Carleton University
COMP 4905 – Honours Project – Winter 2023

Autodo: A Web-Based Study Tasks Organizer System

Zifan Zhu, 101138493
Supervisor: Darryl Hill

Abstract

Autodo is an innovative web-based task organizer system that aims to improve users' time management and productivity. Autodo enables users to create, modify, and delete study events and other events, while also offering a unique automatic task allocation feature. Users can customize their preferences and personalized schedules, which ensures optimal task distribution and efficient use of available time.

It has ability to intelligently allocate tasks based on three allocation modes, catering to users with diverse time management preferences. The system’s custom algorithm efficiently assigns tasks to available time slots, maximizing productivity and minimizing stress associated with task management. Autodo also has a user-friendly interface and customization options make it a versatile tool for individuals seeking to improve their time management skills and optimize their daily routines. While there are various task organizer applications, Autodo sets itself apart through its task allocation feature and the level of personalization it provides, as well as being more student-oriented. By scheduling tasks based on users’ individual preferences and schedules, Autodo offers a more adaptable and efficient solution than traditional task organizers.

Acknowledgements

I would like to express my sincere gratitude to my supervisor, Darryl Hill, for his guidance, insights, and support throughout the course of this Honours Project. His questions and our discussions were helpful in maintaining my focus and determination throughout the process. I am also grateful for the opportunity to independently pursue this project, explore my own ideas and capabilities.

Table of Contents

List of Figures

1. Introduction

1.1. Chapter Overview

This chapter aims to provide the context and background for the Honours Project, Autodo, a web-based task organizer system with automatic task allocation. Autodo is designed to streamline the process of creating, modifying, and deleting tasks while providing a highly personalized and efficient platform for task organization and allocation. The system incorporates task allocation algorithm and customization options to optimize task distribution based on users' individual preferences and schedules, ensuring that tasks are allocated to available time slots in the most efficient manner possible. Autodo aims to be a comprehensive solution for managing academic, professional, and personal responsibilities, enhancing users' time management skills and overall productivity. Section 1.2 delves into the project's motivation in greater detail.

1.2. Project Motivation

Task management is an essential aspect of productivity and organization in today's fast-paced world. With the increasing number of responsibilities and commitments, it has become increasingly challenging to manage time and stay on top of tasks. Many people use task management applications to help them stay organized and productive. In the current internet field, there are various existing to-do applications, most of which are designed to help users store their daily events and provide date reminders. There are also many calendar applications that provide users with an overview of events on a calendar, including Microsoft To Do, which is specifically designed for creating and editing various new tasks, setting deadlines, and reminders, Google Calendar, a popular time-management tool for organizing user schedules and keeping track of events, appointments, and tasks. It provides users with a calendar view, including day, week, and month views. It also includes task planning, reminders, and other features. In addition to these two popular applications, there are many similar applications that help users store and plan various events and tasks. However, most of these task management platforms are generally similar. While these features are helpful, they fail to ddress the specific needs and requirements of individuals that are students.

During my college years, there were always many assignments, exams, quizzes, reports each semester. Therefore, I often spent each week busy with homework, reviewing, and writing reports. My friends and classmates around me sometimes liked to procrastinate, and even failed to submit their work before the deadline. Even they know the deadline for each task, students always do not have a clear idea of how to allocate their time. Autodo aims to solve this problem by providing a solution that allows students to allocate their study tasks according to their preferences and schedules.

Students can set their daily sleep time, weekly course time, daily repeat events (such as breakfast and lunch), and task allocation mode. This platform can provide users with a calendar view based on their personal settings to display their free time, and allocate the learning tasks added by students to free time slots according to different allocation modes. With this main feature, combined with the calendar view on the main interface, it can become the optimal place for students to allocate tasks, manage their time. And it would be easier for students to stay on top of their tasks, reduces stress, and achieve their goals efficiently.

2. Project Design

2.1. Chapter Overview

This chapter delves into the design aspects of the Autodo project, providing a comprehensive understanding of the system's architecture, technologies used, components, and user interactions. The chapter is structured into several sections, each focusing on different aspects of the project design:
Section 2.2: Project Use Cases - This section outlines the various use cases that Autodo is designed to address, illustrating the breadth of functionality offered by the system to cater to users' task management needs.
Section 2.3: Server-Side Technologies - This section describes the server-side technologies used in the development of Autodo, detailing the tools and libraries employed to create the server, API, and database components of the system.
Section 2.4: Client-Side Technologies - This section discusses the client-side technologies used in the development of Autodo, focusing on the front-end framework, user interface, and client-side data management tools.
Section 2.5: Deployment Technologies - This section covers the deployment technologies utilized in the Autodo project, including the hosting services for the back-end server and front-end application.
Section 2.6: Chapter Summary - This section provides a concise summary of the project design elements discussed throughout the chapter, highlighting the key aspects that contribute to the overall functionality and user experience of the Autodo system.

2.2. Project Use Cases

This section outlines the various use cases that Autodo is designed to address, illustrating the breadth of functionality offered by the system to cater to users' task management needs. The use cases are detailed below:

User Registration and login: User can register for an account simply by using a username and password and log in to access the application's features. The authentication and authorization process ensures data privacy and security.
User profile management: Users can update their profile settings, which include sleep time, courses, and allocation mode. These preferences influence how tasks are allocated and scheduled within the system.
Creating events: Users can create study and other events by selecting dates on the calendar panel or clicking the "Create Event" button. Study events can be configured as "auto" or manual, influencing how they are allocated on the calendar.
Modifying events: Users can modify existing events by dragging and dropping them to new start and end dates, resizing them to change new end date directly on the calendar, or updating their properties through a pop-up window by clicking the event on the

calendar. And changes would be saved and reflected on the calendar.
Deleting Events: Users can delete events by clicking them and click the delete button to easily delete them.
Setting Sleep Time: Users can set their preferred sleep time by visiting the profile page, and toggle the edit button, go to the "ROUTINE-SLEEP" tab in profile page then entering their desired start, end time to set sleep time.
Managing Courses and Course Times: Users can add, or delete courses and their associated times on the profile page, "COURSES" tab. After user adds some courses to the system, they can also add corresponding course time in profile page, "ROUTINE-COURSES" tab by entering recurring day of week and start, end time. These settings are used to calculate free time slots for task allocation.
Managing Daily Recurring Events: Users can create, delete daily events that recur each week. These events are also considered when calculating available time slots for task allocation.
Automatic Task Allocation: Study events marked as "auto" (user can select this when creating study event) are automatically allocated to free time slots based on the user's preferences and allocation mode. The system supports three allocation modes: early, average, and late. In addition, tasks would be allocated when user change their tasks allocation mode. Changing the mode can be easily done in profile page, "MODE" tab.
Viewing Free Time Slots and Events: Users can view their events directly on the calendar in different views (month, week, day, or list). What's more, in the week/day view, user can view the available time slots highlighted in a lighter color for easy identification.
Viewing Today's Events: Users can view a list of their study or other events for the current day in a panel next to the calendar by clicking "study" or "other" panel button.
Notification: Events that close(due) within two days are added to the user's reminder message. Users can view these messages by clicking on the bell symbol near the profile button.

These use cases can be adapted to various contexts and situations, demonstrating the versatility and wide-range applicability of the platform. Students can use Autodo for managing individual group projects, coursework, and deadlines, organizing their study schedules and allocating time for academic tasks, improving overall academic performance. Autodo can also be used for personal time management, helping users to allocate time for non-study-related activities. By using Autodo to schedule and manage personal tasks, users can maintain a healthy work-life balance and ensure that they have time for the things that matter most to them.

2.3. Server-Side Technologies

Autodo is a platform designed to store various data related to users and assign tasks based on the users' data. To achieve this, a server that provides data interfaces is necessary. For this project, the implementation of a RESTful Node.js Express server that offers access to multiple endpoints is needed. These endpoints enable clients to interact with the back-end server to retrieve user data. The Appendix B provides detailed information about the various APIs on the back-end.

Utilizing Node.js and Express for implementing the server has significantly simplified the development process and enhanced the platform's usability, scalability, and flexibility. Firstly, it allows the JavaScript code to create a more uniform development environment. Additionally, the Node Package Manager (NPM) provided by Node.js makes it easy to obtain all the required libraries for the entire server development of the project. In case multiple developers work on the project, the package.json file ensures that all required packages can be effortlessly installed using 'npm install'. This process enables to seamlessly work with the development environment and run the code locally. Any new packages, libraries can be easily installed using NPM. Node.js's event-driven, non-blocking I/O model, allows it to handle multiple requests simultaneously, it also has single-threaded architecture, cluster module, and rich ecosystem of modules and packages.[1] These all together make it a highly scalable platform for building scalable applications.

Autodo requires ability to store document-oriented data such as user events, settings, and other related information, thus MongoDB is an ideal choice for the back-end data storage. MongoDB's flexibility allows developers to easily modify data requirements, model data, and query data. MongoDB uses a JSON-like format to store data simplify the ways to store, manage, and retrieve data with most programming languages.[2] Stored documents in MongoDB are BSON files which is human-readable, and it allows user to nest JSON to store complex data objects.[2] Autodo also requires the ability to create fields, deleting fields (leaving a field out) of user data, and the JSON-like format data in MongoDB can take care of these issues. Therefore, the MongoDB would be the most appropriate database for Autodo data storage. Moreover, the platform leverages the schema-based Mongoose technology for efficient interaction with MongoDB. It enables developers to effortlessly apply schema validations, manage relationship between data, and perform various CRUD operations. Created user schema can be used to query, modify, delete data easily with Mongoose technology.

In addition to the core technologies mentioned previously, Autodo also employs server other essential technologies to ensure the platform's security, performance and reliability. To protect the communication between the client and server, Autodo uses https, a secure version of the HTTP protocol. HTTPS ensures that all data transmitted between the client and server is encrypted, safeguarding the platform against eavesdropping and man-in-the-middle attacks. JSON Web Tokens (JWT), and cookie-parser are used to ensure authentication and reduces server load, as the server does not need to store session information. UUID is also utilized to generate unique IDs for events to ensure each event has a distinct identifier, which is crucial for efficient data management, querying, and retrieval.

In Conclusion, Autodo's back-end infrastructure is built upon a combination of core and auxiliary technologies such as Node.js, Express, MongoDB, Mongoose, HTTPS, CORS, JWT, UUID, and cookie-parser. These technologies work together to create a secure, efficient, and scalable platform, providing users with a seamless and reliable experience while managing their tasks and data effectively.

2.4. Client-Side Technologies

The client-side of Autodo has been developed using a combination of modern technologies and libraries that interactive web application. The main components of Autodo's client-side architecture include the React.js framework, React-auth-kit, React-Router-Dom, Material-UI, and Cookies. React.js is a popular JavaScript library for building user interfaces. This framework gives features like Virtual DOM, JSX, Components, Props and State that promote modularity and reusability of code, and optimize the updating and rendering process of the application.[3]

React-Auth-Kit is a library that gives the Client-Side authentication, the library integrates seamlessly with React-Router-Dom, which is used for managing navigation and routing within the application. These two together provide a way to easily navigate through different pages with authentication, they also efficiently handle navigation between different views without the need for full page reloads.

Material-UI is a popular React UI framework that offers fully-loaded component library. Those comprehensive, customizable components and styles ensure a smooth, modern look application. Using it makes the application development efficient and accelerated. The most important library used is FullCalendar, which is a powerful and flexible JavaScript calendar library that integrates seamlessly with React.js. It provides features like creating, displaying, editing, deleting events within a calendar view. It also supports different views with corresponding plugins (e.g., month, week, day, list). Event creation, display business hours, show user's free time slots make it a perfect library for Autodo's event scheduling and management requirements.

Lastly, using cookies to store user information (e.g., email, jwt) from backend makes the authentication easier.

In summary, Autodo's client-side architecture is build on a solid foundation of technologies and libraries, including React.js, React-Auth-Kit, React-Router-Dom, Material-UI, Cookies, and FullCalendar. These components work together to provide a seamless, responsive, and visually appealing user experience, ensuring that users can efficiently manage their tasks and schedules with ease.

2.5. Deployment Technologies

Since the project need to be tested, the project deployment involves using a combination of cloud-based services. AWS EC2 and AWS Amplify are the primary deployment technologies utilized in this project, along with CORS (Cross-Origin Resource Sharing) for enhanced security and accessibility.

AWS EC2 allows users to create virtual machines (called instances), on which applications back-end can be deployed and run. The Node.js Express server of Autodo is deployed on an EC2 instance enabling the application to handle requests. However, due to the inbound rules used for the instance, using HTTPS requires SSL certificate, and the certificate in this application used is a self-signed free certificate. As a result, most common browsers would not trust the URL by default. During testing phase, tester must allow access to that URL before using the application. This limitation should be taken into consideration and be addressed in future to ensure seamless and secure user experience.

CORS(Cross-Origin Resources Sharing) is a security feature implemented in the server-side application to allow secure communication between the client-side application and the server. It enables the server to specify which origins (domains) are allowed to access its resources, thus preventing unauthorized access from malicious websites or third-party applications. CORS in the server-side ensures only authorized clients can interact with its APIs and services.

In conclusion, the deployment of Autodo's web application leverages AWS EC2, AWS Amplify, and CORS to provide a environment to access it.

2.6. Chapter Summary

This chapter provided an overview of the design and technologies used in Autodo web application. It covered the various use cases that the application addresses. Client-Side and Server-Side also leverages a combination of powerful and versatile technologies that enable it to efficiently address its intended use cases. However, the self-signed SSL certificate could cause issues with browser trust. This limitation should be address in future development to ensure a seamless and secure user experience. In summary, the chosen technologies provide a solid foundation for the application's development and ensure that users can interact with the system in a secure and efficient manner.

3. Tasks Allocation Algorithm Elaboration

3.1. Chapter Overview

In this chapter, we delve into the design and development of our custom task allocation algorithm. We begin by discussing the research conducted on existing task allocation algorithms and related approaches. However, given the specific requirements of this project, which demands seamless interaction with the FullCalendar library and the generation of event formats compatible with our user data, a custom algorithm is essential.

As we progress through the chapter, we will elaborate on the design and implementation of this custom algorithm, ensuring that it efficiently allocates tasks while considering users' personal schedules and preferences. Additionally, we will explore the challenges and decisions encountered throughout the development process, highlighting the importance of a tailored solution for our unique project requirements.

3.2. User Data Schema

To better understand the task allocation algorithm for the AutoDo application, it is important to familiarize ourselves with the user data schema. The user schema, created using Mongoose, contains various fields, such as email, password, courses, coursesTime, sleepTime, taskAssignMode, skipNumber, routine, events, and messages. For user schema, see Appendix A. These fields store essential information that the task allocation algorithm utilizes, as shown in the following example user data. The important fields that the algorithm will use here include course time, sleep time, task allocation mode, skipNumber, routine, and events.

3.3. Data Preparation

The first step in the task allocation algorithm is data preparation. The user's settings and preferences, such as course time, sleep time, skip number, and routine events, are used to prepare the data for the algorithm. Based on these settings, we can obtain a data structure containing the free time slots for each day.

We start by introducing a "`day`" class, which mainly includes an array of "`timeSlots`" to store the available study time slots for each day. Additionally, it contains necessary information such as the date, day of the week, and provides functions such as "`addEvent(e)`," "`getNextTimeSlot()`," "`backTimeSlot(timeSlot)`," etc. to support “`timeSlots`” array modification.

Next, we create an array of days to store the user's fixed free time slots for each day of the week. Here, the user's routine, sleep time, and course time come into play. Since these events are fixed and repeated on certain days of the week, they are assigned to an array of 7 days through "`someDayOfWeek.addEvent(e).`" where "`someDayOfWeek`" could be a specific day object in array of 7 days called "`week`". This updates the free time slots array for each day of the week and represents the user's fixed available time slots for a week.

After completing the week structure, we need to create the "`days`" structure by calculating how many days the user needs to allocate tasks. We generate a corresponding number of "`day`" objects based on the difference in days between today's date and the deadline of the last task. Since each day has a "`day of week`" attribute, we can initialize the fixed free time for all these days using the "week." Then, we can continue to add the user's "`other`" events and "`study && !auto`" events to the corresponding day through a map (`key: date, value: day object`). This initializes all the time slot arrays for the necessary day objects in "`days`".

3.4. Tasks Allocation Algorithm

With the necessary data prepared, we can now implement the task allocation algorithm. To discuss the algorithm, we need to introduce several helper functions first:

1. `eventsReorganization(events)`: Reorganizes the user's events array to merge similar events (since the algorithm will split the same event into multiple sub-events and allocate them to appropriate time slots, it's necessary to merge those events before reallocation).

2. `getNextTimeSlotFromDays(days,nextAvailableDay,dueDate)`: A function that needs three parameters: "`days`" is the data structure generated in the preparation stage, "`nextAvailableDay`" is used to keep track of the next available day with free time slots, and "`dueDate`" is the event's due date. This will be used in the "early mode" allocation algorithm. It first traverses the "`days`" in time order and looks for next available time slot then return the found one along with new `nextAvailableDay`. With `nextAvailableDay` returned, we don't have to search for the next available time slot from the begging of the `days`.

3. `getNextTimeSlotSkip(days,nextAvailableDay,iteration,skip, dueDate)`: This function requires four parameters: "`days`," "`nextAvailableDay`," "`iteration`" represents the current loop iteration, "`skip`" represents the user's "`skipNumber`" as the time interval for event allocation, and "`dueDate`" represents the deadline for the current event. This helper function changes the way

to traverse `days` by introducing a new parameter `skip`. When traversing the `days`, instead of increasing the index by 1 each time, now increase the index by `skip.` This way, we can find the next available time slot at a certain time interval. `nextAvailableDay`, keeps track of the index of the day in `days`, while `iteration` keeps track of the iteration of loop we're on. Here's an example to explain the iteration number:

```
Assuming skip=3, length of days=7;
Iteration 0:
   Traverse order of the days array: 0, 3, 6;
Iteration 1:
   Traverse order of the days array: 1, 4;
Iteration 2:
   Traverse order of the days array: 2, 5;
```

4. `getNextTimeSlotBeforeDueDate(days,nextAvailableDay, dueDate):`
   This function has the same logic as the first one, except it traverses the "`days`" backward from the "`dueDate`" to find available time slots.

5. `reallocate(unAllocatedEvents,userEvents):`
   A function used to allocate unallocated tasks if there is not enough time to allocate all tasks. This function searches for the allocated event closest to its deadline and reallocates the time slots based on their weight ratio.

In addition, most of the date operations used in these functions use the "dayjs" library, which is a JavaScript date utility library that allows users to format, add, subtract dates and perform other time-related operations.

The task allocation algorithm has three main allocation functions:

`allocateEarly(user):`
This function calls the "`getNextTimeSlotFromDays`" helper function for each study event to find available time slots and changes the start and end times of the event to allocate it to the available time slot. Therefore, this allocation mode allows user to allocate their tasks as early as possible so that they can finish their coursework early and don't have to be too nervous or pressured to face learning tasks.

`allocateAverage(user):`
This function calls the "`getNextTimeSlotSkip`" helper function to find time slots and allocate tasks. It utilizes the user's "`skipNumber`" setting to allocate tasks with a balanced distribution. Hence provide users with a better solution so they can have a more optimized balance between study and life.

`allocateLate(user):`
This function calls the "`getNextTimeSlotBeforeDueDate`" helper function to find available time slots and allocate tasks as late as possible. Which gives user an overall idea or reference for the latest time to start study tasks.

However, if the time slots are not enough for users to allocate tasks, `reallocate` function comes into play. The unallocated tasks generated by these three common allocation algorithms will be reallocated by the `reallocate` function according to their weight and due date to allocate them to suitable time slots.

In summary, Chapter 3 provides an in-depth explanation of the task allocation algorithm, starting with the user data schema and the necessary data preparation. The chapter discusses helper functions and the three main allocation functions, which are designed to cater to different user preferences for task allocation. The "dayjs" library is utilized for efficient date operations throughout the algorithm.

3.5. Algorithm Analysis

The time and space complexity of the task allocation algorithm are crucial to understanding its efficiency and performance. This section will discuss the complexities concerning various aspects of the algorithm.

3.5.1. Time Complexity

The time complexity of the task allocation algorithm depends on the user's personal schedule and preferences. The initialization of the `week` and `days` structure is based on the number of weekly recurring events, course times, non-study events, and non-automatic study events. Consequently, the preparation stage before calling the algorithm will take time of
`O(# of weekly recurring events + # of user's "other" || "!auto" events).`

During the task allocation phase, the time taken varies depending on the specific situation. The search for available free time slots is linear. For event allocation, the complexity is also linear (loop through the sum of the duration of all study events in auto mode). However, if there are too many time slots so that events must be divided into many parts, the time required becomes number of time slots. Only if the duration of most events is about the same size as the duration of each time slot, then the time taken would be number of events. In most cases, since the number time slots and sum of free available time slots are fixed, and the events can only be assigned to those time slots, the time complexity would be **`O(# of time slots).`**

In summary, the time complexity of this algorithm will vary with the user's personal schedule and preferences. Nevertheless, since the algorithm is designed to reduce the number of unallocated tasks, the time complexity should be acceptable in most cases. And the overall time complexity would be:
`O(# of weekly recurring events + # of user's "other" || "!auto" events + # of time slots)`

3.5.2 Space Complexity

The space complexity of the task allocation algorithm depends on the number of day objects created and the time slots array included in each day object. Since it is not known whether a day's available time slots will be used for allocation before task allocation, it is necessary to create a number of day objects equal to the number of days between today's date and the last event due date. Therefore, the space complexity should be **`O(# of days between today's date and the last event due date).`**

However, the time slots array in each day object will vary depending on the situation, and there may be unused time slots. This indicates that there is potential for optimization of space complexity.

In conclusion, the time and space complexities of the task allocation algorithm are influenced by the user's schedule and preferences. Despite variations in complexity, the algorithm is designed to provide efficient task allocation in most cases. Further optimization of space complexity can be explored to improve the algorithm's performance.

3.6. Chapter Summary

In this chapter, we discussed the task allocation algorithm and its various components in detail. The algorithm is designed to allocate tasks to users effectively, considering their personal schedules and preferences. This chapter begins by examining the user data schema and explained the different components that the algorithm utilizes. Then elaborated on the algorithm’s two main stages: data preparation and task allocation. And last, we analyzed the time and space complexity of the algorithm. In conclusion, the task allocation algorithm is designed to provide a way for users to allocate their tasks automatically. By carefully considering the user’s schedule, preferences, and the algorithm’s time and space complexities, the project aims to create an effect task management solution.

4. Project Development

4.1. Chapter Overview

This Chapter provides a detailed overview of the project development process, including the client-side architecture, server-side architecture, API design, and API testing.

4.2. Client-Side Architecture

The client-side of the application is developed using the React.js framework. React.js is a popular library that is widely used for building user interfaces. And we already discussed it in Section 2.4. It allows for the creation of reusable components to improve the modularity of the code, hence we can use this feature to write reusable code. To support the authentication and navigation of the application, the auth provider from react-auth-kit and browserRouter from react-router-dom are used. These two technologies provide the app authentication and page navigation as discussed in Section 2.4. The app component is the top-level component of the client-side application. In the app component, themeProvider and cssBaseLine from MUI are used to support application theme and styles configuration. Various pages are navigated using routes, with the `"/"`, `"/signup"`, `"/main"`, and `"/profile"` routes leading to the login, signup, main, and profile pages, respectively. These four different views utilize reusable components, including calendar, edit, and topbar components. This modular approach promotes code reusability and maintainability.

4.3. Server-Side Architecture

The server-side of the application is built using JavaScript, Node.js, and the Express server. These technologies facilitate the implementation of APIs, business logic, and data persistence. The server-side architecture is organized as follows: API Routes: the Express server handles incoming requests and routes them to the appropriate API endpoints. Algorithm implementation: The task allocation algorithm is designed and implemented in a separate file and exported as a module function. This modular approach ensures that the algorithm can be easily modified and maintained. Database interaction: The server interacts with a cloud-based MongoDB database for data editing and querying. The Mongoose schema for database interactions is contained in another file and is exported as a module schema model, enabling CRUD operations on user data.

4.4. API Design

The API design is detailed in Appendix B. The APIs are designed to enable effective communication between the client-side and server-side components of the application. The APIs are RESTful, offering a clear and concise structure for requests and responses. Key aspects of the API design include: Consistent Naming Conventions: The API endpoints follow a consistent naming convention, making it easy for developers to understand and use the APIs. Proper Use of HTTP Verbs: The APIs utilize appropriate HTTP verbs (GET, POST, PUT, DELETE) to perform CRUD operations on resources. JSON Data Format: The APIs use JSON as the primary data format for request and response payloads, ensuring compatibility with various client-side libraries and frameworks.

4.5. API Testing

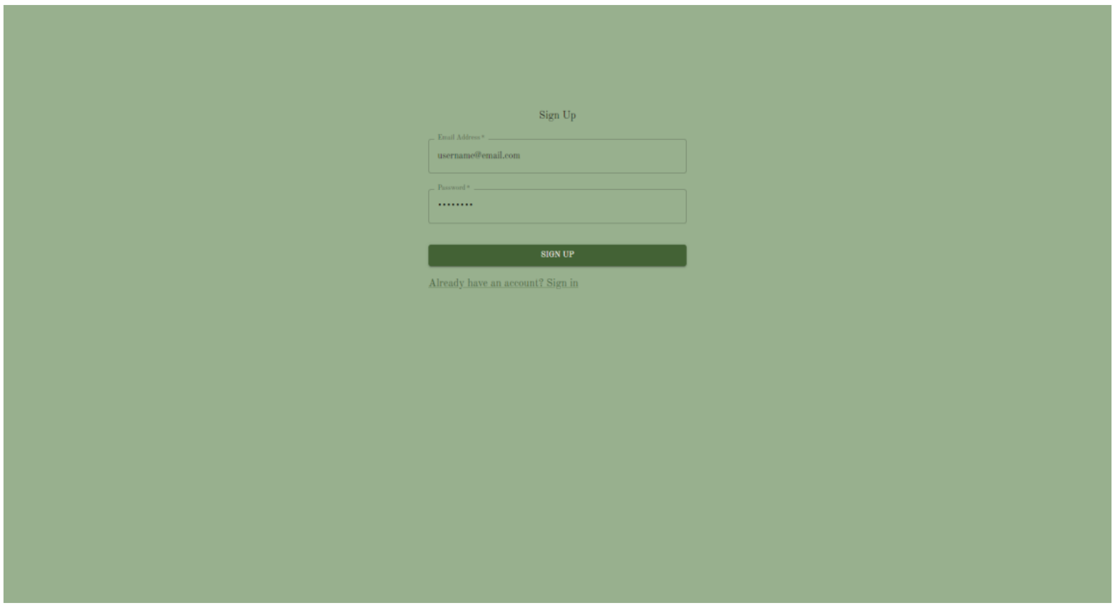
During the development of the server-side APIs, testing is crucial to ensure that the expected output aligns with the actual response data generated. The Visual Studio Code offers an extension called REST Client, which allows users to send all RESTful requests to the backend, including the request body. Key aspects of API testing include: Comparison of Expected and Actual Outputs: The expected output can be compared with the actual response data generated using the REST Client, verifying that the APIs function as intended. Error Handling: API testing also involves testing error handling and ensuring that appropriate error messages are returned when invalid inputs are provided or other issues occur.

5. Application Results

5.1. Chapter Overview

This chapter presents the results of the application, focusing on user authentication and registration, usability and accessibility, and user interface interactions. Relevant images and pictures are included throughout the chapter to provide a visual representation of the application's functionality and user experience.

5.2. User Authentication and Registration


Sign Up

Email Address * : username@email.com

Password * : ••••••••

Figure 5.2.1: Sign up page for user to registrate

Figure 5.2.1 provides inputs field for user to enter their username/email, password to create an account. Then user would be able to login to the application.

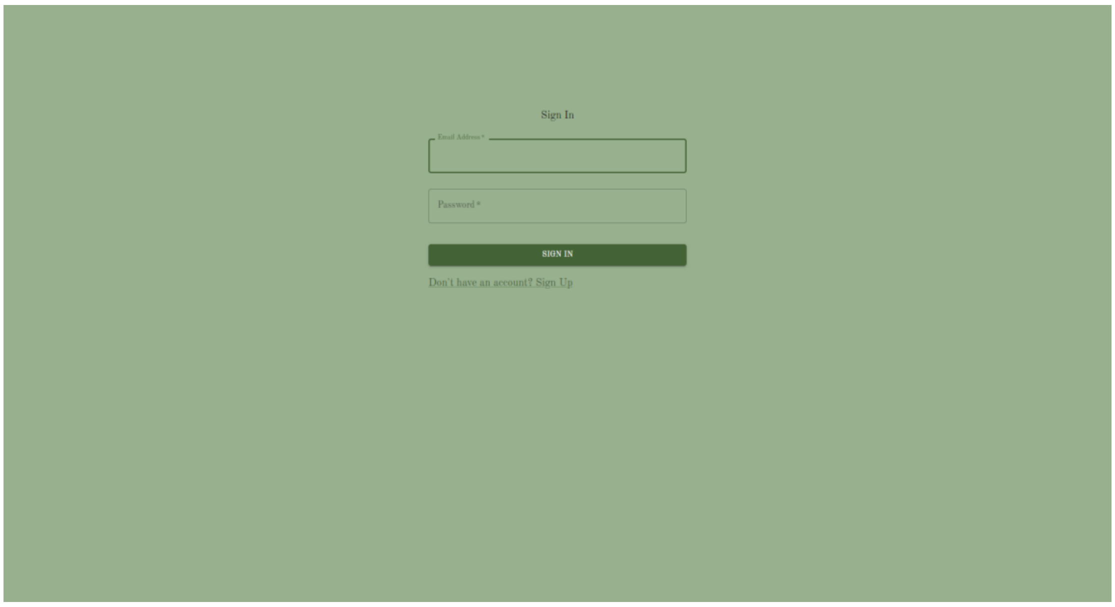


Figure 5.2.2: Log in page for user to log in.

Figure 5.2.2 provides inputs for user to log in to the platform.

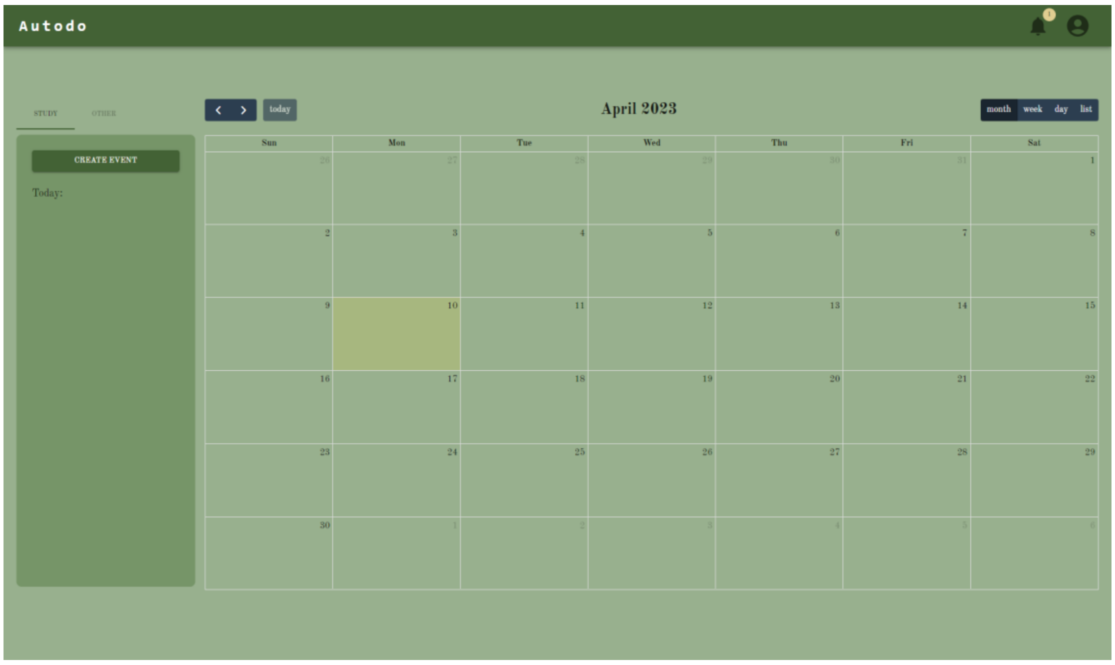
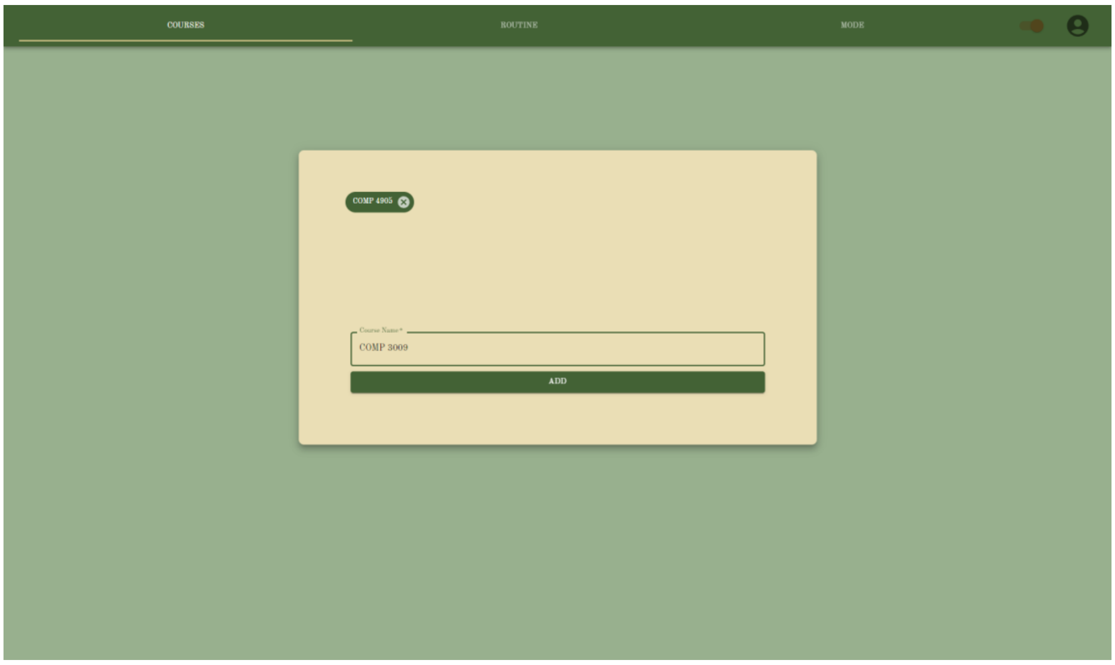


Figure 5.2.3: Main page

Figure 5.2.3 shows the basic layout in the main page. The left box shows what "study" events the user needs to do today, or what "other" events user needs to do if click the other tab button. The calendar on the right shows a calendar that will contain user's events. Now lets show the profile page before we talking about creating events.

Note: The small interface (mobile view) deletes the left panel and only displays the calendar, but other pages have similar components.

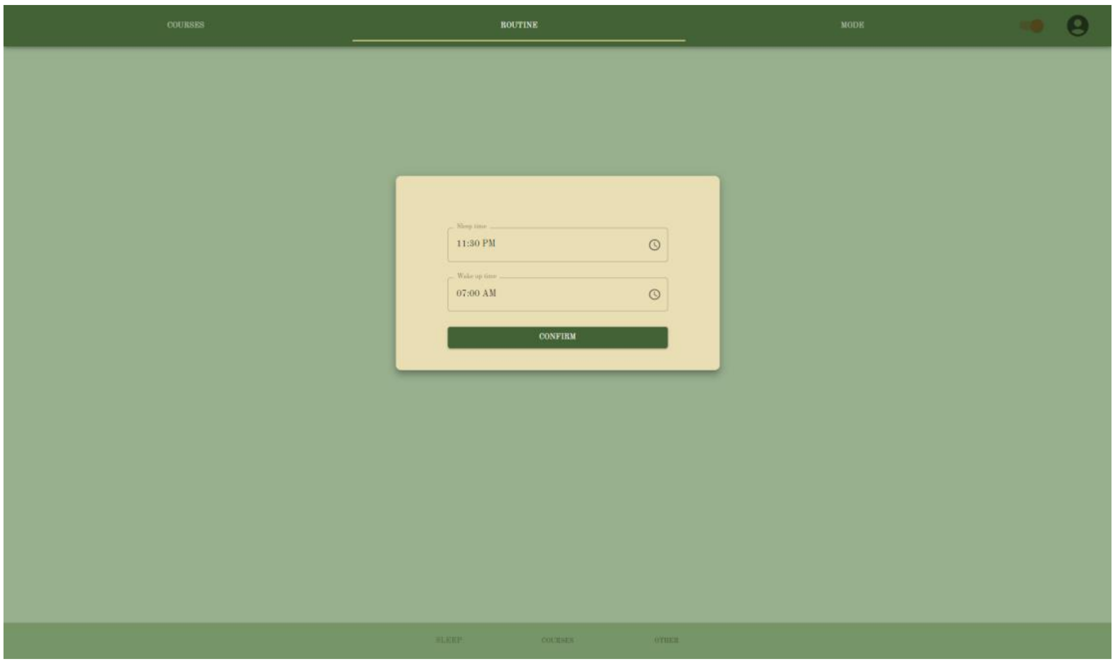


COMP 4905

Course Name *
COMP 3009

Figure 5.2.4: Profile page, courses tab

now if user click the top right icon, there will be a dropdown menu shows up, and user can go to profile page or log out. If user goes to the profile page, the first view would be courses tab. In this page, user can check what courses they have, they can click the edit button next to the profile button to add courses just like Figure 5.2.4. User needs to enter course name and click add and there will be a chip like course name shows in the container. User can click the delete button on the chip to delete a course, and all the course time relate to will also be deleted. We will discuss course time later.

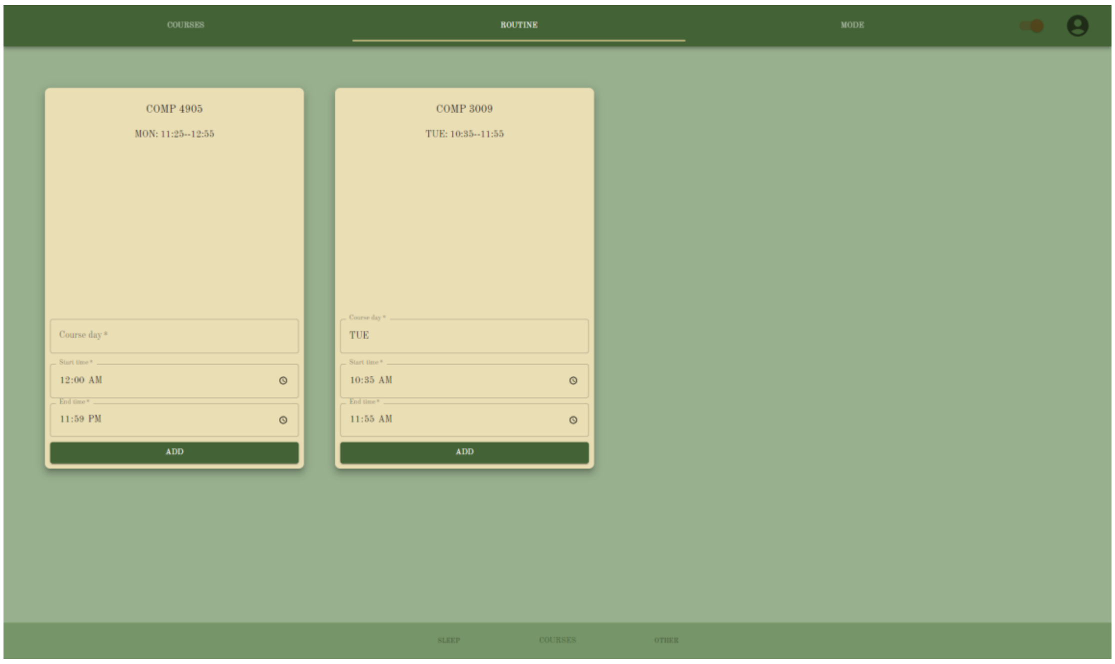


Sleep time
11:30 PM

Wake up time
07:00 AM

Figure 5.2.5: Profile page, routine-sleep tab

Figure 5.2.5 shows how user can change their sleep time once user click the edit button to enter setting mode. The two inputs require user to enter sleep time and wake up time.



COMP 4905

MON: 11:25--12:55

Course day *

Start time *: 12:00 AM

End time *: 11:59 PM

ADD

COMP 3009

TUE: 10:35--11:55

Course day *: TUE

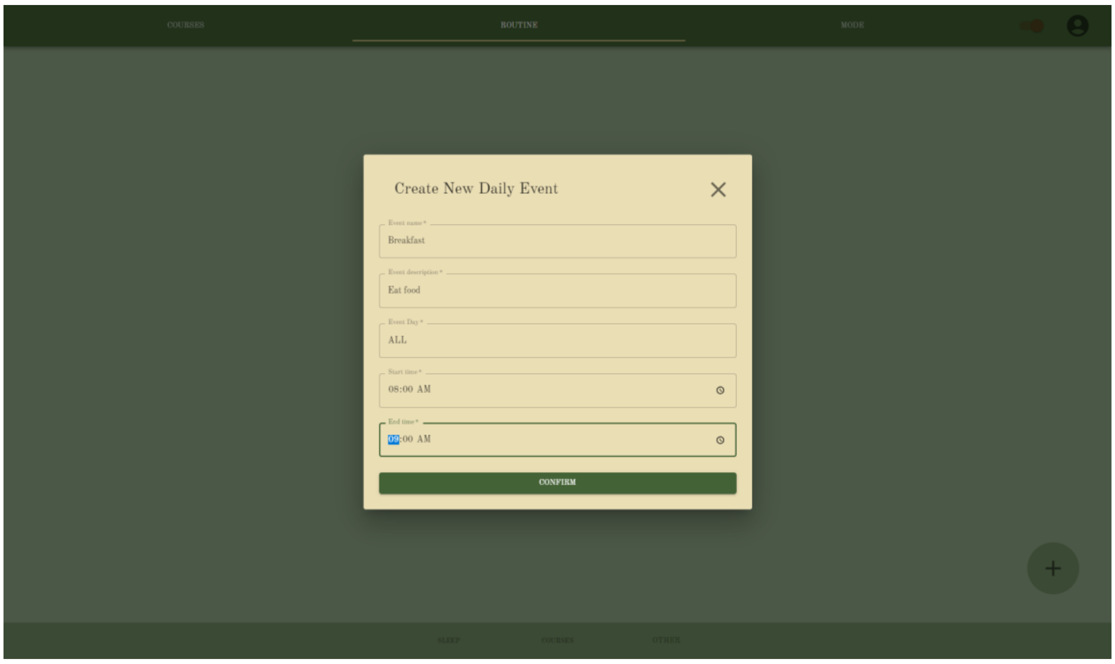
Start time *: 10:35 AM

End time *: 11:55 AM

ADD

Figure 5.2.6: Profile page, routine-courses tab

Figure 5.2.6 shows the look of page for user to edit their courses time. They can enter one of “MON, TUE, WED, THU, FRI, SAT, SUN, ALL” to specify the day of week of the course and enter the start, end time for a course. Click add button would add a course time for that course.



Create New Daily Event

Event name *: Breakfast

Event description *: Eat food

Event Day *: ALL

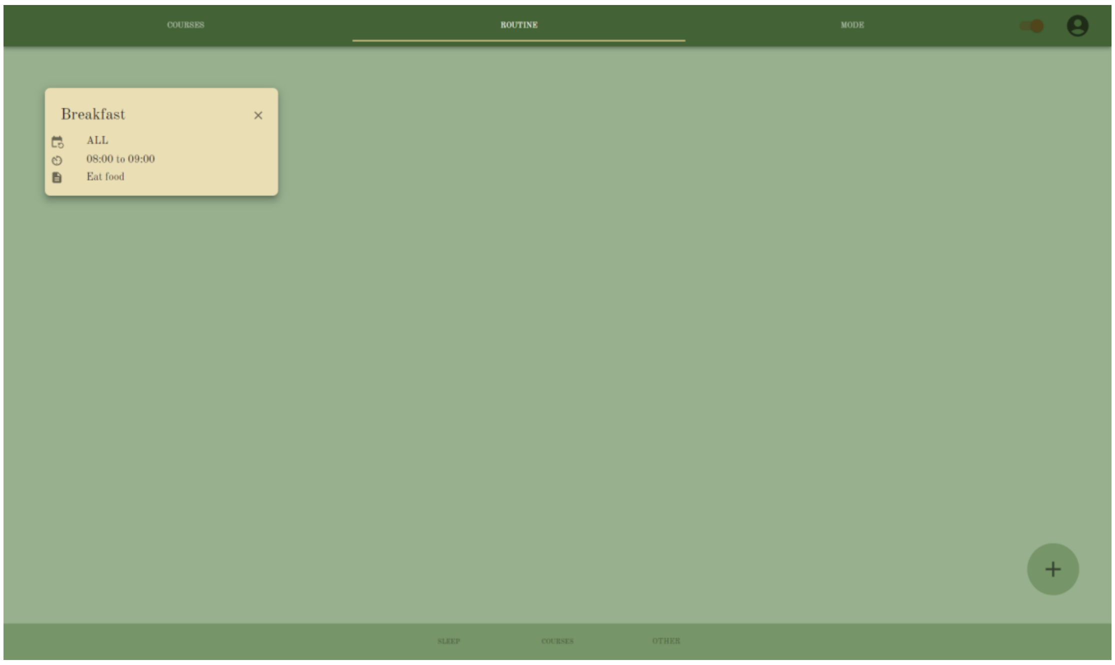
Start time *: 08:00 AM

End time *: 09:00 AM

CONFIRM

Figure 5.2.7: Profile page, routine-other tab

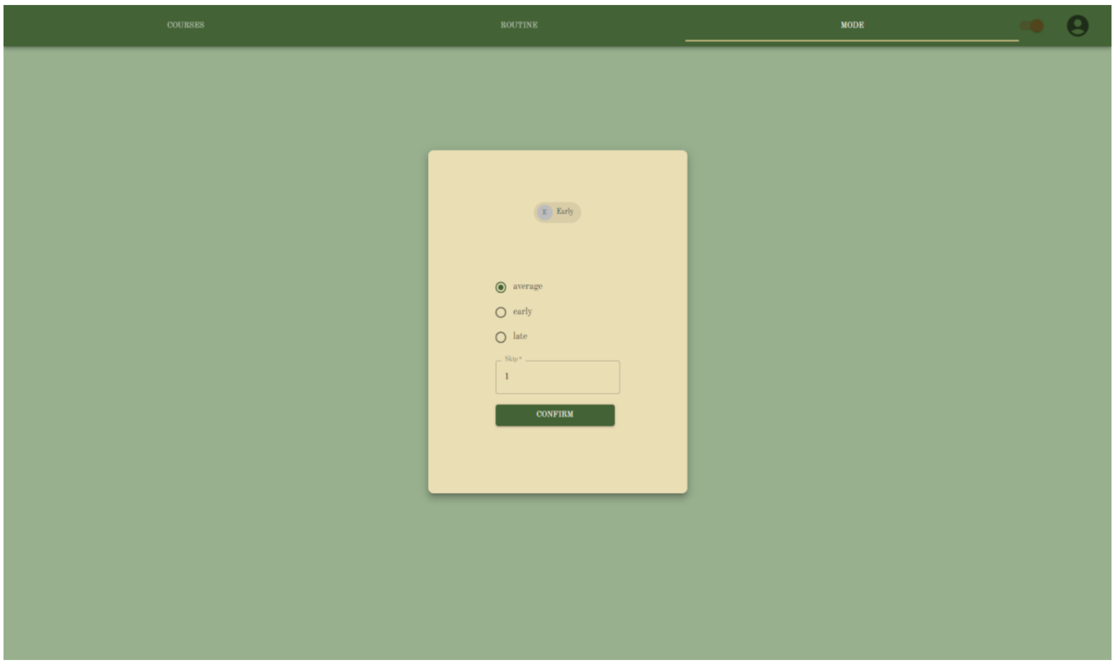
Once user goes to the routine-other tab, user can click the right bottom “+” button to create a new daily event like breakfast, lunch, gym, etc. User also needs to specify the start time, end time, recurring day of week.



Breakfast
ALL
08:00 to 09:00
Eat food

Figure 5.2.8: Profile page, routine-other tab

Once user created a routine event, they can see it in this page. Also, they can delete the event by clicking the delete button on the right top corner of that event card.



[✓ average] early late

Skip *: 1

Figure 5.2.9: Profile page, mode tab

Figure 5.2.9 Shows how user can change their tasks allocation mode. Average mode needs a “skip” number to determine time interval for tasks allocation. Average mode means user’s study events will be allocated evenly depends on the “skip” number. Early mode just allocates user’s events as early as possible. Late mode allocates user’s events as late as possible but before the due date of events. Currently we will use 2 as “skip” number and confirm.

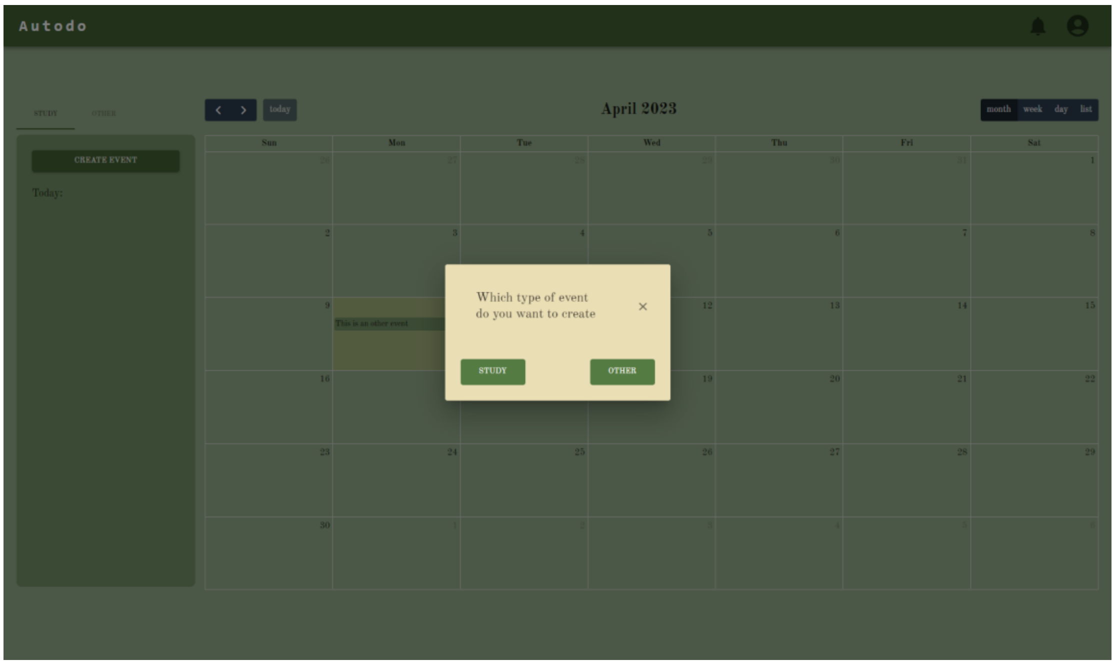
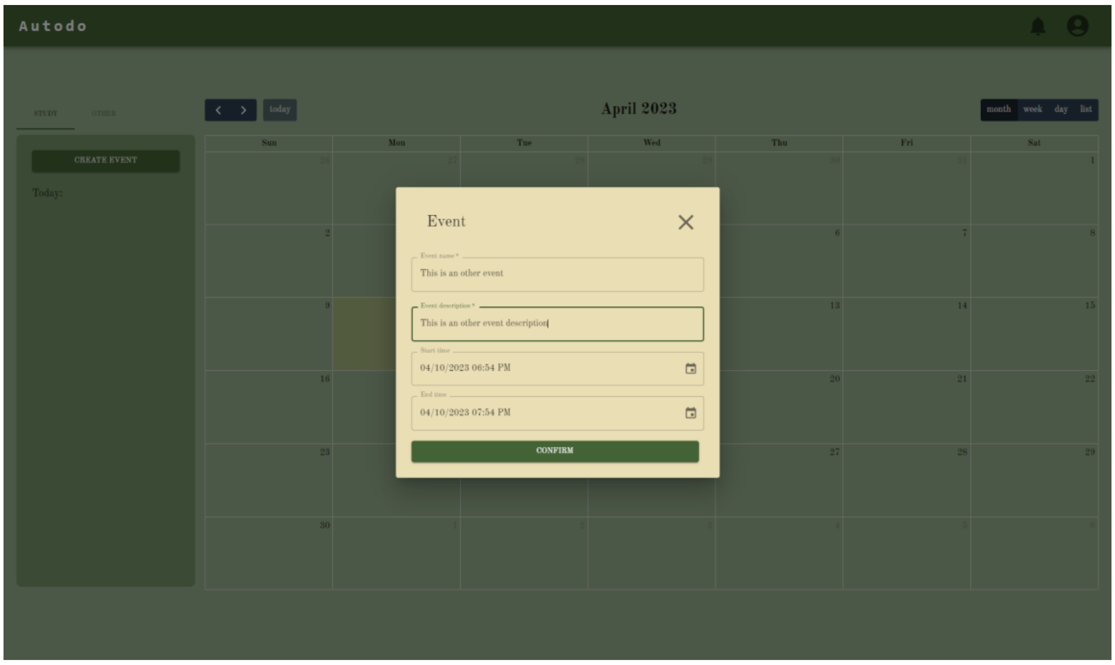


Figure 5.2.10: Main page, after clicking the "CREATEEVENT" button.

Once user click the "CREATEEVENT" button, the dialog window will ask user what type of event user wants to create. Now let's create a "other" event.

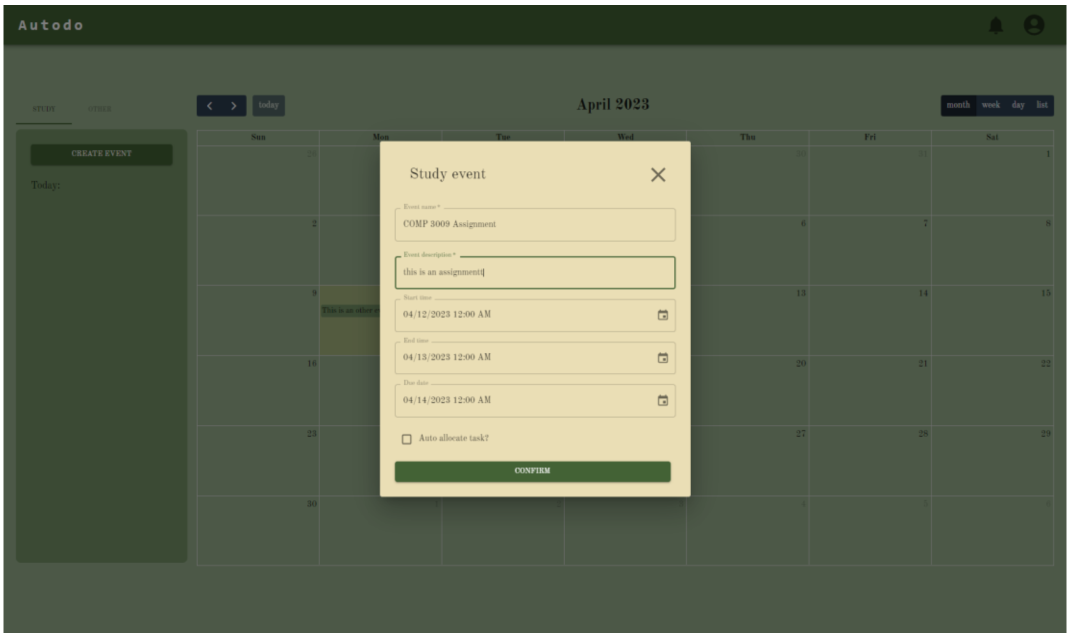
Note: user can also click a date or select a range of dates to create an event.



Event name *: This is an other event
Event description *: This is an other event description
Start time: 04/10/2023 06:54 PM
End time: 04/10/2023 07:54 PM

Figure 5.2.11: Main page, window to create "other" event.

User can specify the title, description, start, end date of the event. Click confirm button will create such an event.



Study event

Event name*: COMP 3009 Assignment

Event description*: this is an assignmentt

Start time: 04/12/2023 12:00 AM

End time: 04/13/2023 12:00 AM

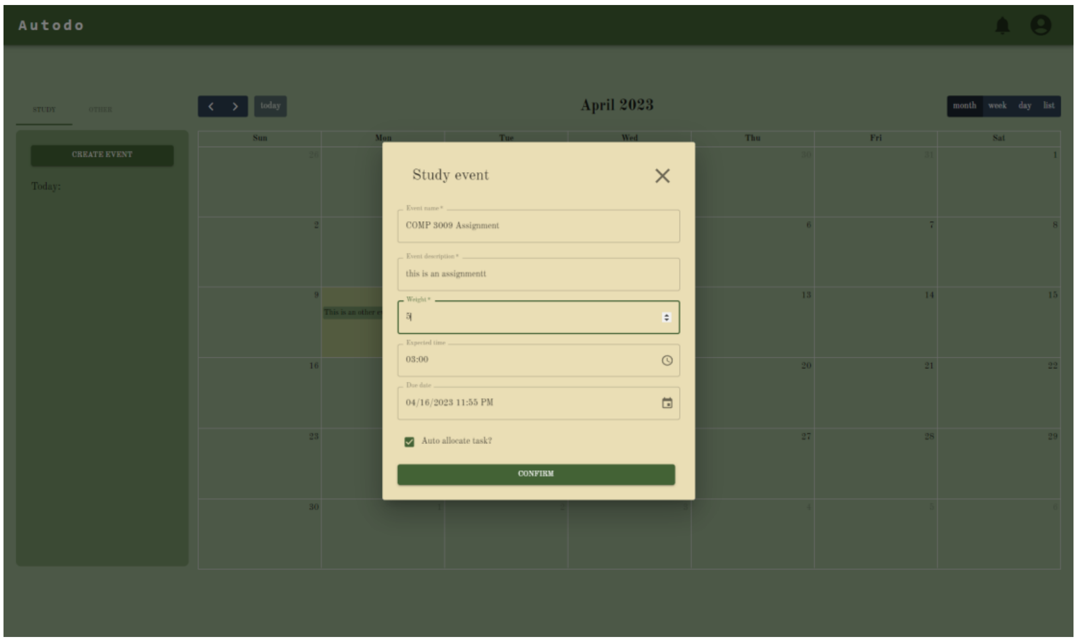
Due date: 04/14/2023 12:00 AM

☐ Auto allocate task?

CONFIRM

Figure 5.2.12: Creating study event

If user wants to create a "study" event, first thing they need to do is determine whether to use "auto" mode for that task. If the "Auto allocate task" is unselected, user just needs to specify the title, description, start time, end time, due date. And this event will not be allocated automatically.



Study event

Event name*: COMP 3009 Assignment

Event description*: this is an assignmentt

Weight*: 5

Expected time: 03:00

Due date: 04/16/2023 11:55 PM

☑ Auto allocate task?

CONFIRM

Figure 5.2.13: Creating study event with "auto" mode

If user checked the "Auto allocate task" checkbox, the start time and end time input fields will be changed to weight, expected time. Weight represents priority of the event, however, it will not be allocated first, I will just determine the percentage of time being allocated to that event if time is not enough. The expected time represents the expected time to finish that task. It must be less than 24 hours.

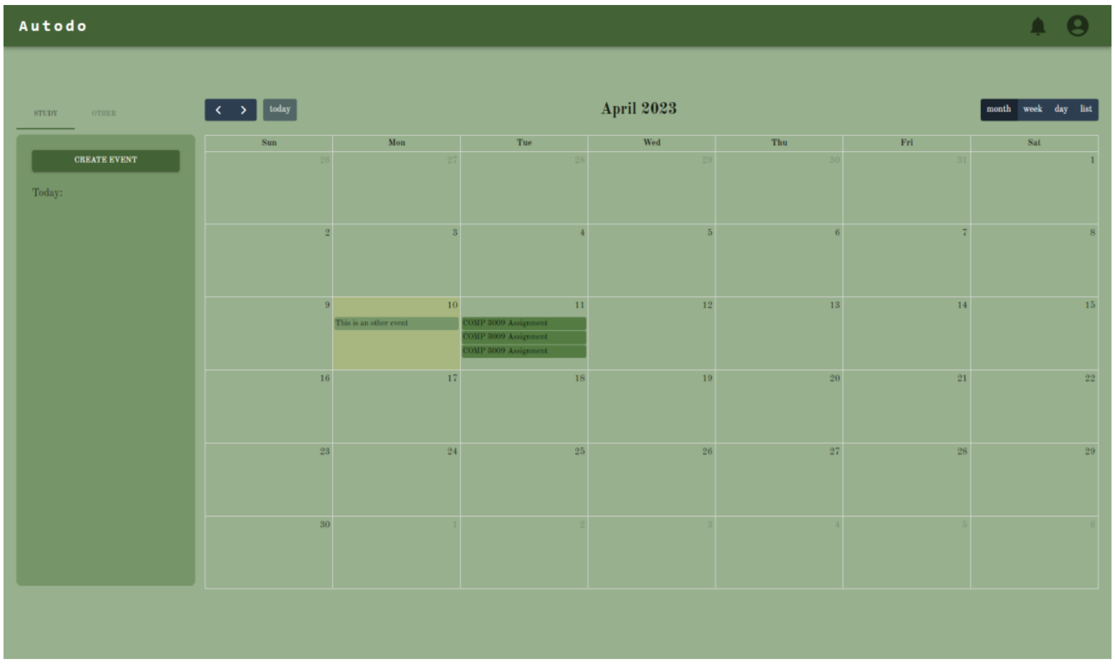


Figure 5.2.14: main page with event created

Once user click confirm button, the event "COMP3009 Assignment" with "auto" mode will be created and allocated to available time slots. Now let's switch to week view to check the time slots.

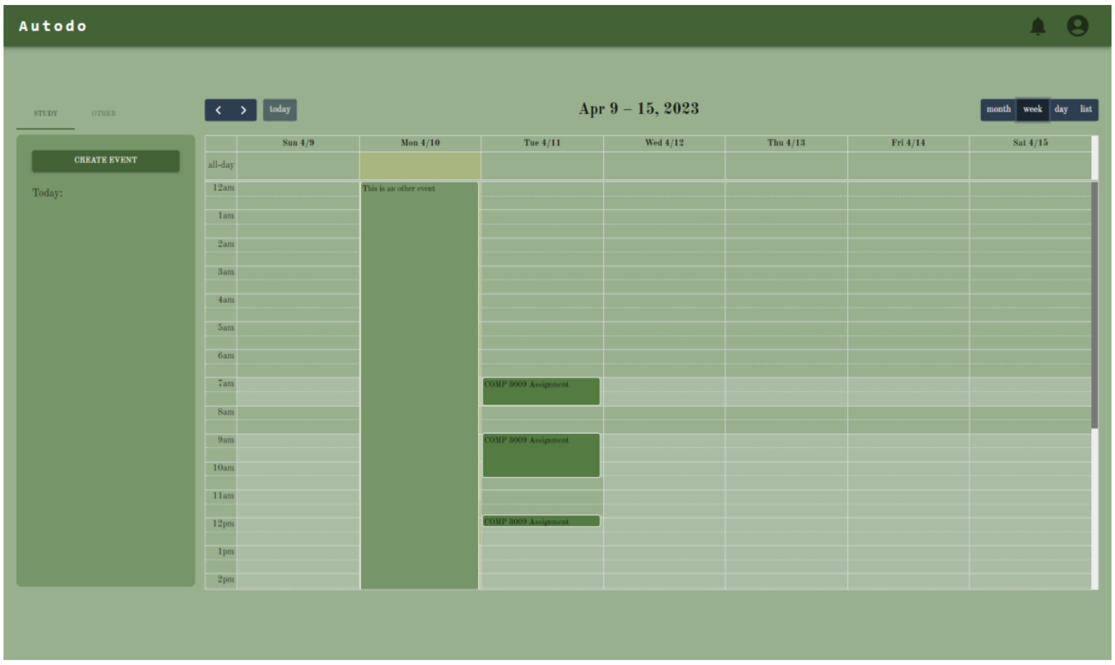


Figure 5.2.15: week view of calendar

The highlighted grid with lighter color means those time are free time slots. Thus, the study tasks will be allocated to those grids. Figure 5.2.15 shows how "COMP 3009 Assignment" gets allocated.

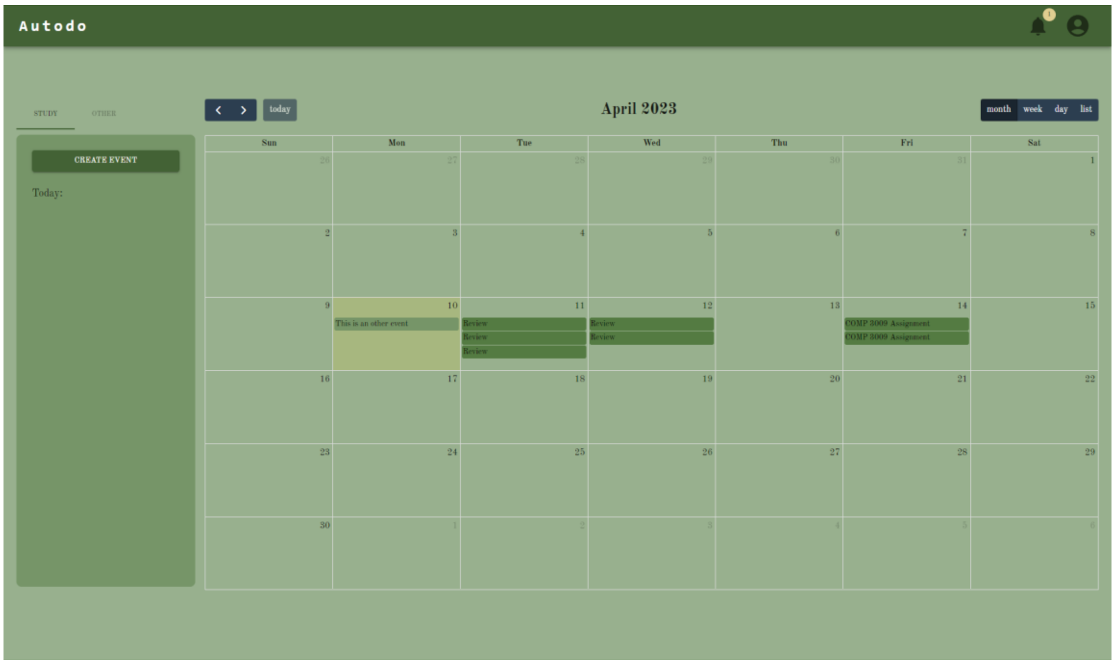


Figure 5.2.16: Events get organized in "average" mode with "skip" number of 2.

Previously, we already set the mode to "average" and "skip" number to 2. Now create another event "Review" with duration of 20 hours, and due date of March-14 00:00. And these two events get organized as shown in Figure 5.2.16. Let me to explain the logic. Review first get allocated to 11th (because this is the first day has free time slots), and it looks for next available time slots two days after 11th (i.e. 14th), however, the event due at 14th 00:00, so there is no free time slots on 14th. Then it looks for 12th free time slots and allocate the task to 12th. The "COMP 3009 Assignment" just gets assigned two days after 11th. Now let's switch to the "early "mode:

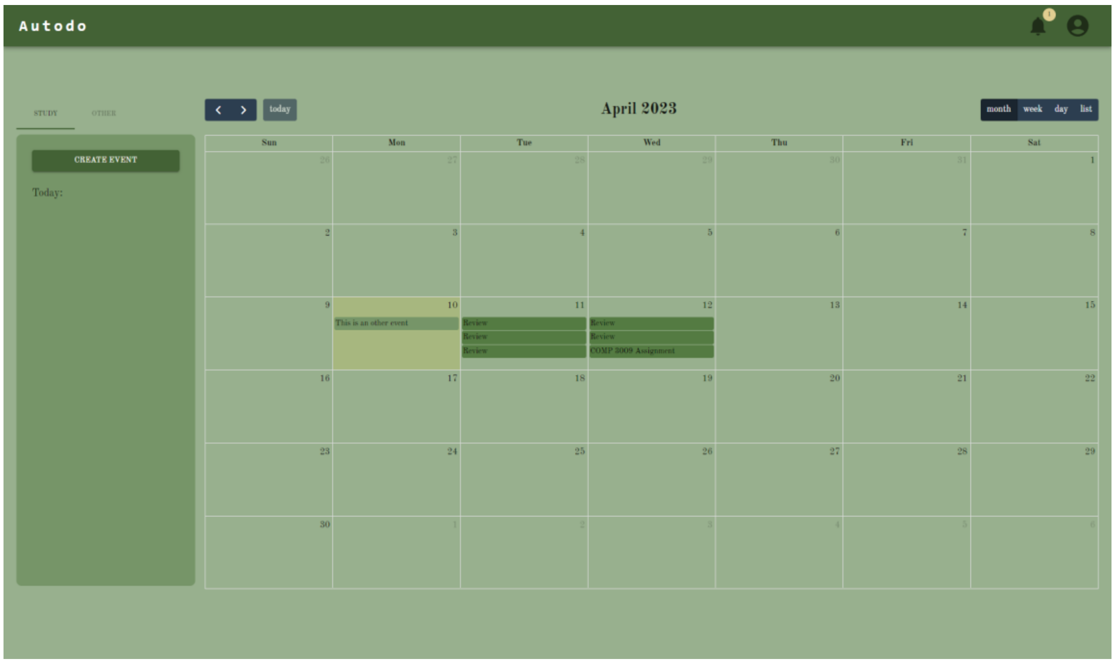


Figure 5.2.17: early mode

In this mode, events just get allocated as early as possible.

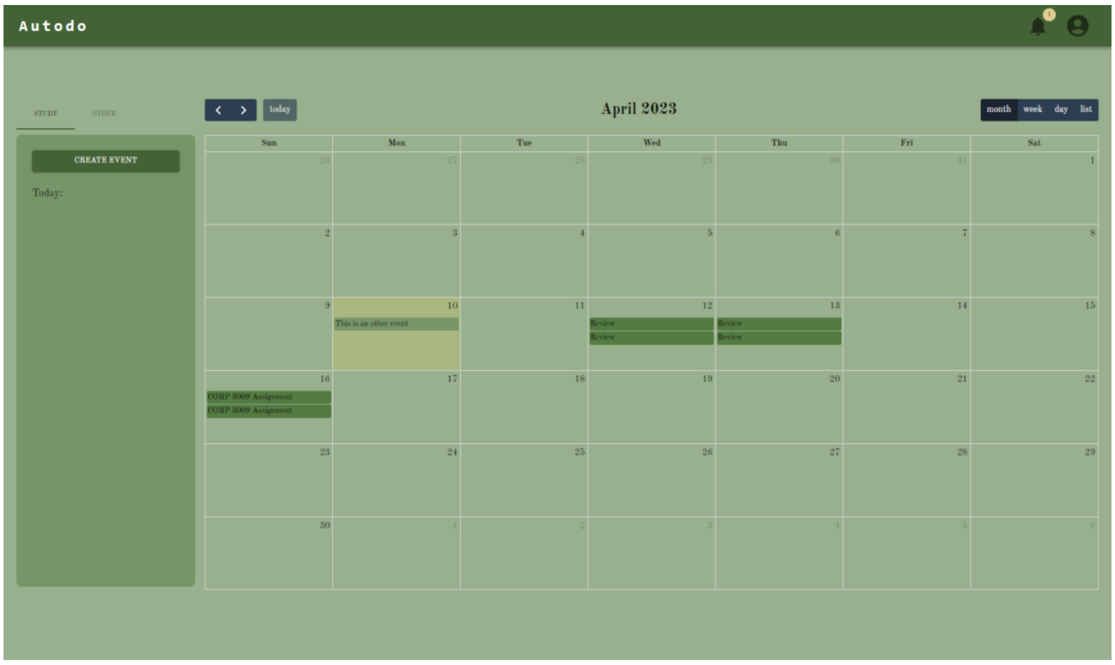
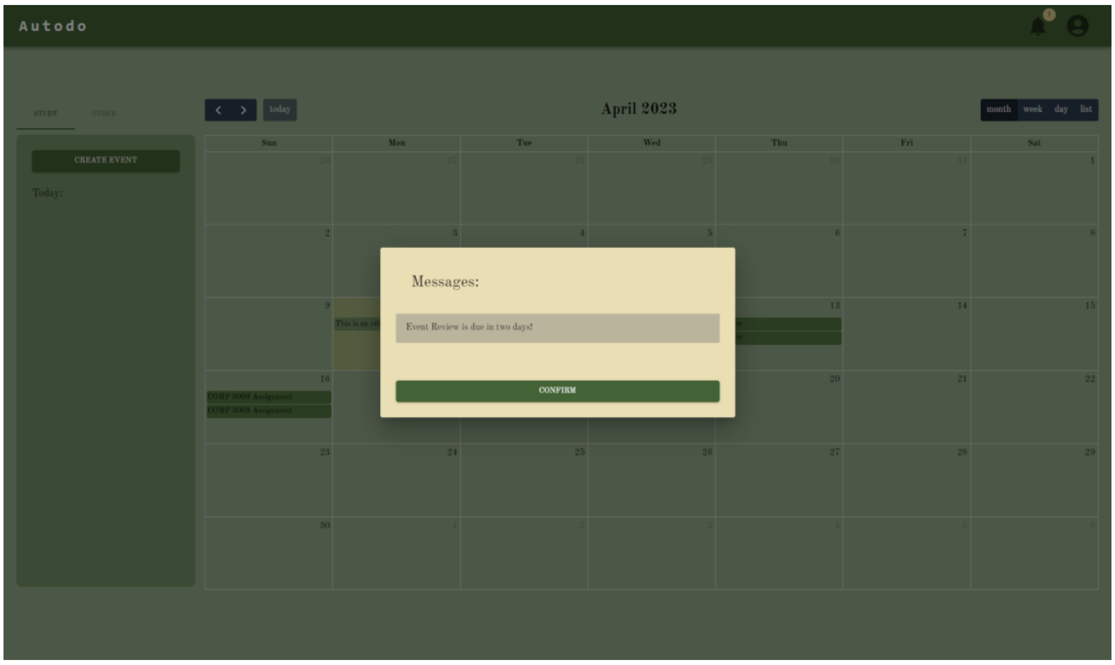


Figure 5.2.18: late mode

For "late" mode, events get organized as late as possible, however, they are still assigned to a time before the due date. Here "Review" is due at 14th 00:00, so the event is assigned to 12th and 13th, the event "COMP 3009 Assignment" is due at 16th 23:59, so the event is assigned to 16th.



Messages:

Event Review is due in two days!

Figure 5.2.19: message

Since the event "review" is due at 14th 00:00 and today is 10th, so the event notification will be added to the messages of user. User can check messages by clicking the bell icon next to profile icon.

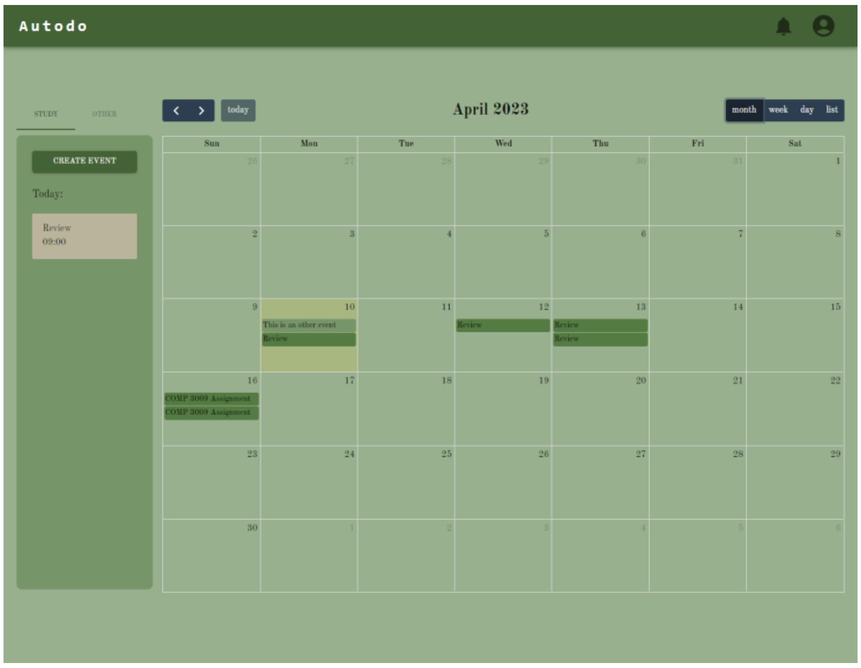
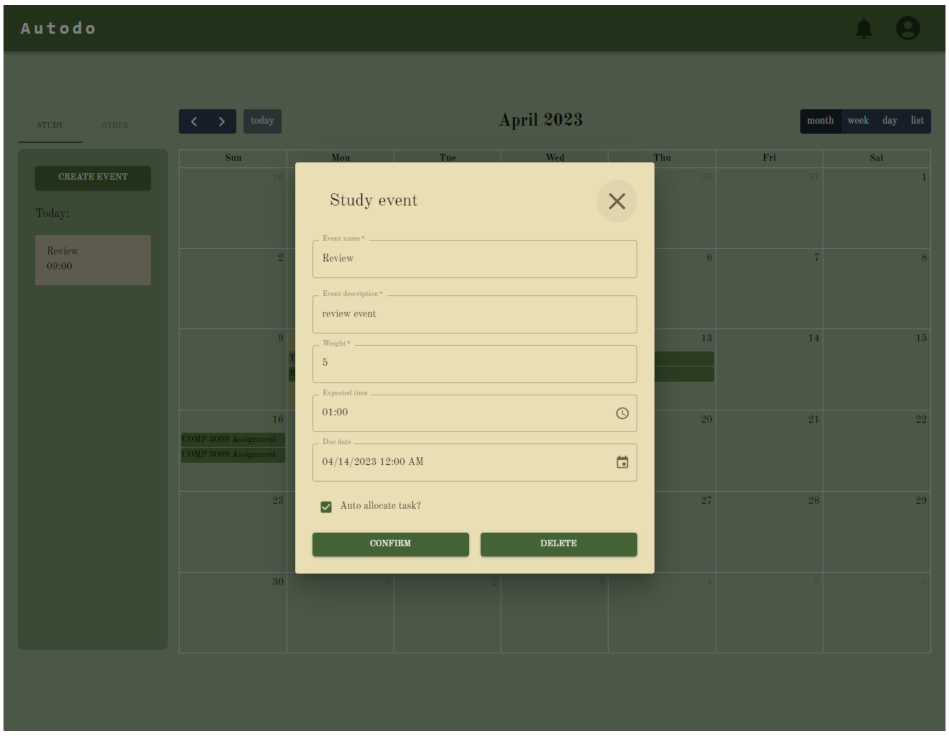


Figure 5.2.20: Left panel

User can drag and drop an event to a certain date/time. Figure 5.2.20 shows the one of the “review” events being dragged and dropped on 10th. In addition, the left panel now shows today’s study event.


Study event

Event name *: Review

Event description *: review event

Weight *: 5

Expected time: 01:00

Due date: 04/14/2023 12:00 AM

☑ Auto allocate task?

Figure 5.2.21: Event editing

Figure 5.2.21 shows a way to edit event. Click an event will generate a window like this. User can change most properties of an event. Or if user wants to delete this event, click the delete button will work. In addition, there other ways to modify event: drag and drop, resizing, etc. directly on the calendar. User can switch views to edit events differently.

5.3. Chapter Summary

In this chapter, we showed and discussed the user interface interactions, which are designed to provide a smooth and engaging experience for users. The application is designed with usability and accessibility in mind, ensuring that users can quickly learn and efficiently use the application's features. The layout is intuitive, with clear and concise labels for buttons and fields, making it easy for users to navigate the application and complete their tasks. MUI played a significant role in achieving the desired user experience and interface design, providing a comprehensive set of components and styles that made it easier to create an aesthetically pleasing and functional application.

6. Conclusion

6.1. Chapter Overview

In this concluding chapter, I will reflect on the project, discuss the lessons learned throughout the development process, and explore potential future research directions. I have successfully designed and implemented a task allocation application that leverages a custom algorithm, client-side and server-side technologies, and an efficient, user-friendly interface. Throughout this project, we have addressed various challenges and achieved an application that meets the objectives set out in the introduction.

6.2. Reflection

The motivation for this project was to address the challenges students face when managing their time and allocating tasks effectively. The goal was to develop an application that could help users optimize their time management, allowing them to allocate tasks based on their preferences, schedule, and deadlines.

Throughout the development process, various technologies were used to build the application. On the client-side, React.js was used as the primary framework, allowing for the creation of reusable components and a modular code structure. MUI played a significant role in providing a responsive and visually appealing user interface. For the server-side, Node.js and Express were employed to develop APIs and manage data persistence with a MongoDB database. The project also utilized several libraries and tools, such as FullCalendar, dayjs, and JWT, etc. to implement specific functionalities and improve the overall user experience.

The development process was a learning experience that allowed for reflection on the choices made in terms of technology, architecture, and design. As the project progressed, it became clear that there was an opportunity to expand the application's functionality and improve certain aspects. For example, the task allocation algorithm could be optimized further, and additional features, such as advanced notifications and gamification, could be implemented to enhance the user experience.

Moreover, the project highlighted the importance of considering security and data privacy. Although user password encryption was not implemented, it is essential to prioritize these aspects in future iterations of the application to ensure user trust and data protection.

Throughout the project, several challenges were faced, and overcoming them required adaptation and problem-solving. Balancing the project's scope with the available time and resources was a crucial aspect of the development process. It was important to remain flexible and open to new ideas while staying focused on the primary goals and objectives.

In conclusion, the reflection on the development process reveals the significance of effective planning, adaptation, and continuous learning. The experiences gained during this project offer valuable insights into the challenges and opportunities associated with developing a task allocation and time management application. By taking these lessons into account, future work can be approached with a more comprehensive understanding and a stronger foundation for success.

6.3. Future Research Directions

There are several potential advanced future research directions and improvements for this project. Some of these include:
Enhancing security measures: Implementing encryption for user passwords to increase data security and user privacy.
Expanding the range of supported devices and platforms: Developing native mobile applications for iOS and Android or a Progressive Web App (PWA) to reach a broader audience. In this case, the notification system will play a significant role, so it will need to be implemented more comprehensively.
Integration with external services: Enabling integration with third-party calendar and task management applications, such as Google Calendar, to provide users with a unified task management experience.
Personalization and machine learning: Implementing machine learning techniques to analyze user behavior and preferences to provide personalized task allocation suggestions.
Advanced collaboration features: Allowing users to share their tasks and schedules with other users, enabling team collaboration and group task allocation.
By considering these future research directions and potential improvements, the application can continue to evolve and provide an even more effective and engaging task allocation solution for students or even other groups.

References

[1]. Solanki, R., & Wadhwani, P. (2023, March 31). *When and why use node.js for backend development*. IT Blog | Mobile App Development India | Offshore Web Development - Bacancytechnology.com. Retrieved April 7, 2023, from https://www.bacancytechnology.com/blog/why-use-nodejs

[2]. *Why use mongodb and when to use it?* MongoDB. (n.d.). Retrieved April 9, 2023, from https://www.mongodb.com/why-use-mongodb

[3]. Kopachovets, Oleg. "8 Advantages of REACT JS: Reasons to Choose." ProCoders, March 13, 2023. https://procoders.tech/blog/advantages-of-using-reactjs/.

Appendices

A. Database Schema

```
const userSchema = new Schema({
email: String,
password: String,
courses: {
type: [Schema.Types.Mixed],
default: []
},
coursesTime: {
type: Schema.Types.Mixed,
default: {}
},
sleepTime: {
type: Schema.Types.Mixed,
default: { start: "23:30", end: "07:00" }
},
taskAssignMode: {
type: String,
default: 'early'
},
skipNumber: {
type: Number,
default: 1
},
routine: {
type: [Schema.Types.Mixed],
default: []
},
events: {
type: [Schema.Types.Mixed],
default: []
},
message: {
type: [Schema.Types.Mixed],
default: []
},
});
```

B. API Documentation

Get Users (for testing purpose)

- URL: `/login`
- Method: `GET`
- Success Response:
  - Status: `200`
  - Content: `{users: [user]}`
- Error Response:
  - Status: `400`
  - Content: `{message: string}`

Log In

- URL: `/login`
- Method: `POST`
- Body: `{`
  ```
  email: string,
  password: string
  ```
  `}`
- Success Response:
  - Status: `200`
  - Content: `{message: string, token: jwtToken}`
- Error Response:
  - Status: `400`
  - Content: `{signInError: string, message: string}`

Sign Up

- URL: `/signup`
- Method: `GET`
- Body: `{`
  ```
  email: string
  ```
  `}`
- Success Response:
  - Status: `201`
  - Content: `{message: string, token: jwtToken}`
- Error Response:
  - Status: `409`
  - Content: `{signUpError: string, message: string}`
  - Status: `500`
  - Content: `{message: string, err: string}`

Main page notification request

- URL: `/main`
- Method: `GET`
- Body: `{`
  `}`
- Success Response:
  Status: `200`
  Content: `{notifications: [string]}`
- Error Response:
  Status: `400`
  Content: `{message: string}`

Calendar data

- URL: `/calendar`
- Method: `GET`
- Body: `{`
  `email: string`
  `}`
- Success Response:
  Status: `200`
  Content: `{events: [event], businessHours: [businessHour]}`
- Error Response:
  Status: `400`
  Content: `{message: string}`

Edit page user data

- URL: `/edit`
- Method: `GET`
- Body: `{`
  `email: string`
  `}`
- Success Response:
  Status: `200`
  Content: `{user: user}`
- Error Response:
  Status: `400`
  Content: `{message: string}`

Add course

- URL: `/edit/addcourse`
- Method: `POST`
- Body: {

```
    email: string,
    course_name: string
}
```

- Success Response:
    Status: `200`
    Content: `{message: string}`
- Error Response:
    Status: `400`
    Content: `{message: string}`
    Status: `404`
    Content: `{message: string}`

Delete course

- URL: `/edit/delcourse`
- Method: DELETE
- Body: {

```
    email: string,
    course_name: string
}
```

- Success Response:
    Status: `200`
    Content: `{message: string}`
- Error Response:
    Status: `400`
    Content: `{errorMessage: string}`
    Status: `400`
    Content: `{errorMessage: string}`
    Status: `404`
    Content: `{errorMessage: string}`

Edit mode

- URL: `/edit/mode`
- Method: `POST`
- Body: {

```
    email: string,
    taskAssignMode: string,
    skipNumber: number
}
```

- Success Response:
  Status: `200`
  Content: `{message: string}`
- Error Response:
  Status: `400`
  Content: `{message: string}`
  Status: `404`
  Content: `{message: string}`

Edit sleep time

- URL: `/edit/sleeptime`
- Method: `POST`
- Body: `{`
```
    email: string,
    start: string,
    end: string
}
```
- Success Response:
  Status: `200`
  Content: `{message: string}`
- Error Response:
  Status: `400`
  Content: `{message: string}`

Add courses time

- URL: `/edit/coursesTime`
- Method: `POST`
- Body: `{`
```
    email: string,
    courseName: string,
    daysOfWeek: string,
    start: string,
    end: string
}
```
- Success Response:
  Status: `200`
  Content: `{message: string}`
- Error Response:
  Status: `400`
  Content: `{message: string}`
  Status: `400`
  Content: `{message: string}`

Add routine event

- URL: `/edit/routine`
- Method: `POST`
- Body: {
```
    email: string,
    eventName: string,
    eventDescription: string,
    day: string,
    start: string,
    end: string
}
```
- Success Response:
  Status: `200`
  Content: `{message: string}`
- Error Response:
  Status: `400`
  Content: `{message: string}`
  Status: `400`
  Content: `{message: string}`

Delete routine event

- URL: `/edit/delroutine`
- Method: `DELETE`
- Body: {
```
    email: string,
    eventName: string,
}
```
- Success Response:
  Status: `200`
  Content: `{message: string}`
- Error Response:
  Status: `400`
  Content: `{message: string}`

Confirm messages

- URL: `/main/confirmmessage`
- Method: `DELETE`
- Body: {
```
    email: string,
    confirm: boolean,
```

    }
- Success Response:
    Status: `200`
    Content: `{message: string}`
- Error Response:
    Status: `400`
    Content: `{errorMessage: string}`

Modify event
- URL: `/main/event/modify`
- Method: `POST`
- Body: {
```
    email: string,
    id: uuid,
    start: string,
    end: string,
    allDay: Boolean,
    extendedProps: extendedProps
}
```
- Success Response:
    Status: `200`
    Content: `{message: string}`
- Error Response:
    Status: `400`
    Content: `{message: string}`
    Status: `404`
    Content: `{message: string}`

Create event
- URL: `/main/event/create`
- Method: `POST`
- Body: {
```
    email: string,
    id: uuid,
    title: string,
    start: string,
    end: string,
    color: string,
    textColor: string,
    allDay: Boolean,
    extendedProps: extendedProps
}
```
- Success Response:

Status: `200`
Content: `{message: string}`

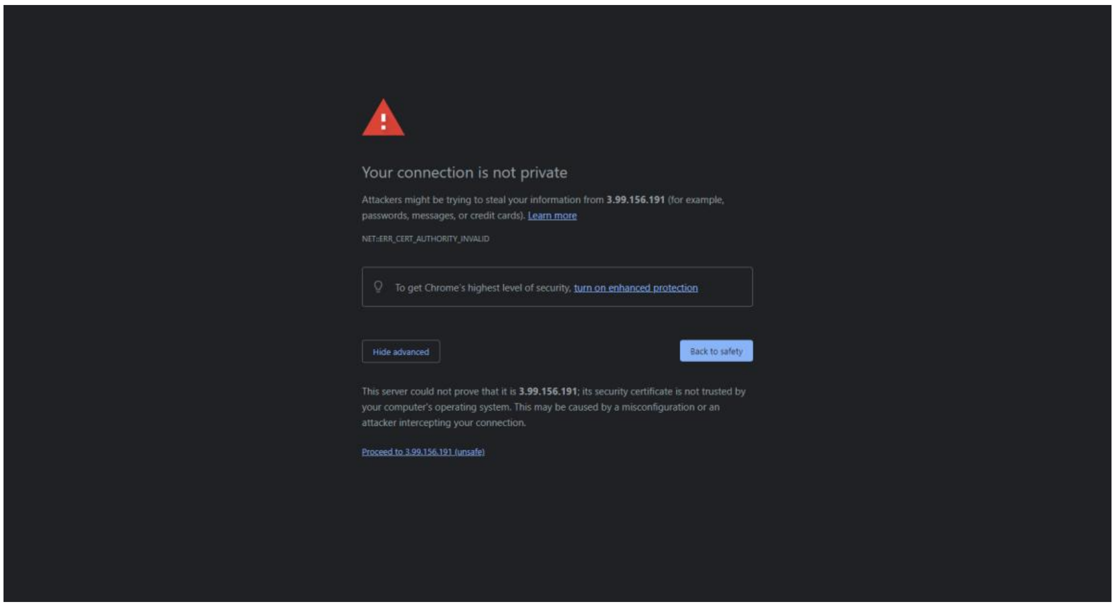
- Error Response:
  Status: `400`
  Content: `{message: string}`

Delete event

- URL: `/main/event/delete`
- Method: `DELETE`
- Body:
```
{
    email: string,
    id: uuid
}
```
- Success Response:
  Status: `200`
  Content: `{message: string}`
- Error Response:
  Status: `400`
  Content: `{message: string}`

C. Instructions to test code

Method 1: Online (Example using chrome browser, edge works too, didn't test other browsers)
Step 1: Go to https://3.99.156.191/users, click proceed t o 3.99.156.191(unsafe). The connection is not safe because the server used a self-signed SSL certificate.


Your connection is not private

Attackers might be trying to steal your information from **3.99.156.191** (for example, passwords, messages, or credit cards). Learn more

NET::ERR_CERT_AUTHORITY_INVALID

To get Chrome's highest level of security, turn on enhanced protection

This server could not prove that it is **3.99.156.191**; its security certificate is not trusted by your computer's operating system. This may be caused by a misconfiguration or an attacker intercepting your connection.

Proceed to 3.99.156.191 (unsafe)

Step2: Go to https://main.dj1lwi8sc3zm.amplifyapp.com/

Method 2.1: Locally with an old version server
Step 1: Make sure the code is prepared, and Node.js is installed on your machine.
Step 2: Go to the server folder, open a terminal, and use the command "npm install."
Step 3: Go to the client folder, open a terminal, and use the command "npm install."
Step 4: In the server terminal, use the command "npm start."
Step 5: In the client terminal, use the command "npm start."
Step 6: Go to the URL http://localhost:3000 .

Method 2.1: Locally with a database version server
Step 1: Make sure the code is prepared, and Node.js is installed on your machine.
Step 2: Go to the server folder, open a terminal, and use the command "npm install."
Step 3: Go to the client folder, open a terminal, and use the command "npm install."
Step 4: Let me add the public IP address of your testing machine to the MongoDB cloud network access whitelist.
Step 5: In the server terminal, use the command "npm start."
Step 6: In the client terminal, use the command "npm start."
Step 7: Go to the URL http://localhost:3000 .