongoDB, carrying the Username/Password as data.

MongoDB will run Authentication using passport.js (*passport.authenticate()*).

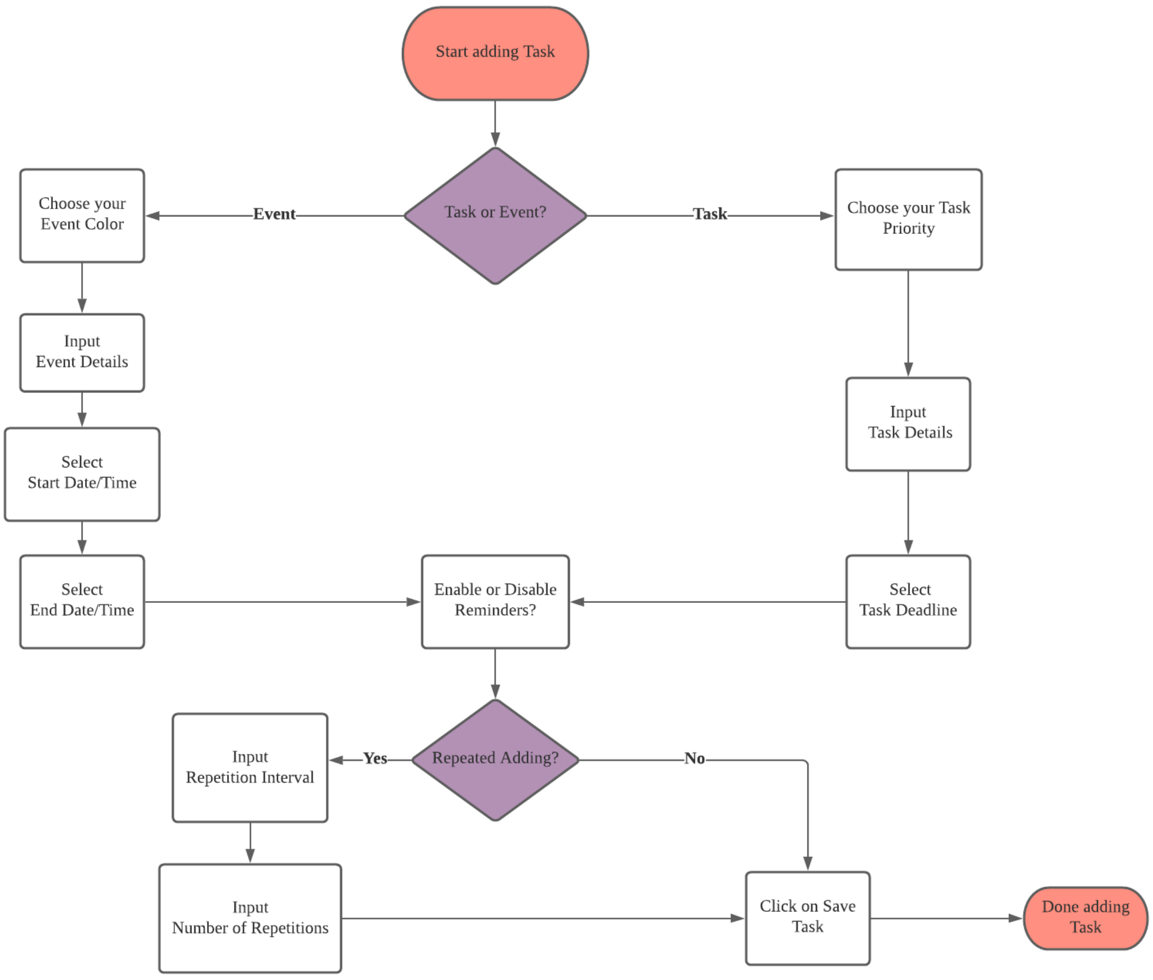
If authentication is successful, MongoDB will proceed to establish a Login session using passport.js as well. (*req.login*).

If the login session is established, MongoDB will send a response to the Backend Server with ‘Successfully Authenticated’, triggering the Backend Server to make a Axios GET request to MongoDB. This fetches the specific User’s data from the database, in preparation for entry to the App. After which, the isAuth state is set to true, which allows the User to be redirected into the App.

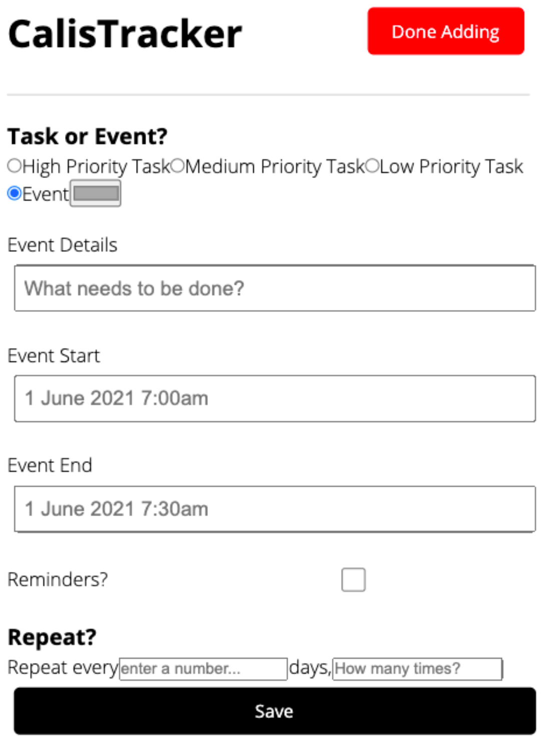
If the authentication is not successful, MongoDB will send a response to the Backend Server with ‘Invalid Username/Password’. This will be shown as an alert on the UI.

AddList (User Flow)

AddList is a core feature which allows Users to add Tasks/Events to their To-Do List, through a UI that guides them through several options.



AddList (Design)


CalisTracker

Done Adding

Task or Event?

High Priority Task Medium Priority Task Low Priority Task [✓ Event]

Event Details

What needs to be done?

Event Start

1 June 2021 7:00am

Event End

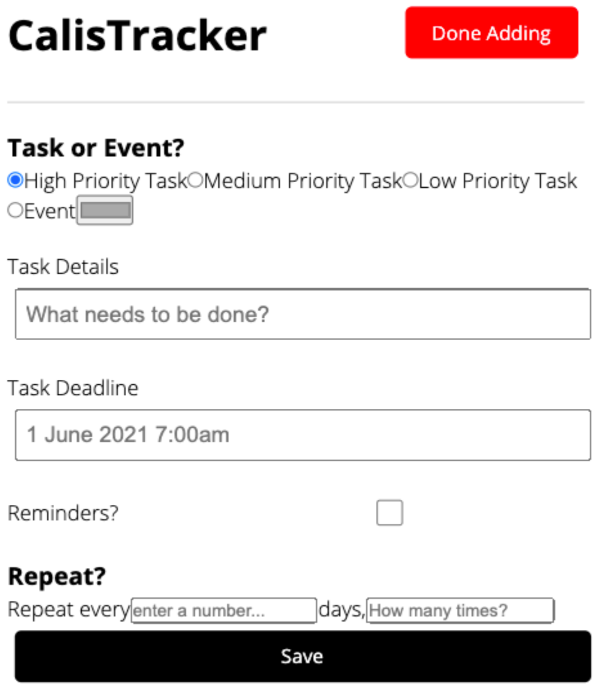
1 June 2021 7:30am

Reminders?

Repeat?

Repeat every enter a number... days, How many times?

Save


CalisTracker

Done Adding

Task or Event?

[✓ High Priority Task] Medium Priority Task Low Priority Task Event

Task Details

What needs to be done?

Task Deadline

1 June 2021 7:00am

Reminders?

Repeat?

Repeat every enter a number... days, How many times?

Save

The fields are arranged to guide the User, with the most important fields being placed higher up.

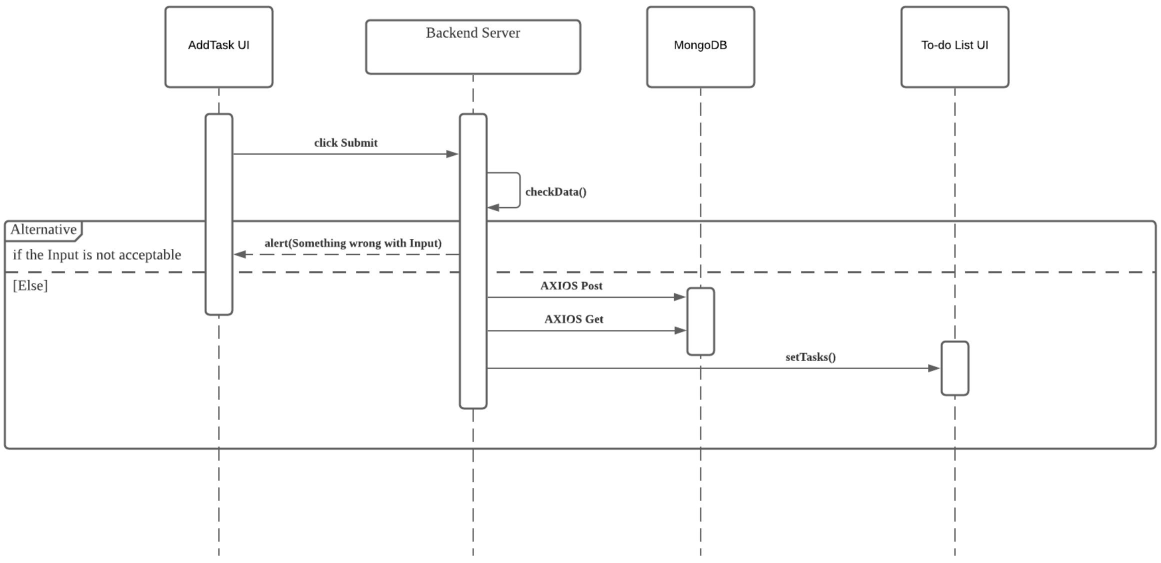
Firstly, *'Task or Event?'* radio buttons will allow the User to establish which one they intend to input into the To-Do List. Since the choice of Task/Event will influence how the rest of the input fields look, it is important these radio buttons are placed highest up, to improve the User Experience.

Secondly, Details and Timings of the occasion will naturally follow after the User has decided on the Task/Event. Hence, input fields are placed directly after to capture this.

Thirdly, the User should be aware of our (currently unimplemented) Telegram Bot-Reminder feature. Hence, it is placed third while we still have the User's attention.

Lastly, we have the *'Repeat?'* field, which allows the User to easily add recurring Task/Events. This is an important feature, hence we did not want to 'absorb' it into one of the above fields.

AddList (Implementation)



When the User has filled in the form and clicks ‘Save’, the UI will register a ‘Submit’ action to the Backend Server.

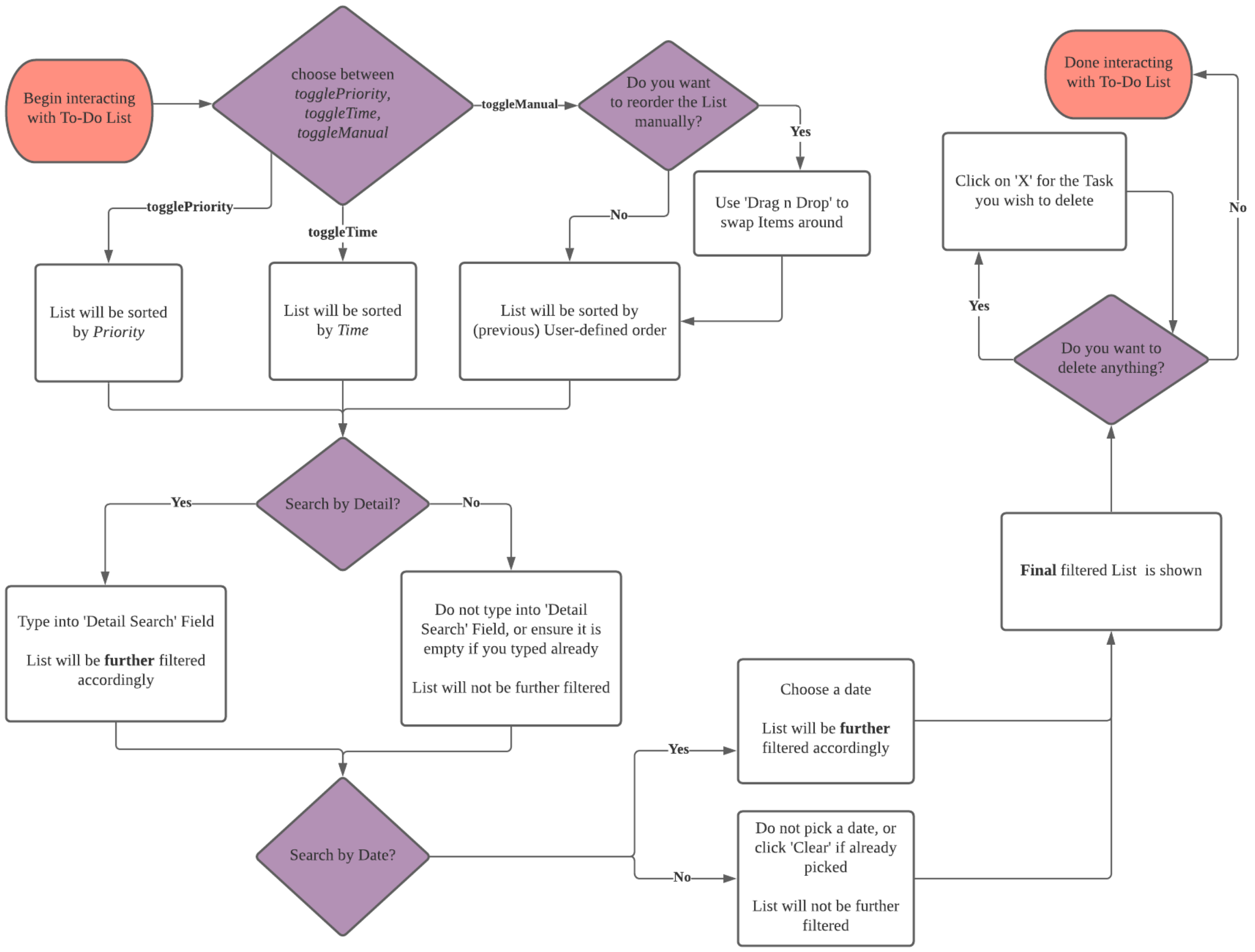
The Backend Server will run a *checkData()* function, which parses the form’s fields and checks for errors. An example of error-checking would be using moment.js and date-fns libraries to determine if the Date/Time input is valid.

If the input is not acceptable, an alert is immediately returned to the UI.

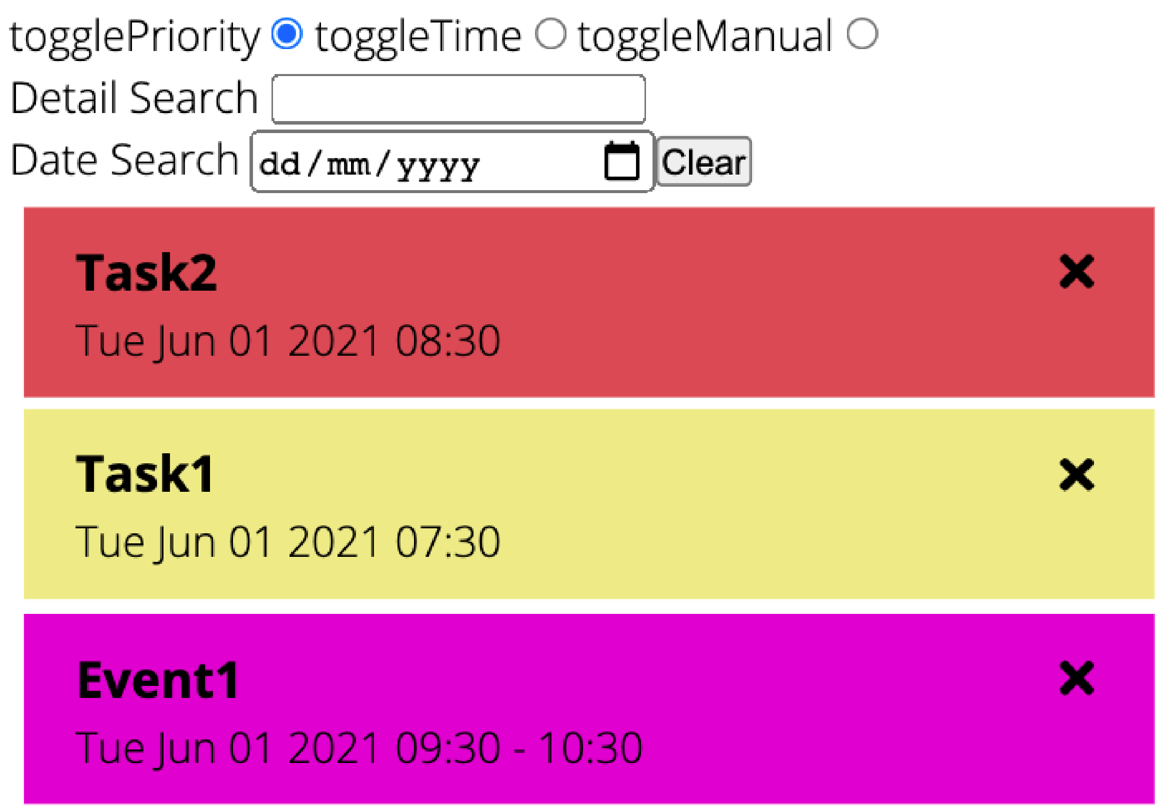
If the input is acceptable, Backend Server will make an Axios POST request to MongoDB, updating the database with the just added Task/Event. After which, an Axios GET request is made to fetch the **entire** list of Tasks/Events from the database. Finally, we run a *setState()* function, passing in this recently fetched list of Tasks/Events as a parameter. This will cause the frontend To-do List UI to re-render, and display the new list of Tasks/Events!

To-Do List (User Flow)

To-Do List is a core feature which allows the User to visualize/modify all of the Task/Events they have, with the aid of several options.


Begin interacting with To-Do List
choose between *togglePriority*, *toggleTime*, *toggleManual*
toggleManual
Do you want to reorder the List manually?
Yes
Use 'Drag n Drop' to swap Items around
No
togglePriority
toggleTime
List will be sorted by *Priority*
List will be sorted by *Time*
List will be sorted by (previous) User-defined order
Search by Detail?
Yes
No
Type into 'Detail Search' Field
List will be **further** filtered accordingly
Do not type into 'Detail Search' Field, or ensure it is empty if you typed already
List will not be further filtered
Search by Date?
Yes
No
Choose a date
List will be **further** filtered accordingly
Do not pick a date, or click 'Clear' if already picked
List will not be further filtered
Final filtered List is shown
Do you want to delete anything?
Yes
Click on 'X' for the Task you wish to delete
No
Done interacting with To-Do List

To-Do List (Design)



Task2
Tue Jun 01 2021 08:30

Task1
Tue Jun 01 2021 07:30

Event1
Tue Jun 01 2021 09:30 - 10:30

The choices are laid out with the most important ones at the top.

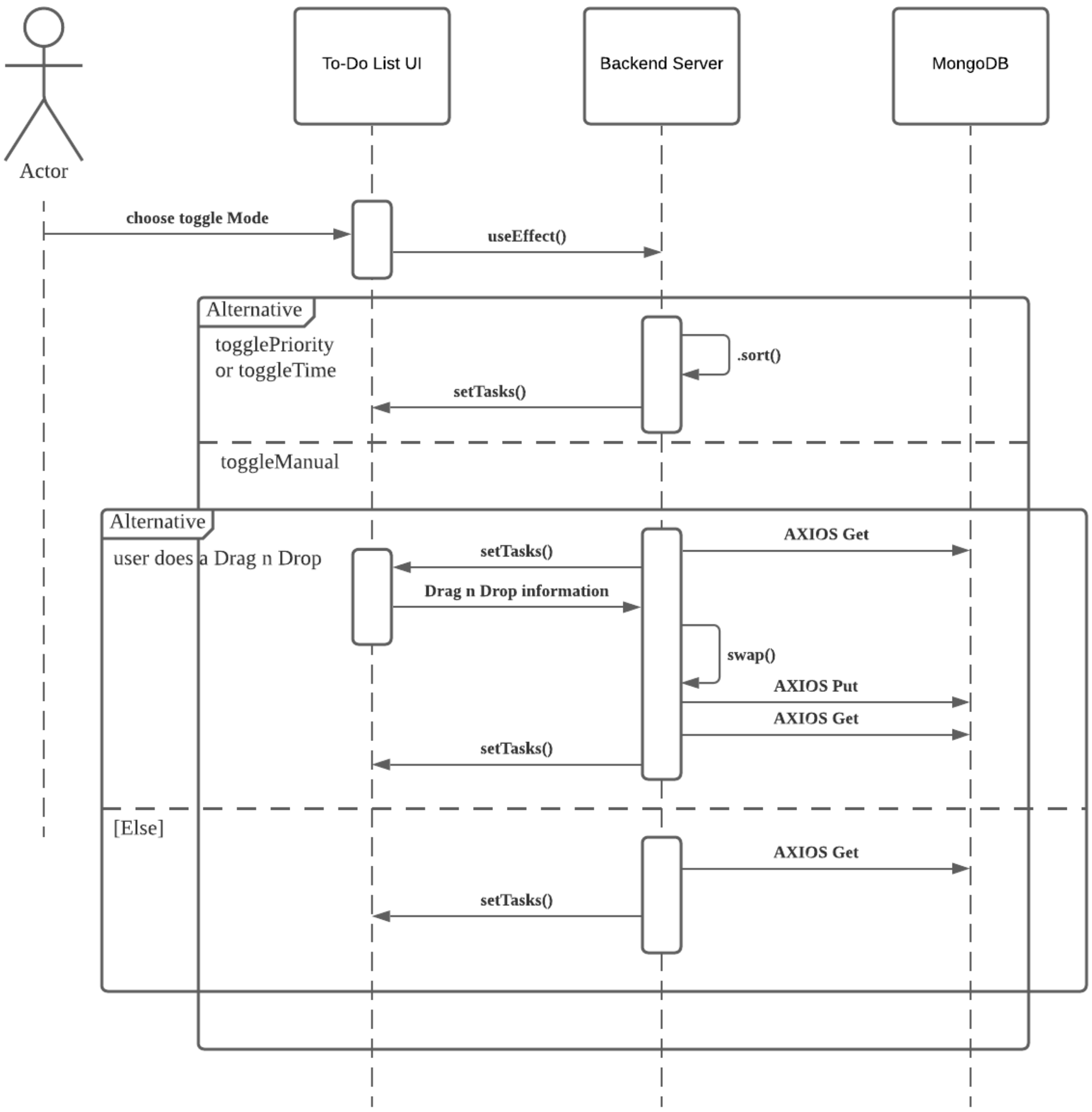
Highest up, we have the ‘toggle’ radio buttons. Since a key feature of our App is the distinction between Priority and Time, as well as having the ability to interactively reorder your To-do List, the ability to control these should rightfully be at the top.

Following that, we have the ‘Detail Search’ bar. It is located above the ‘Date Search’ selector, as it seems more natural for people to first think about ‘what’ they have to do, then think about ‘when’ they have to do it by.

Lastly, ‘Date Search’ is represented by a dedicated selector, instead of a text field that accepts a string. This is because the selector allows the User to view dates in a Calendar grid format, which enhances User Experience.

To-Do List (Implementation)

toggleMode



Users decide between togglePriority, toggleTime, toggleManual radio buttons, which will accordingly reorder the List.

Each radio button is a dependency of a *useEffect()* function, which will trigger when it's respective radio button is selected.

If togglePriority or toggleTime is chosen, the appropriate *.sort()* function is run on a **copied** List (**not** the original List). After which, a *setState()* function is used to update the To-Do List UI. It is important to note that togglePriority and toggleTime **do not** modify the database, all the modifications are done locally.

If toggleManual is chosen, there can be two further scenarios.

Regardless of which scenario, we always start with an Axios GET request to fetch the latest List from MongoDB. Then, a *setState()* function is used to re-render the To-Do List UI.

If the User does not proceed to do a *‘Drag n Drop’* operation, it stops here. The User will be able to view his manually-sorted List.

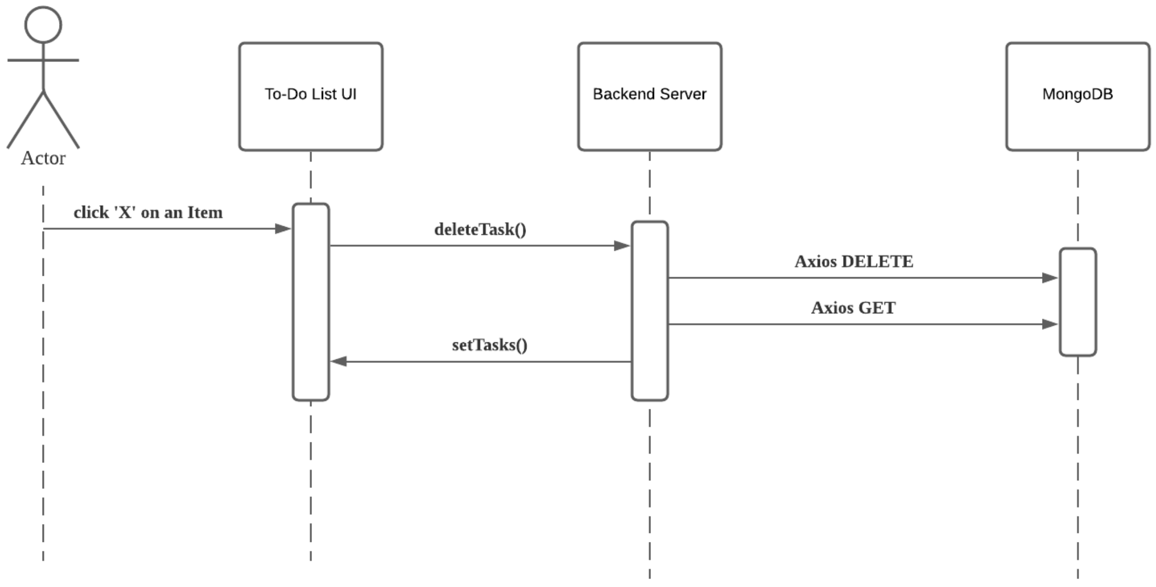
If the User chooses to do a *‘Drag n Drop’* operation, the To-do List UI will first capture this *‘Drag n Drop’* information (using Events). After which, a *swap()* function is run to swap the appropriate information between the two items chosen. Finally, an Axios POST is made to push this latest List to MongoDB, then an Axios GET is made to fetch this List from MongoDB, finishing off with a *setState()* to update the To-Do List UI.

Search by Detail / Search by Date

Users can choose to employ these two options, which allows them to further filter their List.

This is done through conditional rendering, whereby items that do not match the filter inputs are ‘hidden’.

Deletion



Users can opt to delete items from their List.

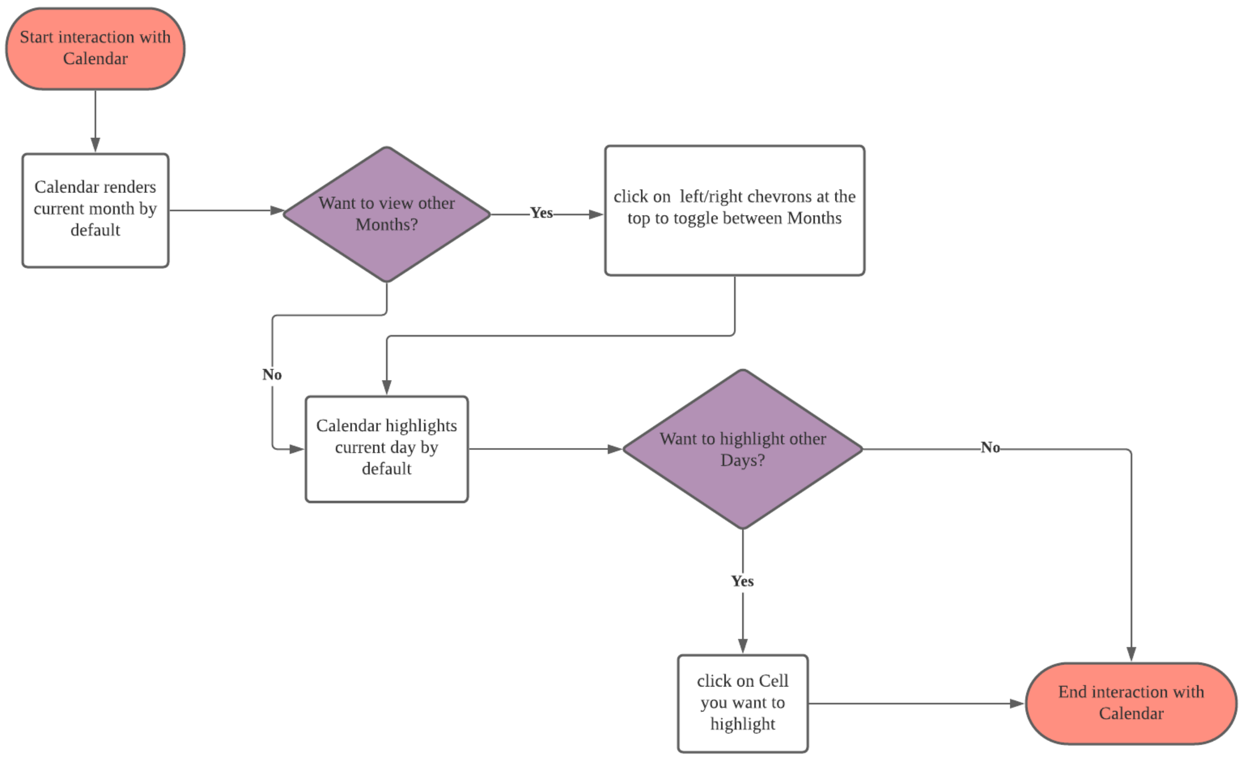
When the ‘X’ is clicked on any item, the UI captures the tagged item and triggers a *deleteTask()* function at Backend Server, with the tagged item as a parameter.

Backend Server runs Axios DELETE to MongoDB, deleting that **specific** task only. Once done, Axios GET is used to fetch the **entire** list from MongoDB, and then a *setState()* is run to update and re render the To-Do List UI

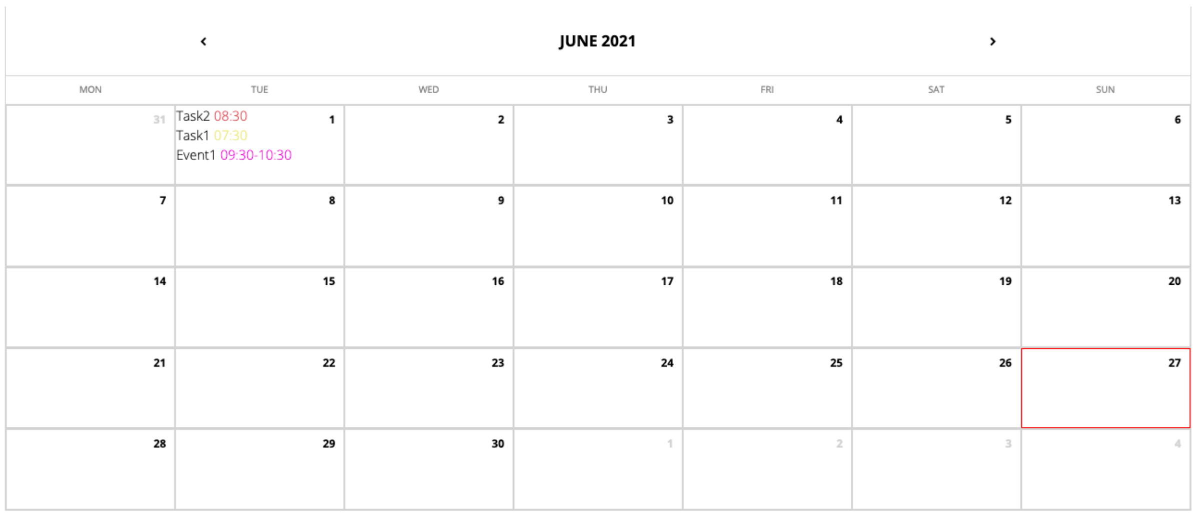
Calendar (User Flow)

The Calendar is a core feature, which allows the User to visualize their Tasks/Events in the context of a Calendar-like grid.

It has less interactive options than the other core features, however it still plays an important part in the User Experience.


Start interaction with Calendar
Calendar renders current month by default
Want to view other Months?
Yes
click on left/right chevrons at the top to toggle between Months
No
Calendar highlights current day by default
Want to highlight other Days?
No
Yes
click on Cell you want to highlight
End interaction with Calendar

Calendar (Design)



‹ JUNE 2021 ›

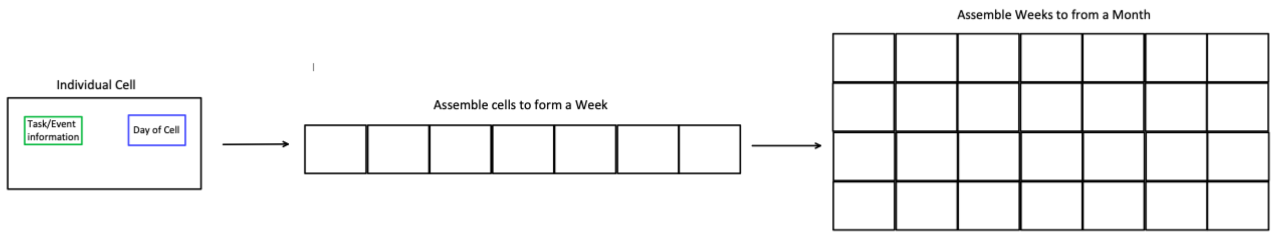
MON	TUE	WED	THU	FRI	SAT	SUN
31	1 Task2 08:30 Task1 07:30 Event1 09:30-10:30	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	1	2	3	4

The coloring of the Calendar should not distract away from other color-based features in the App (e.g. color-based Task/Event coding), hence a neutral Black/White color-scheme was chosen as the base.

A Monday-Sunday structure was also chosen, as opposed to the more common Sunday-Saturday structure. This is to provide easier separation between Weekdays/Weekends.

Calendar (Implementation)

Calendar was built using a Bottoms-Up approach.



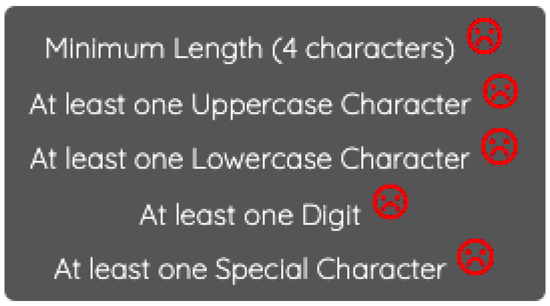
Starting off, a cell is built with its corresponding Task/Event information, and the date of the cell. This represents a Day.

Next, 7 of these Days are pushed into an Array, which is wrapped around a <div>. This <div> forms a Week.

Finally, 4 Weeks are pushed into another Array, which itself is wrapped around a <div>. This <div> forms a Month.

Password Checker

When registering, Users must input a password that is of acceptable quality. In order to aid this process, a hover-over box was added to remind Users of the criteria needed.



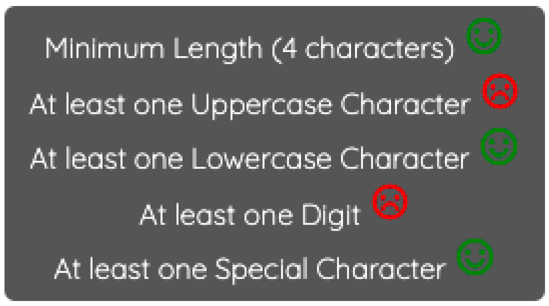
Minimum Length (4 characters) ☹

At least one Uppercase Character ☹

At least one Lowercase Character ☹

At least one Digit ☹

At least one Special Character ☹



Minimum Length (4 characters) ☺

At least one Uppercase Character ☹

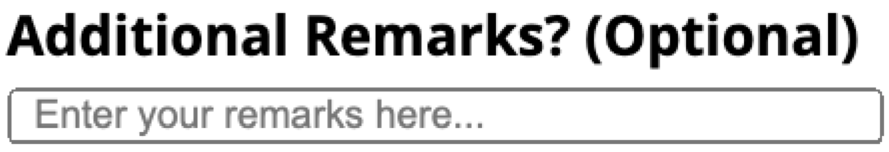
At least one Lowercase Character ☺

At least one Digit ☹

At least one Special Character ☺

Hover-over Text

Users are given the option to add *'Additional Remarks'* under each Task/Event. It was specially designed with the intent to be as unobtrusive as possible.



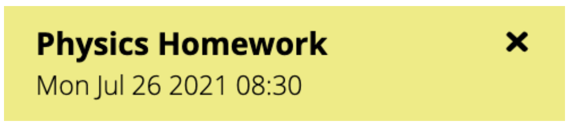
Additional Remarks? (Optional)

Enter your remarks here...

An example is shown below.

When the User is not hovering-over the Task/Event, the 'Additional Remarks' field will not appear (Figure 1).

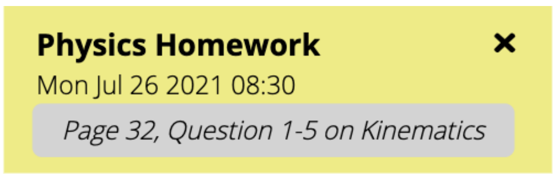
When the User hovers over the Task/Event, the 'Additional Remarks' will neatly appear within the box (Figure 2)



Physics Homework ×

Mon Jul 26 2021 08:30

Figure 1 - No hovering



Physics Homework ×

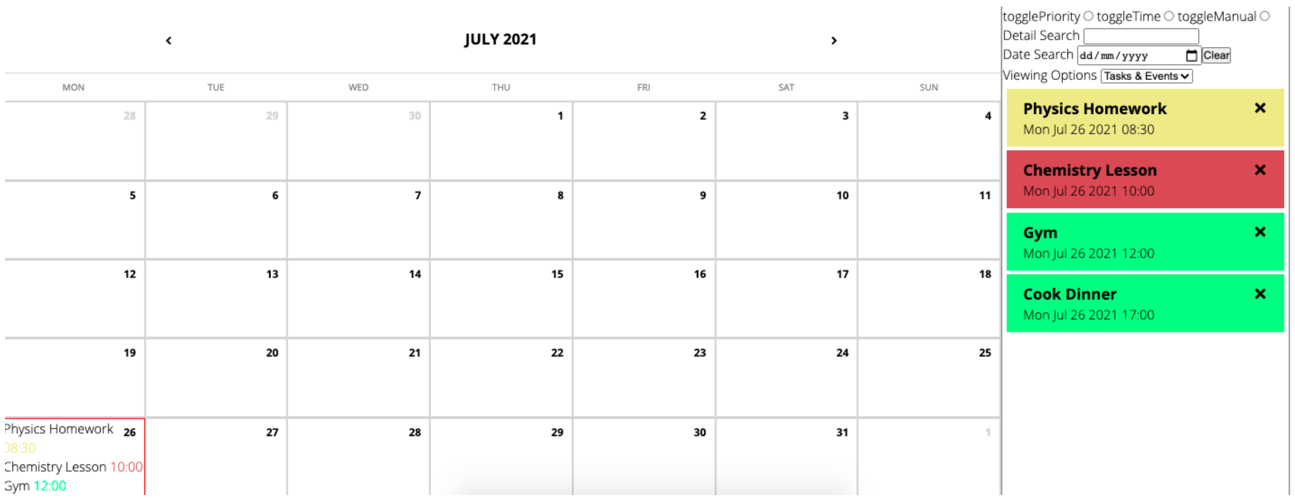
Mon Jul 26 2021 08:30

Page 32, Question 1-5 on Kinematics

Figure 2 - Hovering

Calendar ‘Day-View’ Feature

A drawback of the Calendar view is it’s lack of viewing space. Adding too many Tasks/Events in a single day will cause the cell space to fill up quickly, and the User is not able to view all Tasks/Events on that day from the Calendar’s POV. (Figure 1).



togglePriority ○ toggleTime ○ toggleManual ○
Detail Search
Date Search dd/mm/yyyy Clear
Viewing Options Tasks & Events

Physics Homework
Mon Jul 26 2021 08:30

Chemistry Lesson
Mon Jul 26 2021 10:00

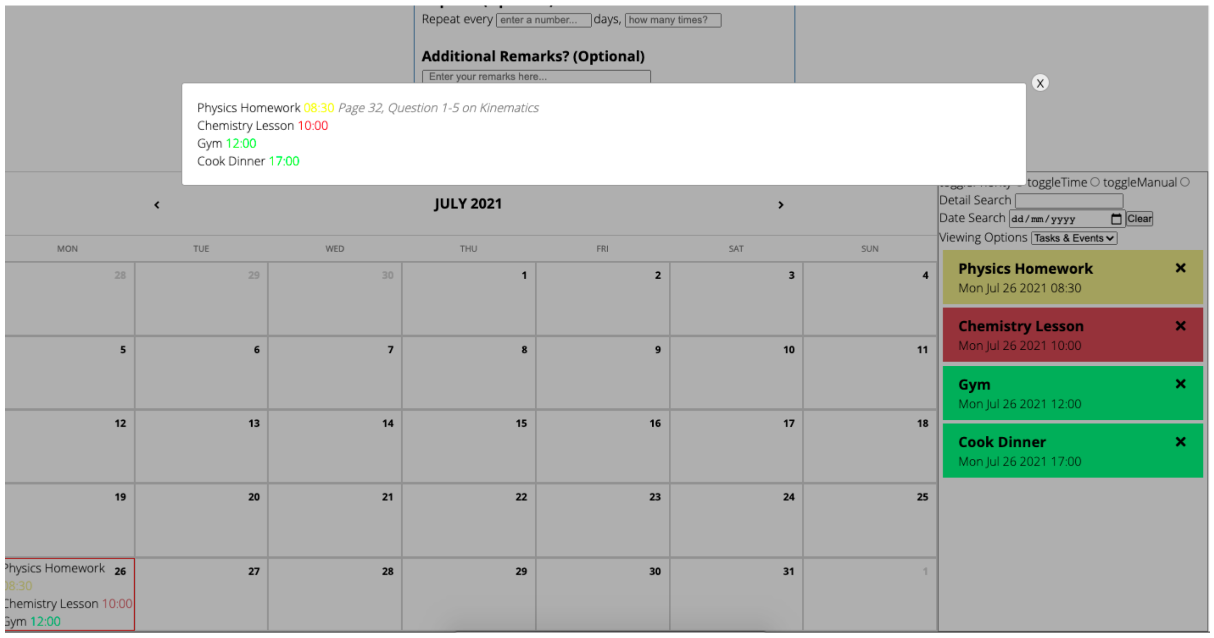
Gym
Mon Jul 26 2021 12:00

Cook Dinner
Mon Jul 26 2021 17:00

Physics Homework 08:30
Chemistry Lesson 10:00
Gym 12:00

Figure 1 - Observe how ‘*Cook Dinner*’ is missing from the Calendar’s cell

To rectify this issue, the ‘Day-View’ feature was added. Simply click on the specific cell that you wish to view all Tasks/Events for.



Repeat every enter a number... days, how many times?

Additional Remarks? (Optional)
Enter your remarks here...

Physics Homework 08:30 *Page 32, Question 1-5 on Kinematics*
Chemistry Lesson 10:00
Gym 12:00
Cook Dinner 17:00

toggleTime ○ toggleManual ○
Detail Search
Date Search dd/mm/yyyy Clear
Viewing Options Tasks & Events

Physics Homework
Mon Jul 26 2021 08:30

Chemistry Lesson
Mon Jul 26 2021 10:00

Gym
Mon Jul 26 2021 12:00

Cook Dinner
Mon Jul 26 2021 17:00

Physics Homework 08:30
Chemistry Lesson 10:00
Gym 12:00

(Manual) Software Testing

Registration and Login Unit Testing

Event Description	Expected Outcome	Bugs found / Steps Taken
Successful Register	alert(User Registered!)	
Unsuccessful Register	alert(Username Already Taken)	
Successful Authentication	Redirected to App	When logging-in from a fresh session, App needs to fetch data from MongoDB first, resulting in a seemingly unresponsive App. A loading screen was added to prevent the User from encountering this.
Unsuccessful Authentication	alert(Invalid Username/Password)	

AddList Unit Testing

Event Description	Expected Outcome	Bugs found / Steps Taken
User inputs duplicate Task/Event. A duplicate is defined as a Task/Event having similar Details, and having similar start times.	Users are disallowed from submitting the form.	Thought of providing more timely feedback by checking on every keystroke, however it would be too costly and slow down the system. Instead, on each submission we first scan through the respective User's database to check for duplicate items. If a duplicate item is found, we must first stop it from being saved in the database, before alerting the User of the error.
User inputs wrong Date/Time format	Text in Date/Time fields turns Red Upon correction, Text will return to Black Users are disallowed from submitting the form if the issue is not fixed.	onKeyUp event is utilized to check the input on every keystroke, providing timely feedback to the User if it is wrong. Used moment.js to check inputs. Many parsing-related bugs occurred, as these libraries have very strict requirements on what constitutes a date/time. This was fixed by ensuring a common 'date-time format' standard throughout, in this case, parsing to ***D MMM YYYY h:mma***, on **strict mode** for moment.js
For Events, the start time is similar to / after the end time.	Users are disallowed from submitting the form	Initial approach was to convert both date/time objects to unixTimestamp. However, an easier solution was to simply compare both moment objects using moment.js
User wants to repeat Task/Events at a	'Repeat every' field's text will turn red.	

more than 28 day interval.	Users are disallowed from submitting the form if the issue is not fixed.	
User wants to repeat Task/Events more than 4 times	‘How many times’ field’s text will turn red. Users are disallowed from submitting the form if the issue is not fixed.	
User inputs NaN into Repetition fields	Text under ‘Repetition’ fields will turn red. Users are disallowed from submitting the form if the issue is not fixed.	Initially, input type was set to “number” for these fields, which seemed tomitigate the issue of NaN. However, browser support for “number” input type was low, so reversion to input type “text” was chosen. A simple check for NaN on each keystroke will notify the User in time. Attempts were made to only allow inputs that were Digits (0-9). However, that proved much more difficult than anticipated. There were many edge cases to tackle (what if the User wants to backspace, or wants to hold down a key?). So a simple check on submission was used to make it easier on our side.

Todo List Unit Testing

Event Description	Expected Outcome	Bugs found / Steps Taken
togglePriority selected OR toggleTime selected	If togglePriority selected, Items in To-do List sorted in descending order by Priority. If toggleTime selected, items in To-do List sorted in descending order by start time.	Initially, a *.sort()* function was done in the **child** to reorder the items, before passing it up into the parent for use. However, this resulted in two bugs. Firstly, synchronization with the Calendar would be off, as mutating data in the child did not cause changes in the parent. Secondly, choosing the togglePriority/toggleTime option once would break the toggleManual option, due to the same reason as above. To solve both of these issues, a code refactor was done. By adhering to React's unidirectional data-flow model, it was much easier to *.sort()* the data in the parent, then passing it into the respective children to be rendered, compared to trying to *.sort()* in the child and passing it up to the parent for redistribution. Furthermore, a *.sort()* function was instead done on a copied version of the items, so as to not unnecessarily mutate the state that we are taking reference to.
toggleManual selected	Users should see the 'last-known' version of their Manual toggling. Users are only able to do *Drag n Drop* on List Items if toggleManual is selected.	*Drag n Drop* did not accurately capture information from Starting/Ending zones. The bug was due to two reasons. Although the *onDragOver* event is not used to capture any information, adding it made the bridge between *dragStart* and *dragDrop* work.

		Secondly, the wrong Events were used to trigger information capture. *onDragEnter* was initially used instead of *onDragDrop* to capture the Ending zone information. However, this resulted in unintended capturing of unneeded information, such as capturing the Start Zone the moment the *Drag n Drop* was initiated.
User types into ‘Detail Search’	List items will be filtered by partial string matching. Must work in tandem with toggleMode options.	
User selects a date under ‘Date Search’	List items will be filtered by start Date. Must work in tandem with toggleMode options.	Initially, there was no way for User to clear the date once chosen. Hence they could not view the entire List again if they wanted. A ‘clear’ button was added for them to reset the date field.
User deletes a Task/Event	List item will be deleted from To-do List	Sometimes the process takes longer than usual to run, which makes the User think that the delete function didn’t work. Perhaps a loading screen similar to the one used on Launch can be used.

Calendar Unit Testing

Event Description	Expected Outcome	Bugs found / Steps Taken
Clicking on Left/Right Chevrons	Calendar will render other Months. Task/Events in other Months must show up.	
Upon first render of Calendar, and with User not clicking on any of the cells	The cell corresponding to the current day should be highlighted.	Rightmost cells were not being highlighted completely (the right border was missing). Issue was due to overwriting in the .css file. Issue fixed by giving priority to the current-day class, which allows the appropriate effects to have priority.
At all times	Days not in the current Month should be greyed out. (Task/Event of those days should still be visible) Calendar should follow the Monday-Sunday structure.	Overlooked an alignment issue found when changing from Sunday-Saturday structure to Monday-Sunday structure. This was fixed by anchoring the start point of each Month to (current date + 1 day), which shifts everything forward by 1 day and aligns up to a Monday-Sunday structure.
User clicks on a Cell	Cell Border changes to Red.	

Integration Testing (Between To-do List and Calendar)

Event Description	Expected Outcome	Bugs found / Steps Taken
User successfully submits AddList form	Task is successfully added to both Calendar and To-do List	
togglePriority selected	To-do List and Calendar are sorted descendingly by priority - Events will always be placed below Tasks	
toggleTime selected	To-do List and Calendar are sorted descendingly by Time	
toggleManual selected	To-do List and Calendar are sorted by the User’s last known sorted preferences.	
User does Drag n Drop	The appropriate items are swapped, and the changes must be reflected in Calendar and To-do List	
User deletes a Task	Task is successfully removed from both Calendar and To-do List	
User utilizes ‘Detail Search’ feature User utilizes ‘Date Search’ filter	Tasks are filtered on both Calendar and To-do List	

Project Log

Edly (143h)

1	Python self-learning and practice • https://www.programiz.com/python-programming • https://www.youtube.com/watch?v=t8pPdKYpowI&t=7370s	1 July	59h
2	Telegram bot tutorials • https://github.com/tau-bar/qotd-bot/commit/ad572241e3753f0a9aa19348585ea3da8495c91f • https://www.codementor.io/@karandeepbatra/part-1-how-to-create-a-telegram-bot-in-python-in-under-10-minutes-19yfdv4wrq • https://www.youtube.com/watch?v=doPo9q6on6c • https://www.youtube.com/watch?v=8PvsGyAghDg • https://www.youtube.com/watch?v=7qJFtGi0hNQ&list=PL08sfd5Z39kw-Luzh4tYIycPtJT01szQE&index=11	15 July	41h
3	Implementation of Telegram bot	23 July	10h
4	Heroku deployment tutorial	24 July	2h
5	Heroku deployment	24 July	8h
6	Testing of Telegram bot	25 July	10h
7	Final changes to Telegram bot	26 July	13h

Damien (189h)

1	*Team Meeting* • Lift Off submission • Brainstorming on features • Creating rough Storyboard	16 May	6h
2	*Self-Learning (Online Resources)* • HTML • CSS (Flex + Grid) • JavaScript • Nodejs • Expressjs • MongoDB • React	16 May - 21 May	30h
3	*Starting of Prototype* 1. Creating Frontend UI of To-Do List (using React) 2. Connecting Frontend UI to mock Backend (json-server) 3. Authentication Service (Passport.js) 4. Porting over from mock Backend to MongoDB 5. Debugging • Alot of debugging time was spent on Passport.js and porting over to MongoDB 6. Trying to integrate UI with MongoDB Backend (WIP)	24May -31May	40h
4	Tasks can now be sorted by *Date and Time*	5 June	2h
5	Implemented bare-bones *Calendar* • Tried implementing *react-big-calendar* first, but found out after some time that it was actually deprecated (could not get it to work with latest version of React)	6 June	6h

6	Tasks/Events can now be displayed inside *Calendar* cells Bug fixes • Calendar now updates if add/delete on same day • Calendar now displays time alongside text • Calendar text now displays color Calendar text can now be toggled between *Priority* and *Time* • Spent significant amount of time debugging asynchronous behaviour, that was causing the webpage to 'lag' behind	7 June	8h
7	Team Meetup • Code Review • Storyboarding Tried implementing visual representation of Deadlines • Realised that it is likely unfeasible due to the way *Calendar* is constructed • Implemented a feature whereby "Current Day" is highlighted in *Calendar*	8 June	7h
8	Implemented "Color-Picker" for *Events* **ONLY** Events now accept time range Time range can be seen on Calender cells Refactor code for Calendar.js and App.js	9 June	5h
9	Implementing Drag n Drop for To-do List	10 June	10h
10	Implementing Drag n Drop for To-do List	11 June	10h
11	Implemented 'Repeated Adding' feature	16 June	2h
12	Integrating Drag n Drop with Priority and Time sorting	17 June	4h
13	Sanitizing User Input Added some interactivity (for when User inputs wrong info)	20 June	3h
14	Added search filter (by Date and by Task Name) Fixed bugs related to Drag n Drop	24 June	6h

	Disallow duplicate task/no task		
15	Documentation • Design • Implementation • Software Testing Implemented minor features • Loading Screen • Edited Calendar.css • UI adapts to Task/Event choices • Search by Detail / Search by Time / Search by Type Fixed some minor bugs • Calendar not adapting to Monday-Sunday structure • Form <number> input bug	25-28 June	25h
16	Authentication Security (Regex) Password Criteria (Hover-box)	27 June - 4 July	4h
17	Calendar 'Day - View' feature	5 July - 11 July	4h
18	Hover-over Text / Additional remarks Registration/Login animation Save Button Animation	12 July - 18 July	9h
19	Attempted to deploy on Heroku (failed)	19 July - 26 July	8h