ChronoHawk

Evan Gardner

Daniel Lawrie

Mentor: Dr. Cui Yu

Department of Computer Science & Software Engineering

Monmouth University

Fall 2023

Table Of Contents

1. Motivation

Our inspiration for this project stemmed from the challenges we faced in managing our time effectively as students. We often found ourselves rushing to complete assignments just hours before the deadline. To address this common issue, we designed a system that calculates the daily study hours a student needs before a due date. This not only prevents last-minute

cramming but also aids students in preparing for other academic tasks efficiently. By offering a system open to everyone, we aim to revolutionize the way students approach time management.

2. Related Work

2.1 Programming Languages

2.1.1 HTML & CSS

Hypertext Markup Language (HTML) is a collection of elements that dictate the layout and content structure of web pages [1]. The fundamental syntax comprises an opening bracket, an element tag, a closing bracket, followed by the content. All web pages, regardless of how they are crafted—using state-of-the-art frameworks or markdown-based ones—ultimately boil down to HTML that browsers interpret. HTML tags shape every visual aspect of a website: every piece of text, image, or box is configured using HTML. Additionally, these elements can have attributes that offer further instructions to the browser.

Attributes like 'id' and 'class' are invaluable as they differentiate elements with identical tags, making it feasible to independently edit and style them. In conjunction with HTML, there's Cascading Style Sheets (CSS), a language that outlines how the content should visually appear to users [2]. While HTML delineates what content is shown, CSS determines its aesthetic appeal. It specifies, for instance, if text appears bold or red, or the placement of a box on a page. Together, HTML and CSS create a harmonious web page experience [2]. A web page's existence relies on HTML, while CSS ensures it's aesthetically pleasing.

CSS employs a combination of selectors and styling rules. Selectors pinpoint which elements are impacted by the subsequent styles. These selectors can target tag names, classes, IDs, or even specific relationships like adjacent siblings or direct descendants [2]. After designating the selectors, styles are applied. These styles can range from basic positioning and

colors to intricate transformations like scaling or shifting [2]. Notably, elements can carry 'class' or 'id' attributes, enabling CSS selectors to style them distinctively.

2.1.2 SQL

Structured Query Language (SQL) is a domain-specific language used in programming and managing relational databases [20]. Initially developed in the 1970s by IBM, SQL has become an indispensable tool in data manipulation and retrieval [20]. It enables users to insert, query, update, and delete data, as well as create and modify schemas and control access to data [20]. Various forms of SQL are used in many of the world's most popular database systems, such as Oracle, Microsoft SQL Server, and MySQL, showcasing its widespread applicability [20]. One of SQL's most notable features is its ability to manage large sets of structured data and conduct intricate operations, like joining data from multiple tables or aggregating data based on specific conditions [20]. Given its robust capabilities and standardization by the American National Standards Institute (ANSI), SQL remains a pivotal tool for data analysts, database administrators, and back-end developers who aim to efficiently harness the potential of relational databases [20].

2.1.3 Typescript & Javascript

While HTML and CSS lay the foundation for the static layout and design of a webpage, JavaScript (JS) introduces dynamic interactivity to these elements [3]. In essence, JS infuses life into a web page, enabling actions like button clicks, form submissions, and dynamic content updates. Predominantly, JS finds its application in web browsers; however, its versatility extends to non-browser environments like Node.js, Apache CouchDB, and even Adobe Acrobat [3]. At its core, JS is a prototype-based, single-threaded language, embracing a range of coding paradigms from object-oriented to imperative to declarative [3]. It's imperative to delineate the

distinction between Java and JavaScript, with the primary differentiator being that Java undergoes compilation, while JavaScript is interpretative in nature.

JavaScript's dynamic typing offers flexibility, especially for novices, by determining types during runtime. This adaptability can streamline the initial learning curve. However, as applications scale, the allure of strong typing becomes evident. The interpretative nature of JS means that type-related inconsistencies are identified only at runtime, potentially elongating the development cycle as developers grapple with type errors during testing phases rather than code development. Herein lies the value proposition of TypeScript (TS). Born from JavaScript, TypeScript integrates strong typing, augmenting the development experience [4]. It not only understands the nuances of JS but also boasts type inference capabilities, ensuring a more seamless and error-minimized coding process [4]. With TypeScript, variables are endowed with either inferred or explicit types, facilitating real-time error detection during the code development phase, long before the code undergoes compilation to its JS counterpart.

2.2 Development Environment

2.2.1 Visual Studio Code

The plethora of development environments available offers a broad spectrum of features, but based on comprehensive analysis and hands-on experience, Visual Studio Code (VS Code) emerges as a premier choice for this project's application development. From the onset, VS Code provides intrinsic support for pivotal web development languages: JavaScript, TypeScript, CSS, and HTML [5]. Such innate support sets it apart from various contemporaries which may offer limited language compatibility. Beyond its core offerings, VS Code's modular architecture is laudable, granting users the ability to augment its capabilities via "extensions"[6]. This extensibility ensures that VS Code can readily adapt to accommodate languages like C++, C#,

Java, Python, and SQL, among others [6]. A distinguishing feature of VS Code is its minimalist design philosophy, ensuring a clutter-free workspace tailored to a developer's unique needs [6].

2.2.2 Github

Embedded within VS Code is an integrated source control mechanism, prominently featuring Git [7]. To ease the transition for users, the VS Code documentation offers valuable resources such as Git "cheat sheets" and instructive videos [7]. Git's inherent strength, especially in fostering collaboration, becomes evident between timestamps 2:45 and 3:00 in the mentioned video, where the emphasis is laid on simultaneous collaboration on common files [8]. Furthermore, between 04:55 and 05:03, Git is acknowledged for facilitating a streamlined workflow for collaborative file modifications [8]. For the project at hand, harnessing Git's capabilities for collaborative version control becomes indispensable. The seamless integration of Git within VS Code simplifies the version control process, offering functionalities like repository creation, branching, file change tracking, and GitHub publishing, as highlighted between 0:00 and 0:20 in an informative video by Microsoft's VS Code team [9]. Overall, the amalgamation of Git within VS Code's ecosystem offers an intuitive platform for tracking codebase changes and fostering team collaboration.

2.2.3 Microsoft Live Share

Supplementing VS Code's collaborative suite is the "Live Share" feature, fostering real-time collaborative coding sessions [10]. Live Share transcends traditional screen-sharing by permitting simultaneous code modifications, thus mirroring a virtual pair programming experience. Notably, debugging sessions can be co-shared, ensuring that team members can collectively address code anomalies. A remarkable attribute of Live Share is its ability to initiate collaborative sessions devoid of any prerequisite software setups or repository cloning, thereby

fast-tracking development cycles [10]. With "Live Share" complementing Git, the collaboration narrative within the development team is poised to reach unprecedented heights.

2.3 Database Technologies

2.3.1 Supabase

Supabase stands as an exemplary open-source relational database system, positioning itself as a sophisticated alternative to Firebase in the realm of advanced web development [19]. Its defining feature is the real-time database capability, offering developers an unparalleled advantage in tracking database alterations, thus ensuring seamless synchronization between the web application and the underlying database [19]. The architectural design of Supabase incorporates a state-of-the-art authentication and authorization framework, facilitating a harmonious integration into web-based projects, promoting a unified development methodology [19]. Furthermore, Supabase demonstrates remarkable versatility in its capacity to assimilate third-party tools; notable integrations include esteemed platforms such as GitHub and StackOverflow, fortified by intricate access controls [19]. Committing to the highest standards of security, Supabase mandates that access privileges are strictly reserved for verified users and authenticated providers [19]. One of its most commendable innovations is the provision of auto-generated APIs, tailored based on bespoke database schemas. This innovation not only expedites backend development processes but also substantially reduces the need for labor-intensive backend coding [19]. Supabase's client libraries have been meticulously developed to ensure compatibility across a wide array of frontend architectures, emphasizing its commitment to universal adaptability [19]. Efficient management and storage of files and multimedia assets is another prowess, with Supabase offering scalable, high-performance storage solutions tailored for contemporary web applications [19]. As a testament to its forward-thinking

approach, this continually evolving open-source relational database platform is enriched with regular updates, bolstered by exhaustive documentation, and is endorsed by a robust and supportive community [19].

2.4 Graphic Design Tools

2.4.1 Adobe Illustrator

Adobe Illustrator is prominently recognized as a top-tier graphic design software, replete with a vast array of features tailored for intricate illustration and design tasks integral to routine workflows [15, 16 1:59:58]. At the heart of its prowess lies the powerful vector drawing capabilities, ensuring graphics maintain their pristine quality regardless of their scaling, be it for minuscule website favicons or expansive billboard advertisements [15, 16 1:20:02]. Vectors utilize mathematical formulas and sequences of commands to produce shapes and designs. Whether in two-dimensional space or three-dimensional realms, the precision of vectors is unparalleled [15, 16 1:20:02]. This distinct methodology contrasts sharply with pixel-centric images, where a canvas of pixels is colored to manifest images. Such images, when resized, often suffer from pixelation, while vectors remain crisp [15, 16 20:14]. Facilitating the creation of these vector masterpieces, Illustrator offers an assortment of tools. The precision-focused pen tool, coupled with the diverse range of geometric shape tools, which can be harmoniously combined using the shape builder tool, stands testament to Illustrator's dedication to design perfection [15, 16 2:16:24]. Moreover, Illustrator's expansive color palette not only introduces depth and flair to graphics but smartly classifies colors based on their intended deployment—CMYB for tangible print outputs and RGB for digital displays [15, 16 1:20:02]. Staying at the forefront of innovation, Illustrator's recent integration of the generative recolor tool, leveraging machine learning algorithms akin to its counterpart, Photoshop, exemplifies

adaptability. Rather than merely tweaking image compositions, this feature spawns novel color palette prompts, which can be deftly applied to various graphic elements [15, 16 1:20:02]. Layer management in Illustrator resonates with the intricate layering system found in Photoshop, allowing users to meticulously organize and synergize various illustration elements into a harmonious composition [15, 16 1:10:45]. Illustrator's vast reservoir of brushes, patterns, and effects, further complemented by its trace functions, empowers designers to actualize their creative visions with vibrant precision [15, 16 4:40:35]. Rounding off its capabilities, the software's adaptability with an array of file formats ensures a seamless design experience, making it an indispensable asset in the graphic design arsenal [15, 16 1:59:58].

2.4.2 Midjourney

Midjourney emerges as a potent Discord extension, harnessing the might of sophisticated machine learning, especially large language and diffusion models, to craft intricate visuals tailored to individual preferences [17, 18]. Serving a broad spectrum of applications, from website aesthetics and professional graphics to personal entertainment and idea stimulation, Midjourney sits at the nexus of innovation and creativity [17, 18]. Distinct from its peers, this platform boasts support for cutting-edge diffusion models, notably DALL-E and Stable Diffusion [17, 18]. The beauty of Midjourney lies in its evolving nature; as more interactions occur, the AI refines itself, optimizing to more closely match users' aspirations [17, 18]. Whether users input a succinct or an elaborate text description via the /imagine command, the system meticulously sifts through related images, culminating in a visual that resonates with the user's initial vision [17, 18]. However, the prowess of Midjourney isn't confined to just one feature. The /blend function merges images into a singular, cohesive visual, while /describe reverses the /imagine process, extracting a description from a provided graphic [17, 18]. For users seeking an elevated

customization experience, Midjourney presents advanced prompts and controls, allowing for alterations in scale, detail enhancement, or even selective removal of specific visual elements [17, 18]. The recent unveiling of Midjourney V5.2 has further enriched the user experience, introducing the "weird" parameter, which infuses visuals with a distinctive, avant-garde flair [17, 18]. Another noteworthy addition is the Permutation Prompt, which facilitates the generation of varied visual outcomes by delineating options with commas within curly braces, presenting users with a bouquet of possibilities [17, 18].

2.5 Web Framework/ Component Library

2.5.1 Next.JS / React

React.js, as a client-side web framework, revolutionizes user interface development by prioritizing components over standard HTML elements [11]. At their essence, these components still rely on foundational HTML structures. Yet, with JavaScript's capabilities, they can be efficiently returned via JS functions, fostering the creation of reusable HTML blocks, termed "components." Additionally, React's conditional rendering capability is derived from standard JS conditional statements [11]. One of React's standout features is its adeptness in integrating dynamic data into static websites through APIs, facilitating data exchange with backend services. This process allows inline insertion of data into HTML, ensuring a smooth transition. Moreover, React's extensive hooks cater to state management, asynchronous rendering, memoization, and contexts, among others [11]. These hooks facilitate consistent data flow throughout applications, maintaining optimal performance. However, React's intrinsic limitation lies in its complete client-side orientation. All functionalities are executed within the browser, excluding server involvement, which, while not undermining performance, poses significant discoverability challenges. Search engine giants like Google and Bing rely heavily on server-rendered HTML to

extract vital keywords and enhance search outcomes. Given React's architecture, its content remains opaque to these search engine crawlers, potentially hindering businesses vying for prominent search engine placements. In response, Next.js emerges as a beacon, offering server-rendered architecture for React components [12]. Although both frameworks share vast syntactical similarities, Next.js's distinguishing feature is its provision for server-rendered components. This capability transforms applications from merely front-end to comprehensive full-stack entities [12]. With server-side component creation and client-side dispatching, search engine crawlers can now efficiently index web content. While browser-centric tasks remain consistent, Next.js enables seamless database data integrations, obviating the erstwhile necessity for intermediary API calls [12]. Additionally, Next.js optimizes the web development trajectory in numerous ways. A prime example is its intuitive route definition process, wherein directories and file architectures dictate routes. The name of a directory inherently translates to its corresponding route [12]. For instance, a directory structure like "about > services > page.tsx" would naturally result in a "/about/services" route. Next.js also embraces dynamic routes, accommodating unique identifiers like product IDs or usernames to tailor content delivery. Notably, caching is a cornerstone of Next.js [12]. When leveraging the browser fetch API for data retrieval, Next.js inherently caches the response, significantly expediting any follow-up requests. Moreover, developers wield comprehensive control over cached data revalidation, deciding between timed or manual revalidation [12]. This superior caching mechanism greatly amplifies performance metrics. For styling connoisseurs, Next.js incorporates innate support for CSS modules [12]. Direct CSS file imports into component files are possible, and with React's harmony with classes and IDs—similar to traditional HTML—styling remains a familiar

endeavor. Alternatively, the 'style' attribute in components offers a CSS-in-JS approach, adhering closely to conventional CSS paradigms [12].

2.5.2 Next.UI

To begin with, NextUI stands out as a commendable UI choice for the project in question. Serving as a React-based library, NextUI empowers developers to craft user interfaces that are not just appealing but also accessible. Built upon the sturdy foundations of Tailwind CSS and React, it showcases its efficiency and versatility [13]. One notable feature of NextUI is its ability to seamlessly integrate Tailwind CSS classes within its components, ensuring that the resultant CSS remains optimally compact [13]. In contrast to Tailwind CSS, which lacks React-centric components and the associated logic, props, accessibility features, and compositional elements, NextUI fills these gaps proficiently [13]. Moreover, the embrace of TypeScript in the architecture of NextUI is not merely superficial—full-fledged support ensures developers can harness TypeScript's benefits to the fullest [13]. Although intricately intertwined with React, NextUI doesn't confine its usability strictly to this framework [13]. Developers could harness the stylistic attributes of NextUI components across diverse frameworks and libraries, enhancing its versatility [13]. Animations within NextUI deserve a special mention, given their intricate nature. Leveraging Framer Motion, these animations are not only complex but also streamlined, offering developers a more intuitive approach to integrating and handling them [13]. One of the standout characteristics of NextUI is its modular design ethos for its components [13]. By ensuring that each component functions as an independent module, it bestows developers with the freedom to pick, choose, and integrate only those elements they deem necessary. The tangible benefits? More nimble applications characterized by swifter load times [13]. NextUI's API deserves applause for its uniformity across all components [13]. Shared attributes and

functionalities across distinct components not only simplify the learning curve for developers but also promote a sense of predictability when interacting with them. Adding to this flexibility, the inclusion of slots in NextUI components allows developers to incorporate custom styles or content with pinpoint precision within a component's specific areas [13].

2.5.3 Tailwind CSS

Tailwind CSS has emerged as a prominent UI framework since its inception in May 2019. Designed as a cutting-edge front-end CSS framework, it has now reached version 2.2, boasting an impressive adoption rate with over 260,000 developers incorporating it into their applications [14]. One of the distinctive features of Tailwind CSS is its departure from the traditional default themes that are commonplace in most other CSS frameworks. Its prowess lies in its unparalleled speed when it comes to styling HTML [14]. By providing an extensive array of built-in classes, Tailwind CSS empowers developers to seamlessly style elements without starting from the ground up [14]. Exclusively tailored for the front-end, Tailwind excels in facilitating the creation of responsive web app themes. An added advantage is its ability to purge redundant CSS classes, enabling developers to maintain a streamlined CSS [14]. With Tailwind, there's minimal necessity for manual rule-writing, which can be especially advantageous for those less versed in CSS [14]. Consequently, this results in an integration of style rules directly within the HTML files [14]. In essence, Tailwind CSS stands as an optimal choice for developers aiming for premium styling solutions without delving deep into manual coding processes.

2.5.4 FullCalendar.io

FullCalendar.io stands as a paramount tool in the realm of web application development, particularly for projects necessitating calendar functionalities. Introduced as a jQuery plugin, FullCalendar.io has evolved to be compatible with various JavaScript frameworks, making it a

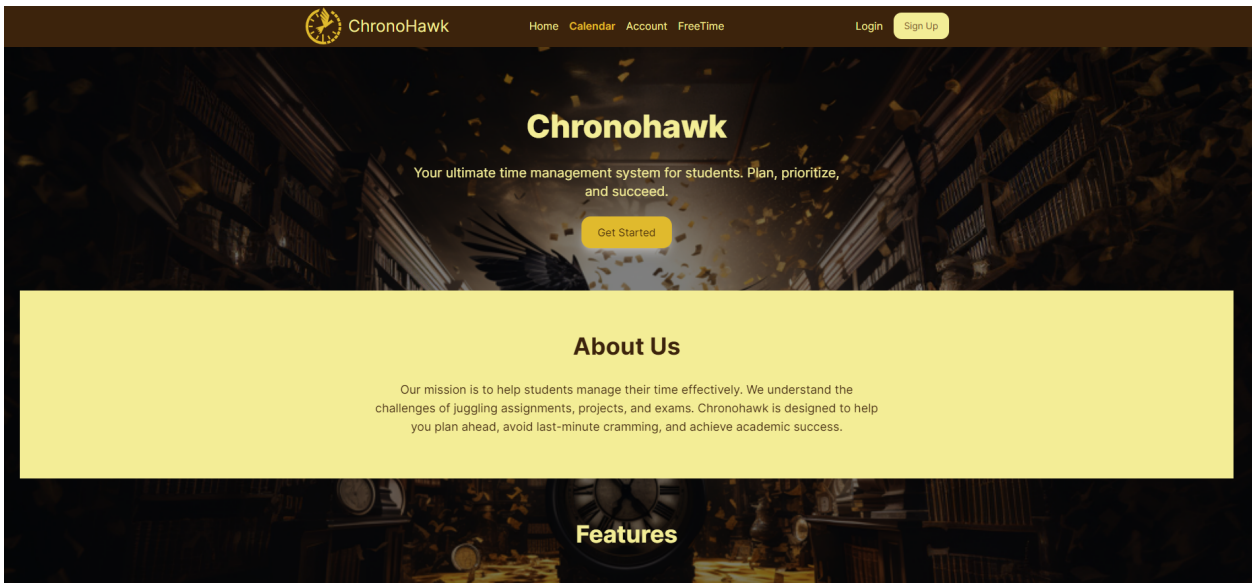
versatile option for developers [21]. Its core strength lies in its adaptability, allowing it to render both basic views, such as month, week, and day formats, and more intricate views, including resource timelines and custom views. Beyond its display capabilities, FullCalendar.io supports a multitude of interactions, ranging from event dragging and resizing to time grid selection, thus providing developers with a robust platform to craft dynamic scheduling applications [21]. One of its distinguishing features is its ability to fetch event data from external sources, which can be seamlessly integrated using JSON feeds or other formats. This ensures real-time event updates and enhances user interactivity. Furthermore, FullCalendar.io's extensive documentation and active community support bolster its appeal, ensuring that developers can swiftly troubleshoot issues and customize the tool to fit unique project requirements [21]. As modern web applications gravitate towards enhanced user experiences, tools like FullCalendar.io play a pivotal role in shaping interactive and functional user interfaces.

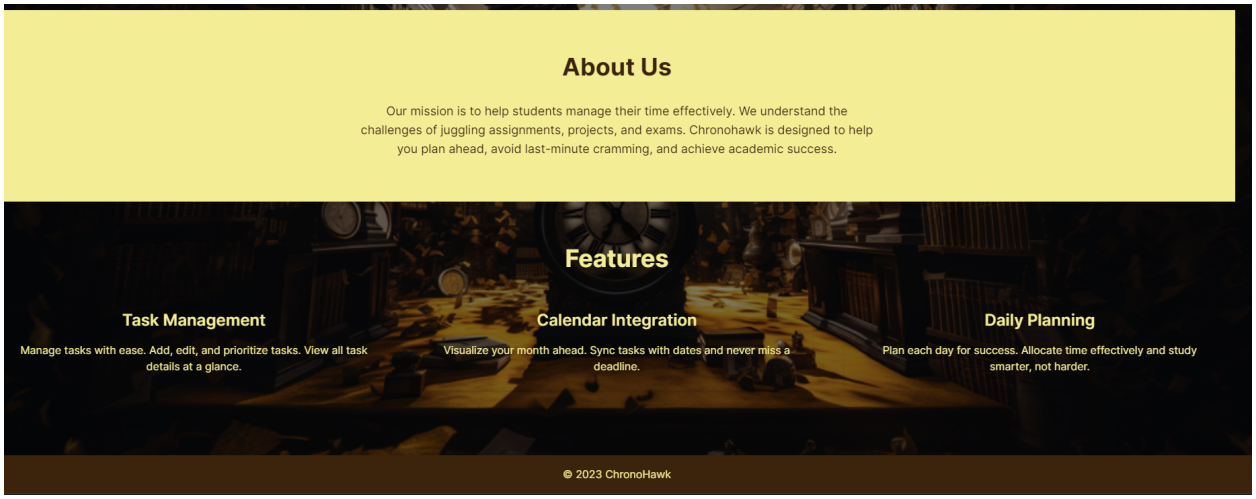
3. Design and Implementation

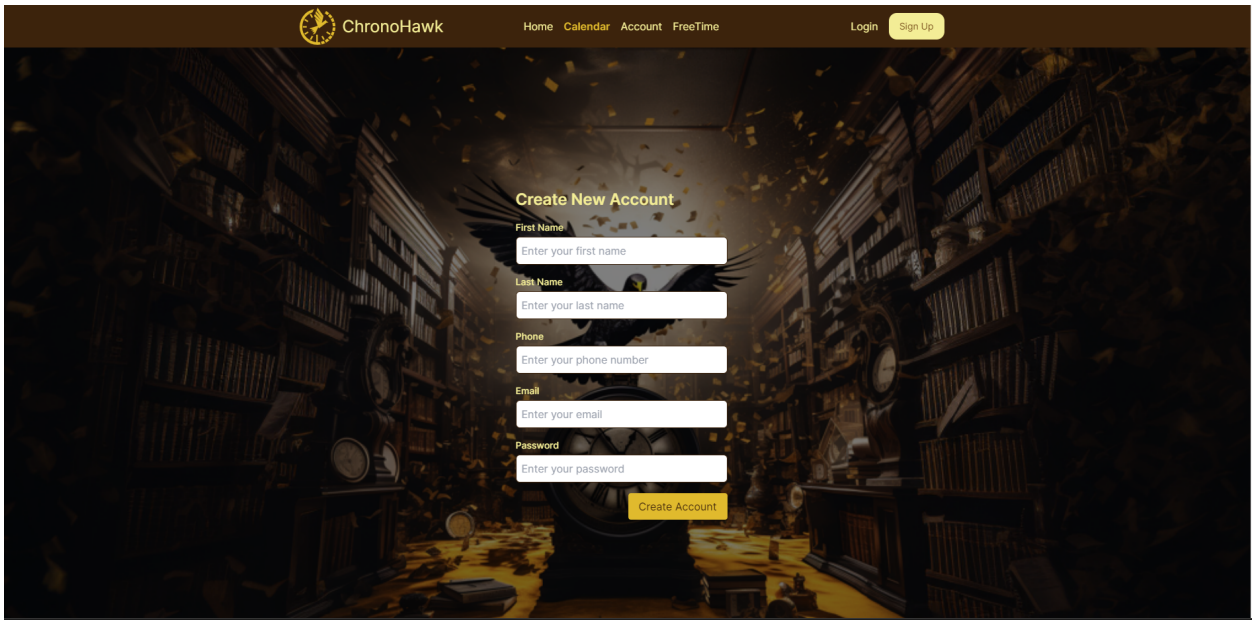
3.1 Logo/ Favicon

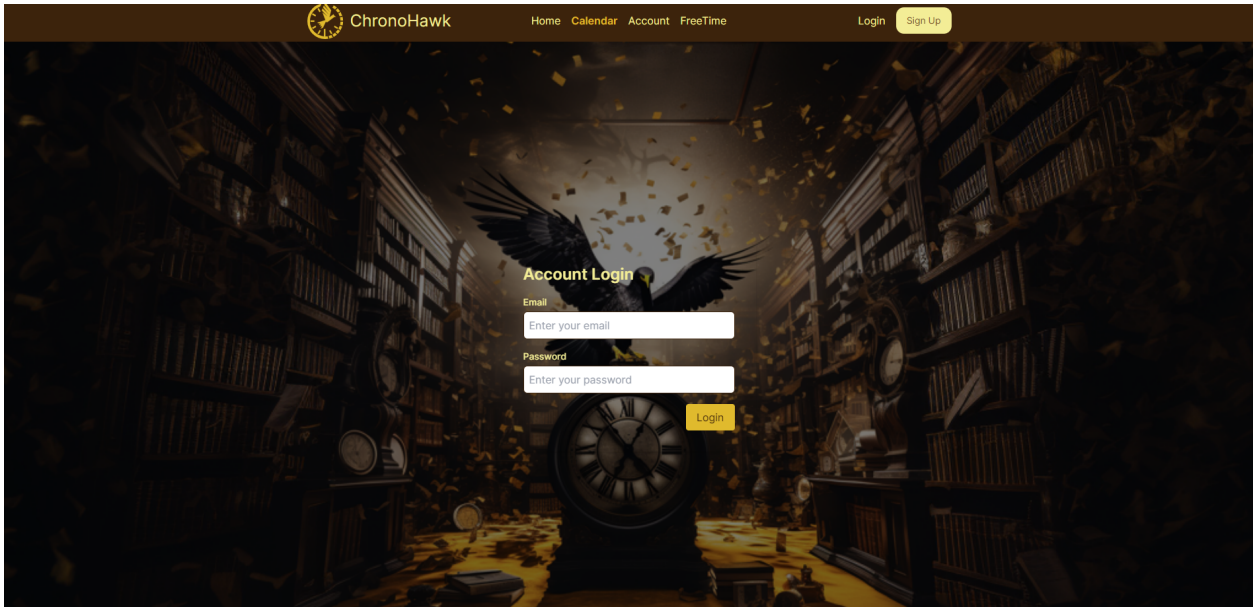
3.2 Background Images

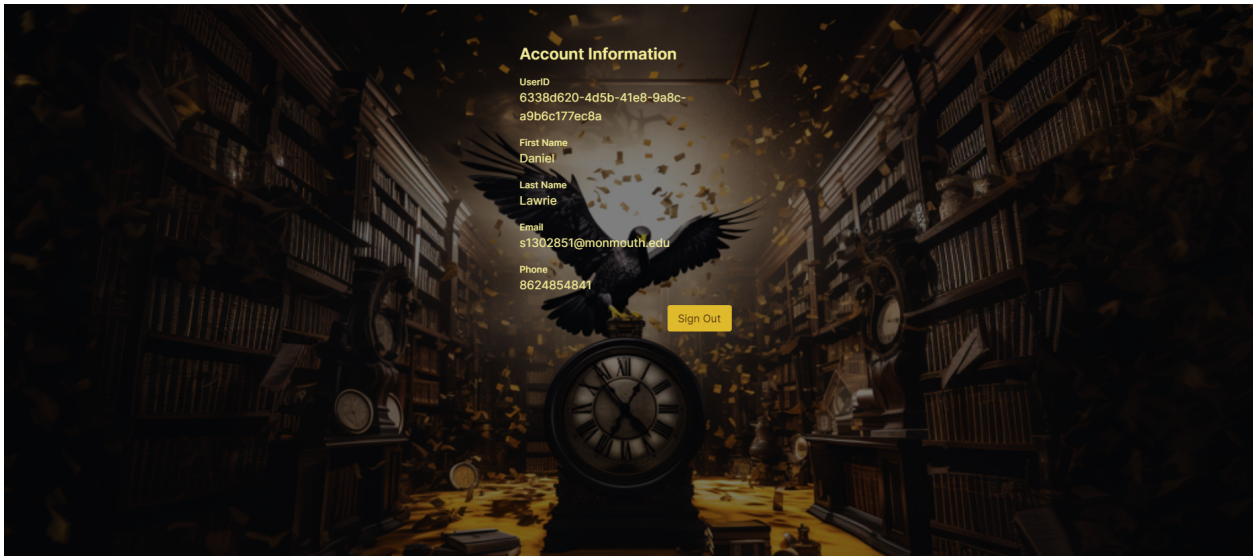
3.3 User Interface Design





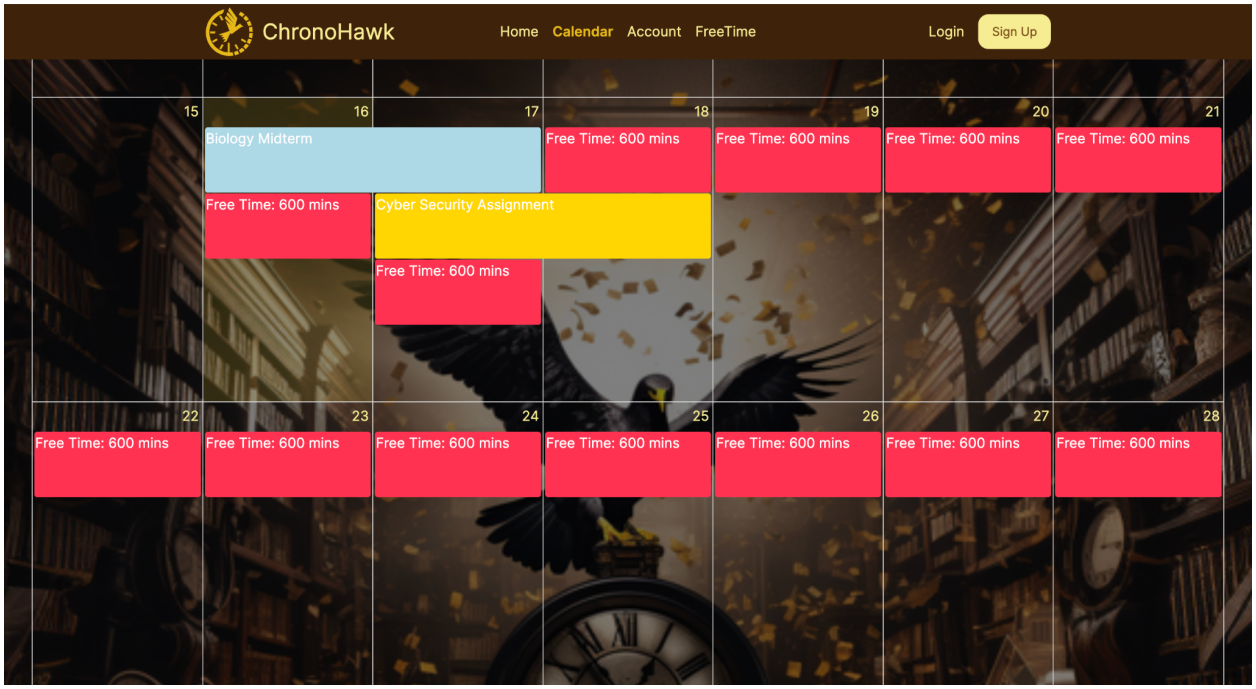

ChronoHawk
Home
Calendar
Account
FreeTime
Login
Sign Up
Create New Account
First Name
Enter your first name
Last Name
Enter your last name
Phone
Enter your phone number
Email
Enter your email
Password
Enter your password
Create Account

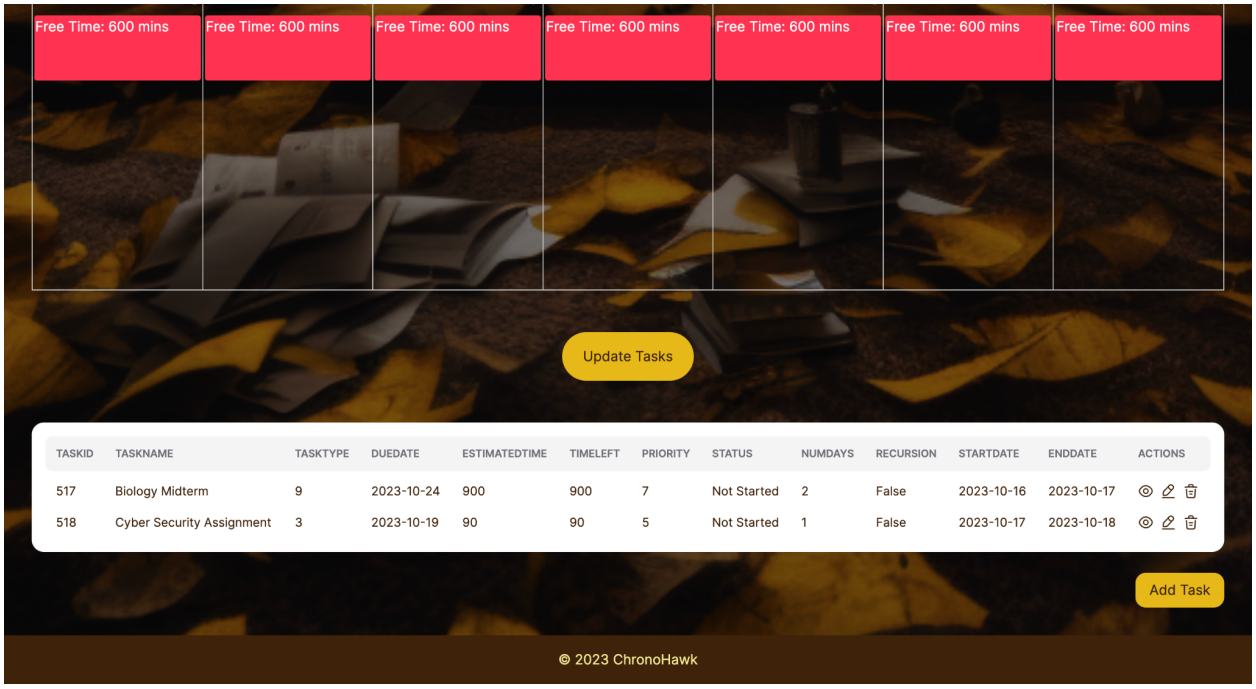

ChronoHawk
Home
Calendar
Account
FreeTime
Login
Sign Up
Account Login
Email
Enter your email
Password
Enter your password
Login

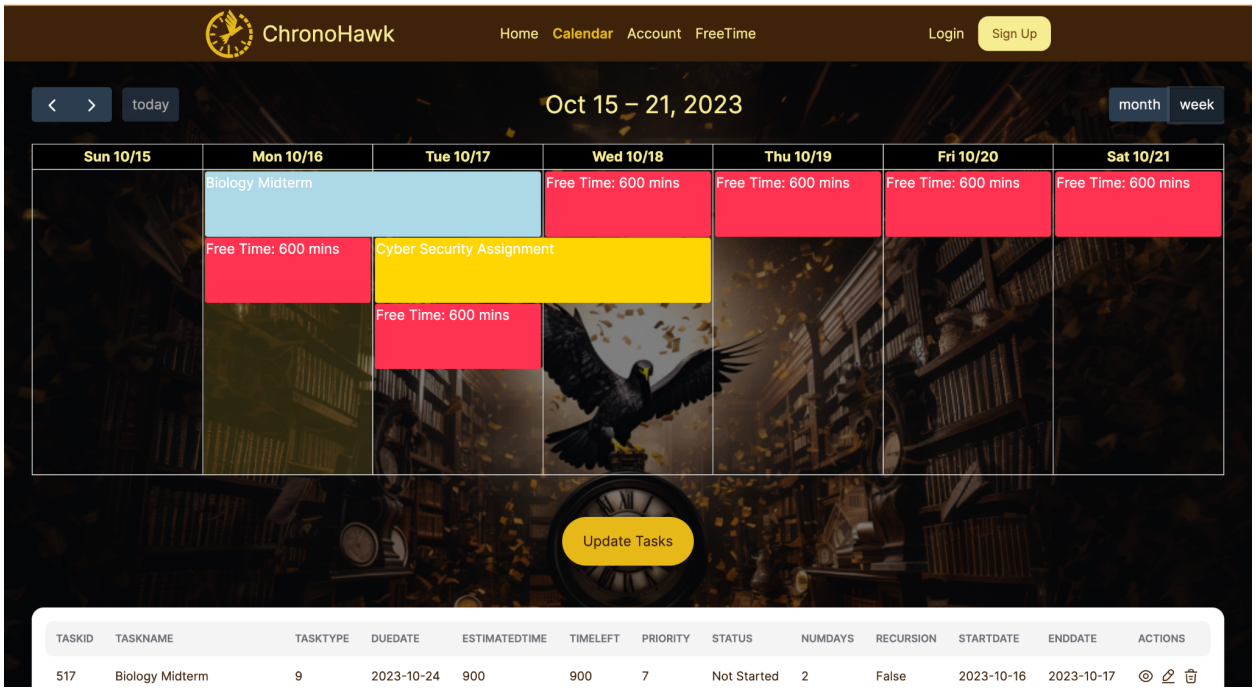

Account Information
UserID
6338d620-4d5b-41e8-9a8c-a9b6c177ec8a
First Name
Daniel
Last Name
Lawrie
Email
s1302851@monmouth.edu
Phone
8624854841
Sign Out

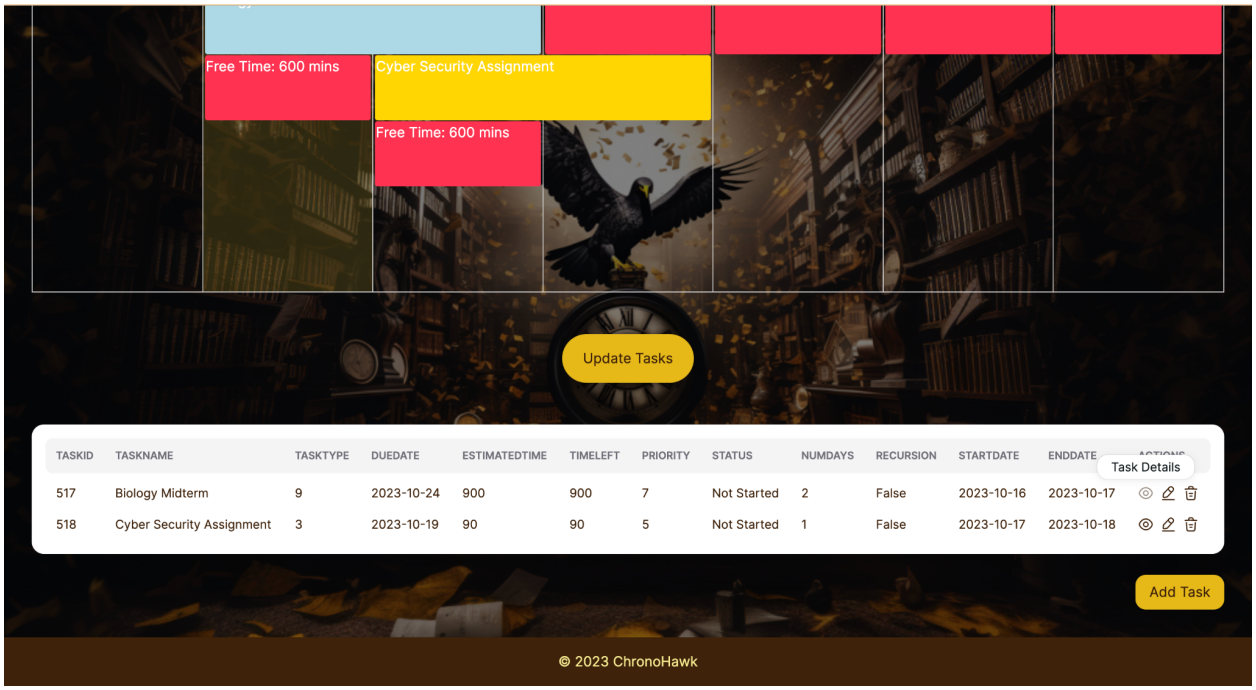

ChronoHawk
Home
Calendar
Account
FreeTime
Login
Sign Up
Enter Your Free Time During The Week
Monday - Minutes Available:
0
Tuesday - Minutes Available:
0
Wednesday - Minutes Available:
0
Thursday - Minutes Available:
0
Friday - Minutes Available:
0
Saturday - Minutes Available:
0
Sunday - Minutes Available:
0
Submit

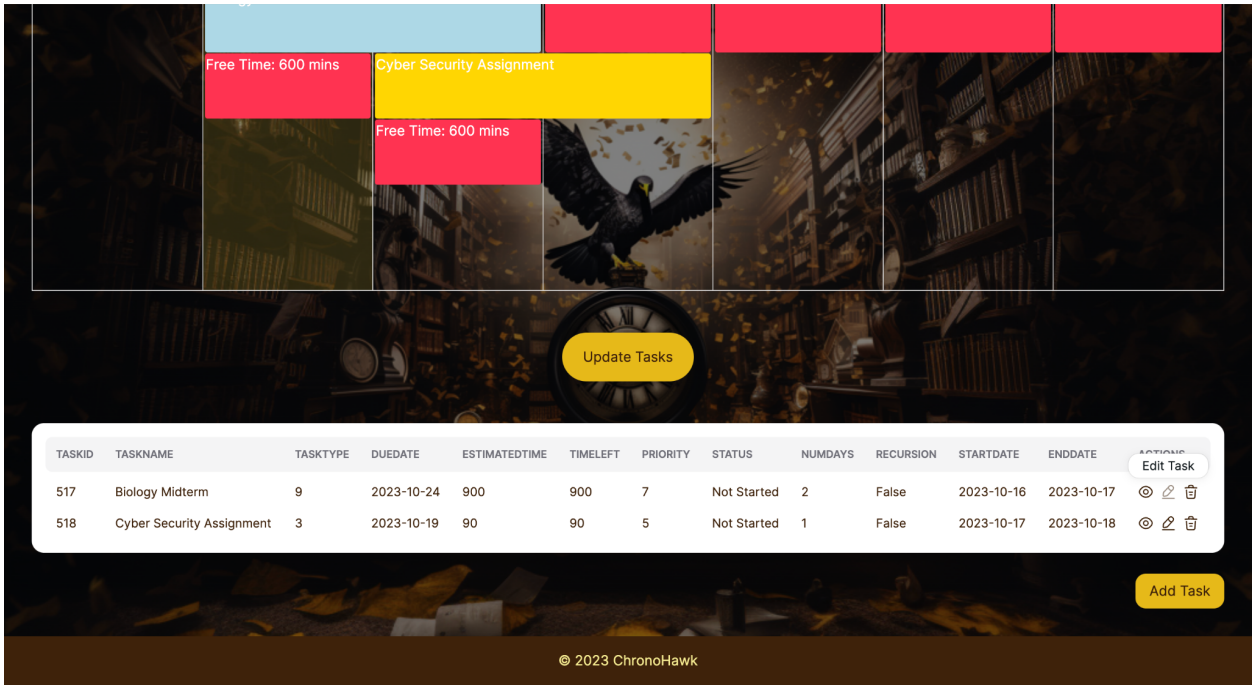

ChronoHawk
Home
Calendar
Account
FreeTime
Login
Sign Up
today
October 2023
month
week
Sun
Mon
Tue
Wed
Thu
Fri
Sat
1
2
3
4
5
6
7
8
9
10
11
12
13
14

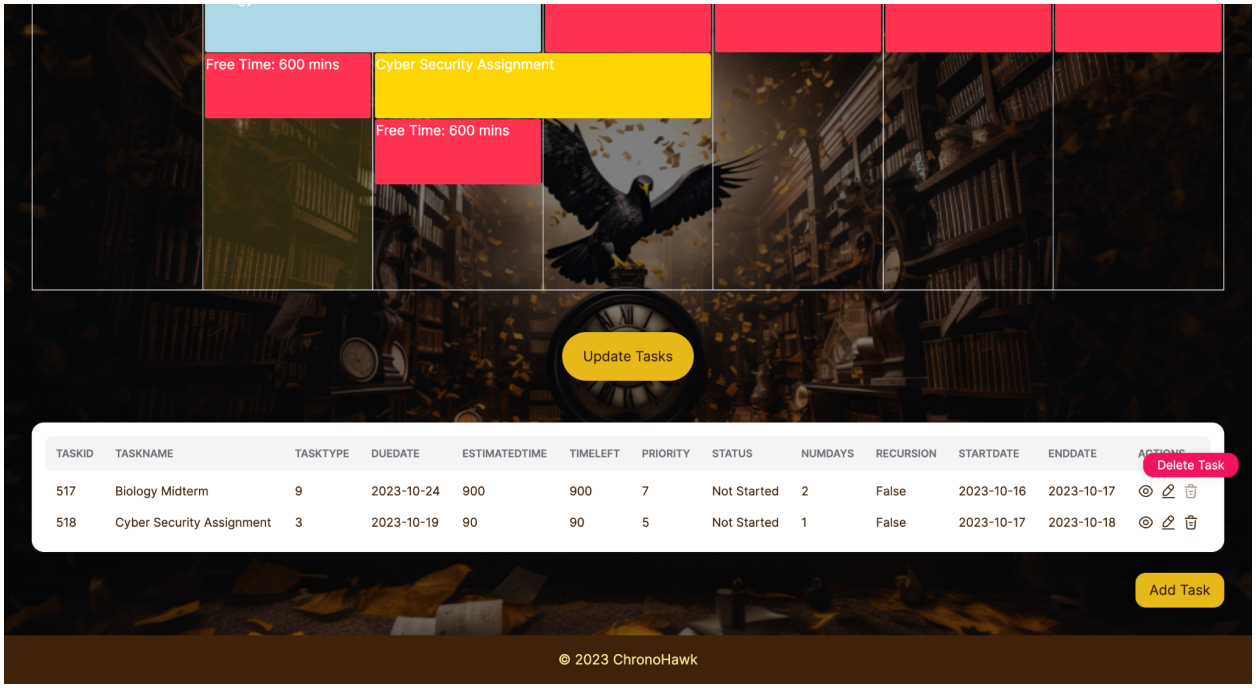

ChronoHawk
Home
Calendar
Account
FreeTime
Login
Sign Up
15
16
17
18
19
20
21
Biology Midterm
Free Time: 600 mins
Free Time: 600 mins
Free Time: 600 mins
Free Time: 600 mins
Free Time: 600 mins
Free Time: 600 mins
Free Time: 600 mins
Cyber Security Assignment
Free Time: 600 mins
22
23
24
25
26
27
28
Free Time: 600 mins
Free Time: 600 mins
Free Time: 600 mins
Free Time: 600 mins
Free Time: 600 mins
Free Time: 600 mins
Free Time: 600 mins

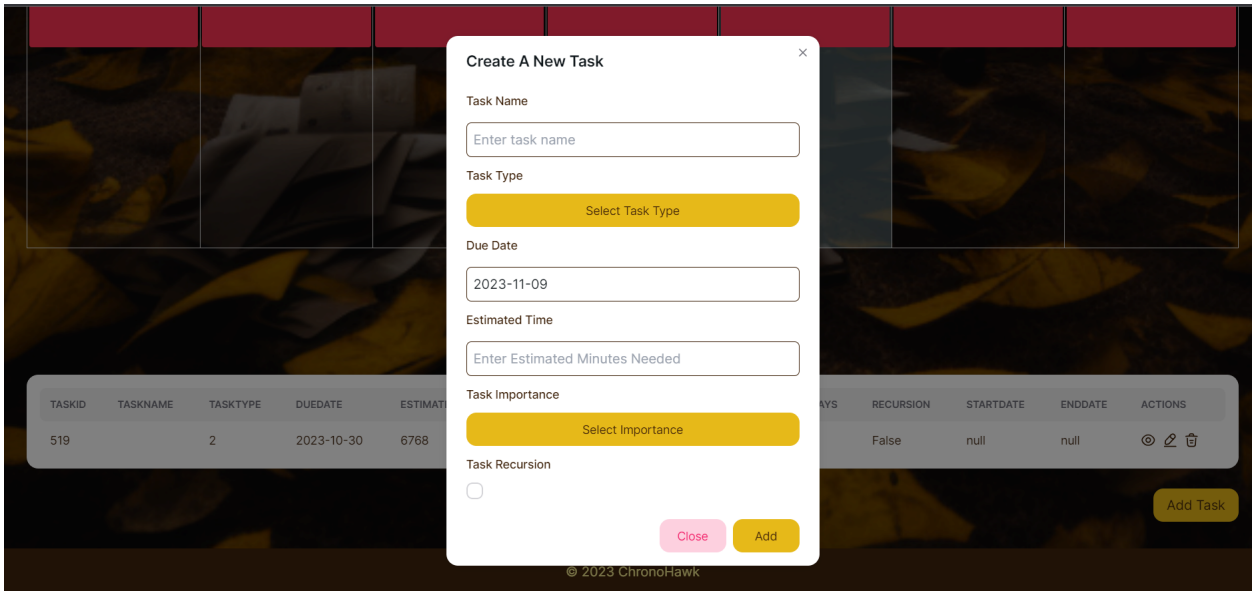

Free Time: 600 mins
Free Time: 600 mins
Free Time: 600 mins
Free Time: 600 mins
Free Time: 600 mins
Free Time: 600 mins
Free Time: 600 mins
Update Tasks
TASKID TASKNAME TASKTYPE DUEDATE ESTIMATEDTIME TIMELEFT PRIORITY STATUS NUMDAYS RECURSION STARTDATE ENDDATE ACTIONS
517 Biology Midterm 9 2023-10-24 900 900 7 Not Started 2 False 2023-10-16 2023-10-17
518 Cyber Security Assignment 3 2023-10-19 90 90 5 Not Started 1 False 2023-10-17 2023-10-18
Add Task
© 2023 ChronoHawk

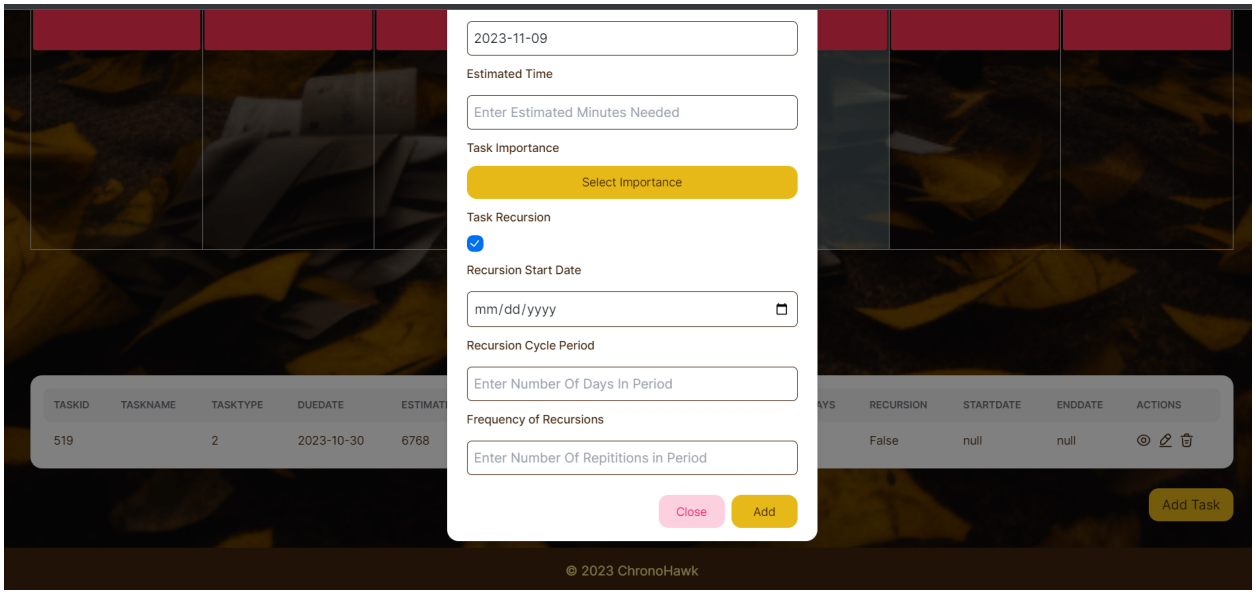

ChronoHawk
Home Calendar Account FreeTime
Login Sign Up
today
Oct 15 – 21, 2023
month week
Sun 10/15 Mon 10/16 Tue 10/17 Wed 10/18 Thu 10/19 Fri 10/20 Sat 10/21
Biology Midterm
Free Time: 600 mins
Free Time: 600 mins
Free Time: 600 mins
Free Time: 600 mins
Free Time: 600 mins
Free Time: 600 mins
Cyber Security Assignment
Free Time: 600 mins
Update Tasks
TASKID TASKNAME TASKTYPE DUEDATE ESTIMATEDTIME TIMELEFT PRIORITY STATUS NUMDAYS RECURSION STARTDATE ENDDATE ACTIONS
517 Biology Midterm 9 2023-10-24 900 900 7 Not Started 2 False 2023-10-16 2023-10-17

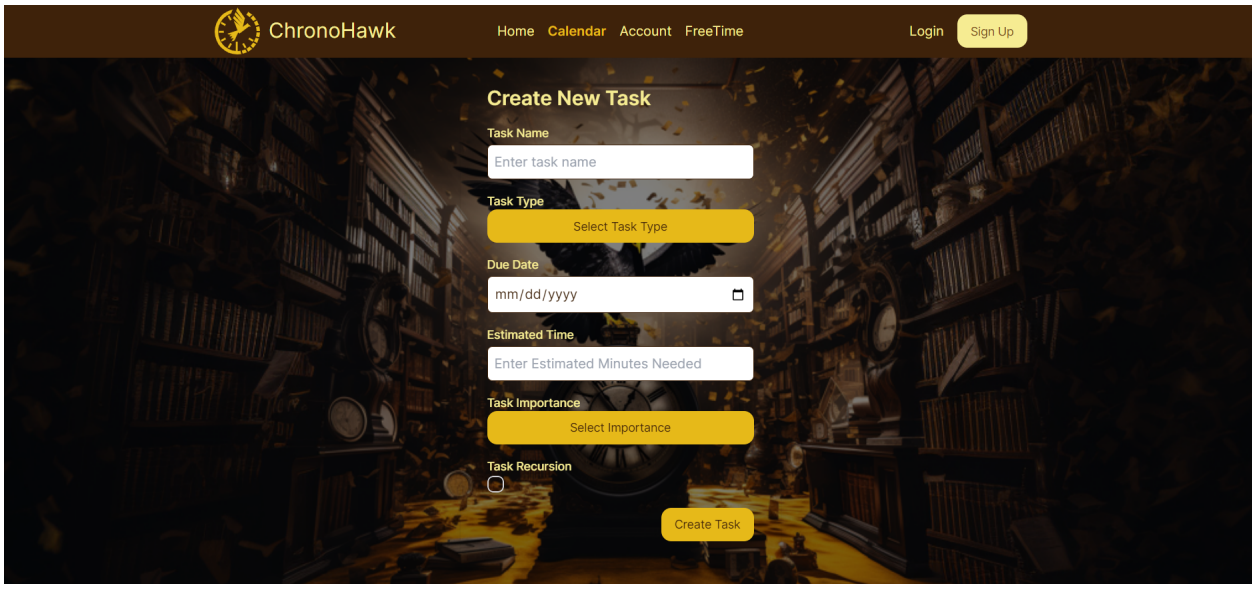

Free Time: 600 mins
Cyber Security Assignment
Free Time: 600 mins
Update Tasks
TASKID TASKNAME TASKTYPE DUEDATE ESTIMATEDTIME TIMELEFT PRIORITY STATUS NUMDAYS RECURSION STARTDATE ENDDATE
Task Details
517 Biology Midterm 9 2023-10-24 900 900 7 Not Started 2 False 2023-10-16 2023-10-17
518 Cyber Security Assignment 3 2023-10-19 90 90 5 Not Started 1 False 2023-10-17 2023-10-18
Add Task
© 2023 ChronoHawk

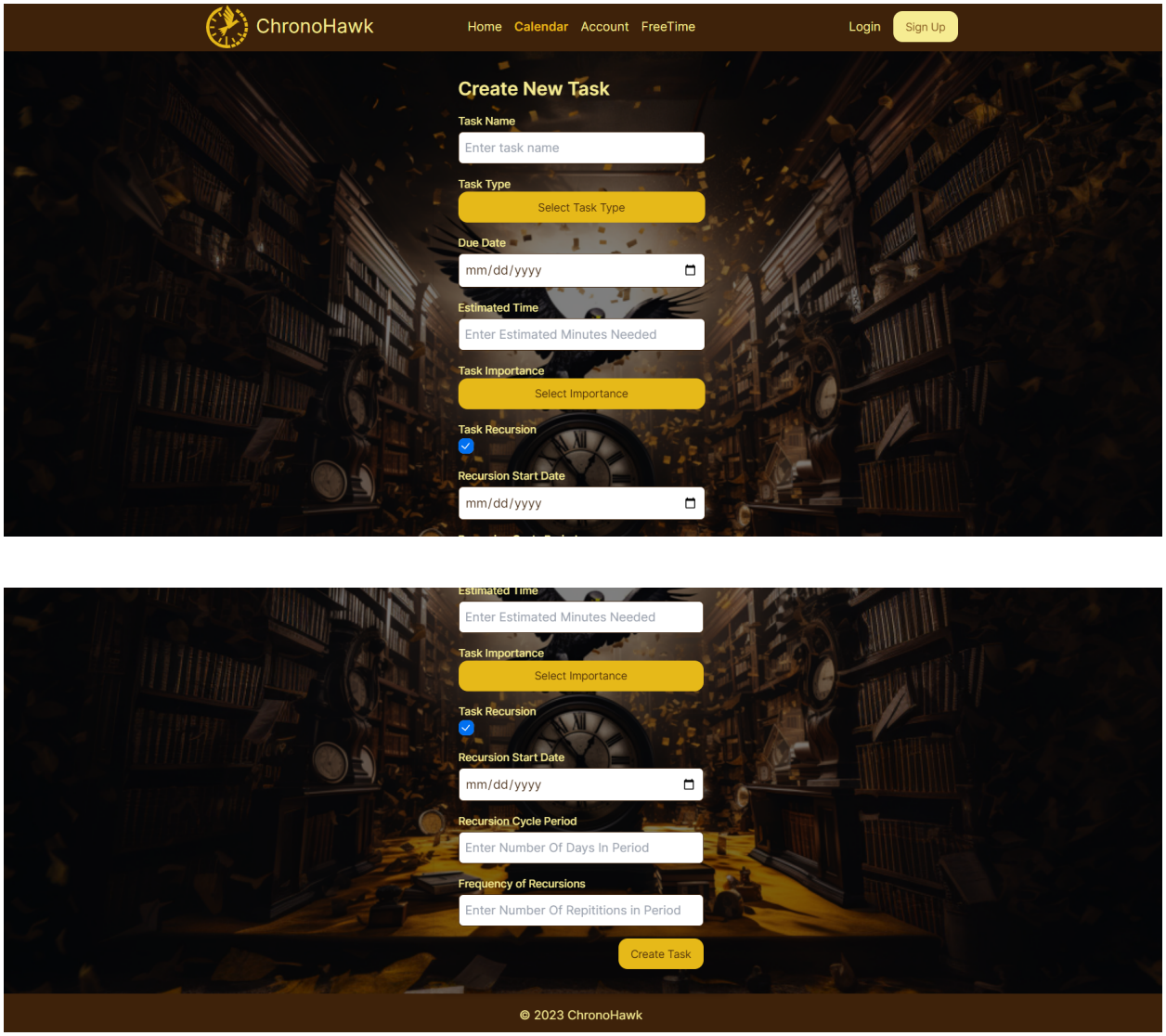

Free Time: 600 mins
Cyber Security Assignment
Free Time: 600 mins
Update Tasks
TASKID TASKNAME TASKTYPE DUEDATE ESTIMATEDTIME TIMELEFT PRIORITY STATUS NUMDAYS RECURSION STARTDATE ENDDATE
Edit Task
517 Biology Midterm 9 2023-10-24 900 900 7 Not Started 2 False 2023-10-16 2023-10-17
518 Cyber Security Assignment 3 2023-10-19 90 90 5 Not Started 1 False 2023-10-17 2023-10-18
Add Task
© 2023 ChronoHawk

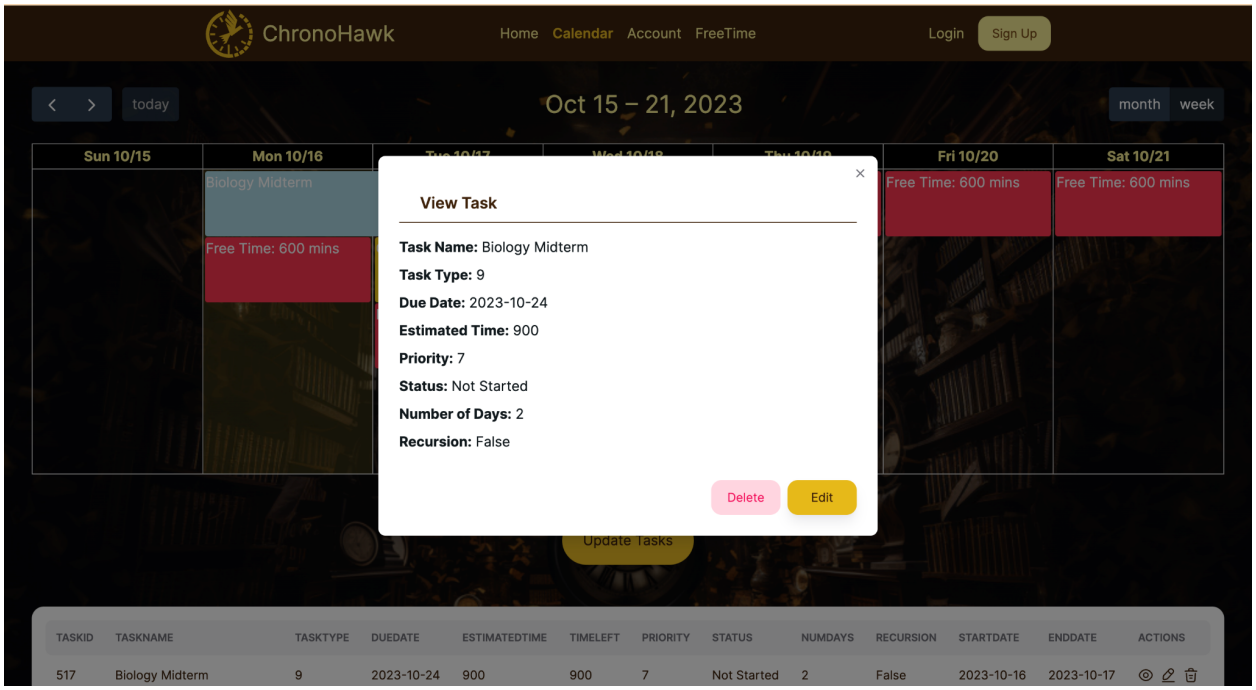

Free Time: 600 mins
Cyber Security Assignment
Free Time: 600 mins
Update Tasks
TASKID TASKNAME TASKTYPE DUEDATE ESTIMATEDTIME TIMELEFT PRIORITY STATUS NUMDAYS RECURSION STARTDATE ENDDATE ACTIONS
517 Biology Midterm 9 2023-10-24 900 900 7 Not Started 2 False 2023-10-16 2023-10-17
518 Cyber Security Assignment 3 2023-10-19 90 90 5 Not Started 1 False 2023-10-17 2023-10-18
Delete Task
Add Task
© 2023 ChronoHawk

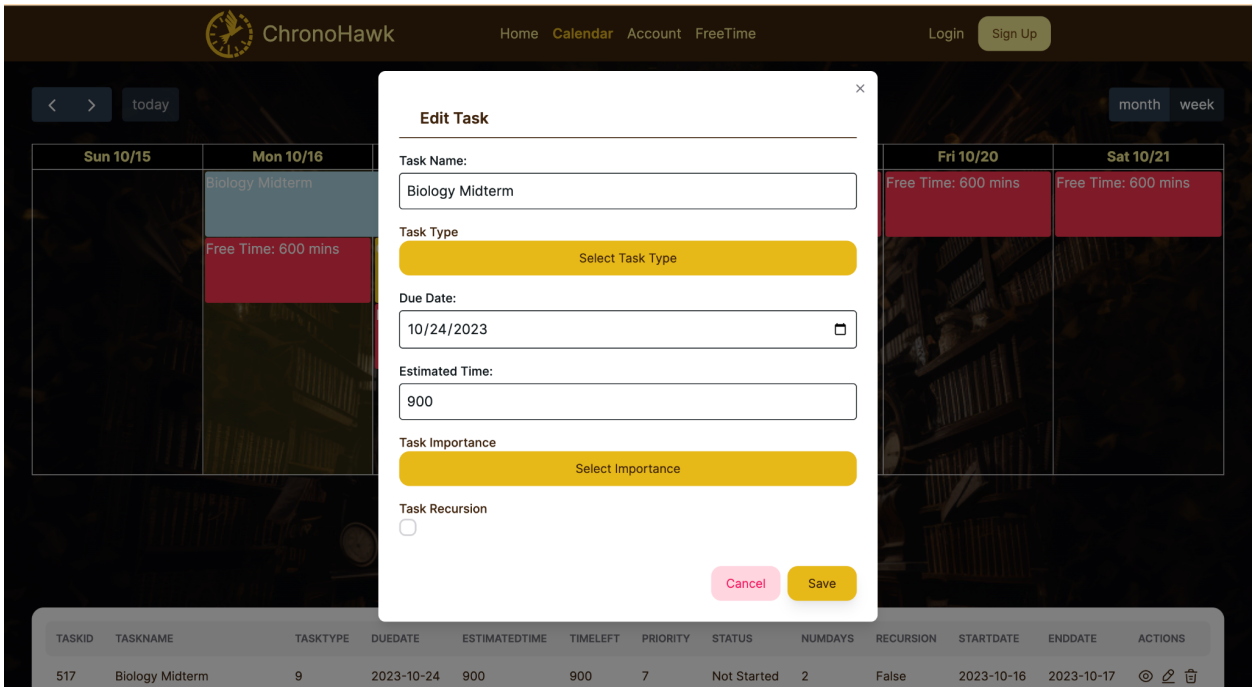

Create A New Task
Task Name
Enter task name
Task Type
Select Task Type
Due Date
2023-11-09
Estimated Time
Enter Estimated Minutes Needed
Task Importance
Select Importance
Task Recursion
Close
Add
TASKID TASKNAME TASKTYPE DUEDATE RECURSION STARTDATE ENDDATE ACTIONS
519 2 2023-10-30 False null null
Add Task
© 2023 ChronoHawk

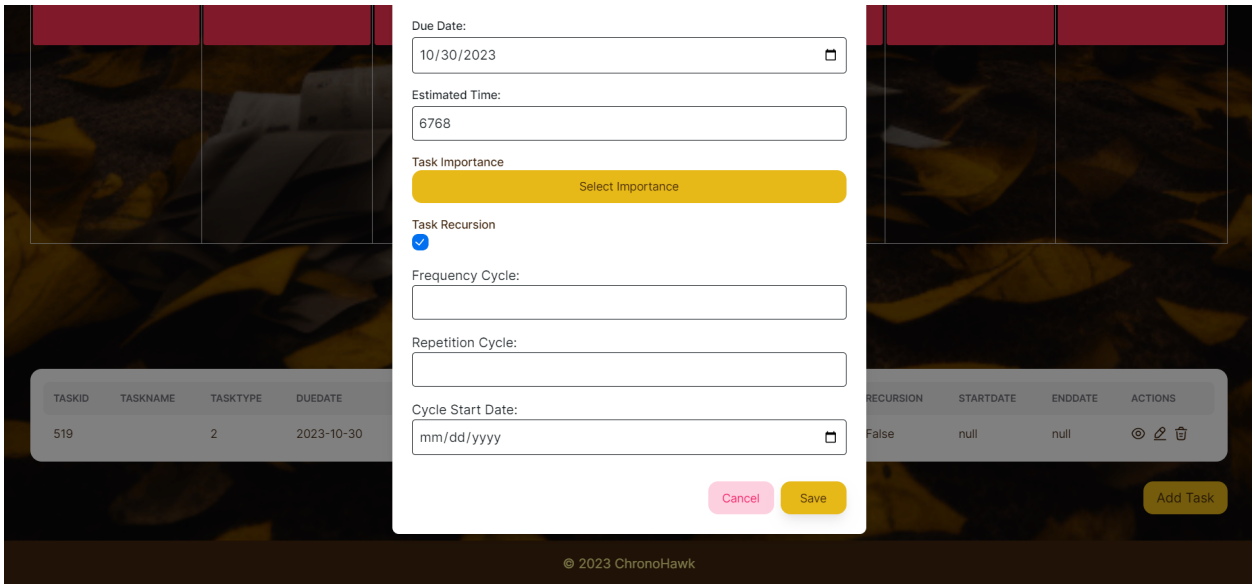

2023-11-09
Estimated Time
Enter Estimated Minutes Needed
Task Importance
Select Importance
Task Recursion
Recursion Start Date
mm/dd/yyyy
Recursion Cycle Period
Enter Number Of Days In Period
Frequency of Recursions
Enter Number Of Repititions in Period
Close
Add
TASKID
TASKNAME
TASKTYPE
DUEDATE
RECURSION
STARTDATE
ENDDATE
ACTIONS
519
2
2023-10-30
6768
False
null
null
Add Task
© 2023 ChronoHawk

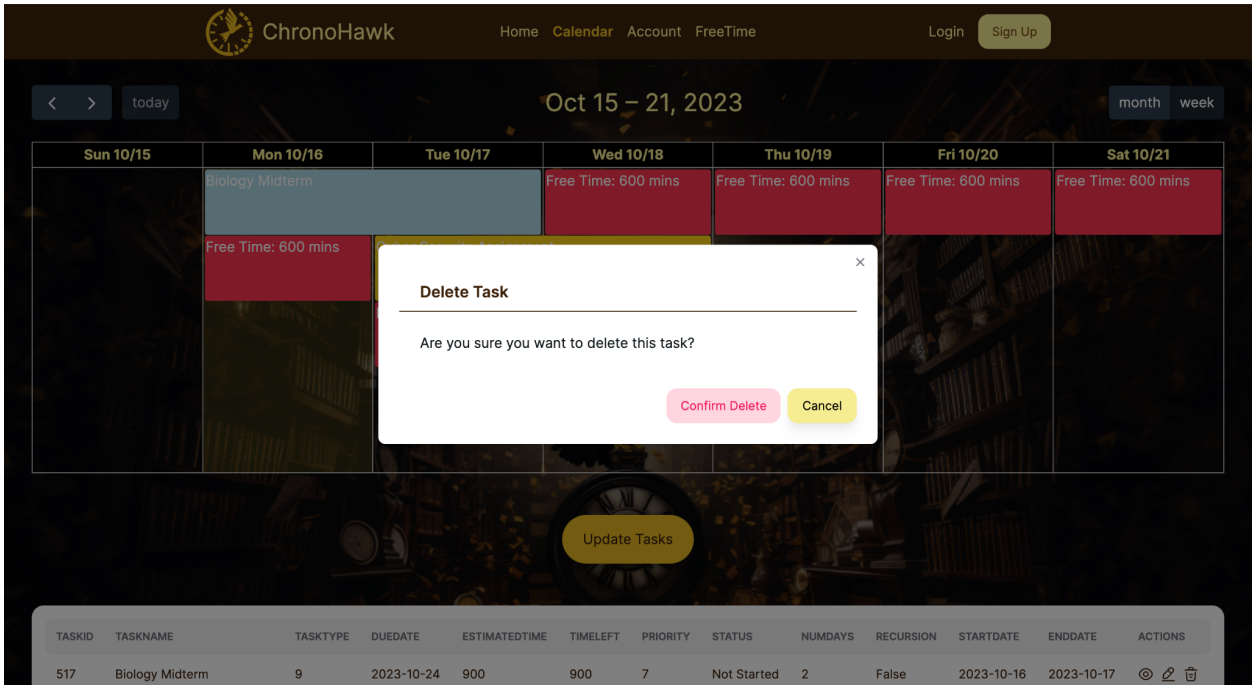

ChronoHawk
Home
Calendar
Account
FreeTime
Login
Sign Up
Create New Task
Task Name
Enter task name
Task Type
Select Task Type
Due Date
mm/dd/yyyy
Estimated Time
Enter Estimated Minutes Needed
Task Importance
Select Importance
Task Recursion
Create Task


ChronoHawk
Home
Calendar
Account
FreeTime
Login
Sign Up
Create New Task
Task Name
Enter task name
Task Type
Select Task Type
Due Date
mm/dd/yyyy
Estimated Time
Enter Estimated Minutes Needed
Task Importance
Select Importance
Task Recursion
Recursion Start Date
mm/dd/yyyy
Estimated Time
Enter Estimated Minutes Needed
Task Importance
Select Importance
Task Recursion
Recursion Start Date
mm/dd/yyyy
Recursion Cycle Period
Enter Number Of Days In Period
Frequency of Recursions
Enter Number Of Repititions in Period
Create Task
© 2023 ChronoHawk

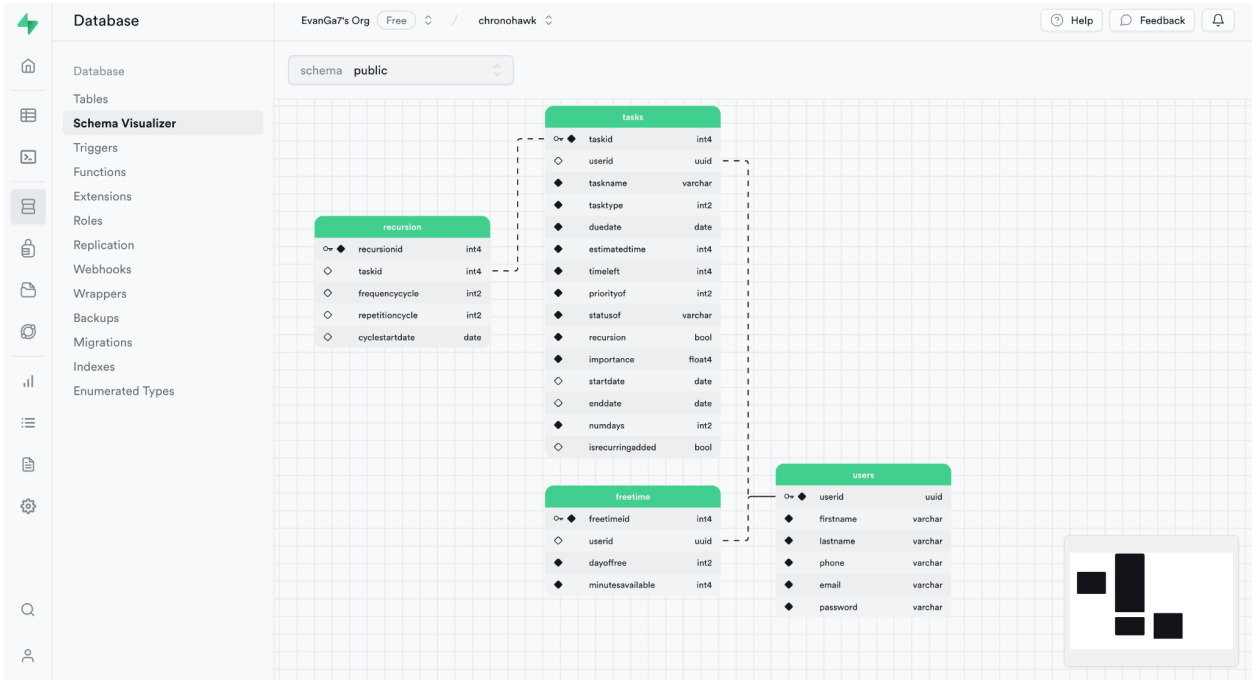

ChronoHawk
Home
Calendar
Account
FreeTime
Login
Sign Up
today
Oct 15 – 21, 2023
month
week
Sun 10/15
Mon 10/16
Fri 10/20
Sat 10/21
Biology Midterm
Free Time: 600 mins
Free Time: 600 mins
Free Time: 600 mins
View Task
Task Name: Biology Midterm
Task Type: 9
Due Date: 2023-10-24
Estimated Time: 900
Priority: 7
Status: Not Started
Number of Days: 2
Recursion: False
Delete
Edit
Update Tasks
TASKID
TASKNAME
TASKTYPE
DUEDATE
ESTIMATEDTIME
TIMELEFT
PRIORITY
STATUS
NUMDAYS
RECURSION
STARTDATE
ENDDATE
ACTIONS
517
Biology Midterm
9
2023-10-24
900
900
7
Not Started
2
False
2023-10-16
2023-10-17


ChronoHawk
Home
Calendar
Account
FreeTime
Login
Sign Up
today
month
week
Sun 10/15
Mon 10/16
Fri 10/20
Sat 10/21
Biology Midterm
Free Time: 600 mins
Free Time: 600 mins
Free Time: 600 mins
Edit Task
Task Name:
Biology Midterm
Task Type
Select Task Type
Due Date:
10/24/2023
Estimated Time:
900
Task Importance
Select Importance
Task Recursion
Cancel
Save
TASKID
TASKNAME
TASKTYPE
DUEDATE
ESTIMATEDTIME
TIMELEFT
PRIORITY
STATUS
NUMDAYS
RECURSION
STARTDATE
ENDDATE
ACTIONS
517
Biology Midterm
9
2023-10-24
900
900
7
Not Started
2
False
2023-10-16
2023-10-17


Due Date:
10/30/2023
Estimated Time:
6768
Task Importance
Select Importance
Task Recursion
Frequency Cycle:
Repetition Cycle:
Cycle Start Date:
mm/dd/yyyy
Cancel
Save
TASKID
TASKNAME
TASKTYPE
DUEDATE
RECURSION
STARTDATE
ENDDATE
ACTIONS
519
2
2023-10-30
False
null
null
Add Task
© 2023 ChronoHawk


ChronoHawk
Home
Calendar
Account
FreeTime
Login
Sign Up
today
Oct 15 – 21, 2023
month
week
Sun 10/15
Mon 10/16
Tue 10/17
Wed 10/18
Thu 10/19
Fri 10/20
Sat 10/21
Biology Midterm
Free Time: 600 mins
Delete Task
Are you sure you want to delete this task?
Confirm Delete
Cancel
Update Tasks
TASKID
TASKNAME
TASKTYPE
DUEDATE
ESTIMATEDTIME
TIMELEFT
PRIORITY
STATUS
NUMDAYS
RECURSION
STARTDATE
ENDDATE
ACTIONS
517
Biology Midterm
9
2023-10-24
900
900
7
Not Started
2
False
2023-10-16
2023-10-17

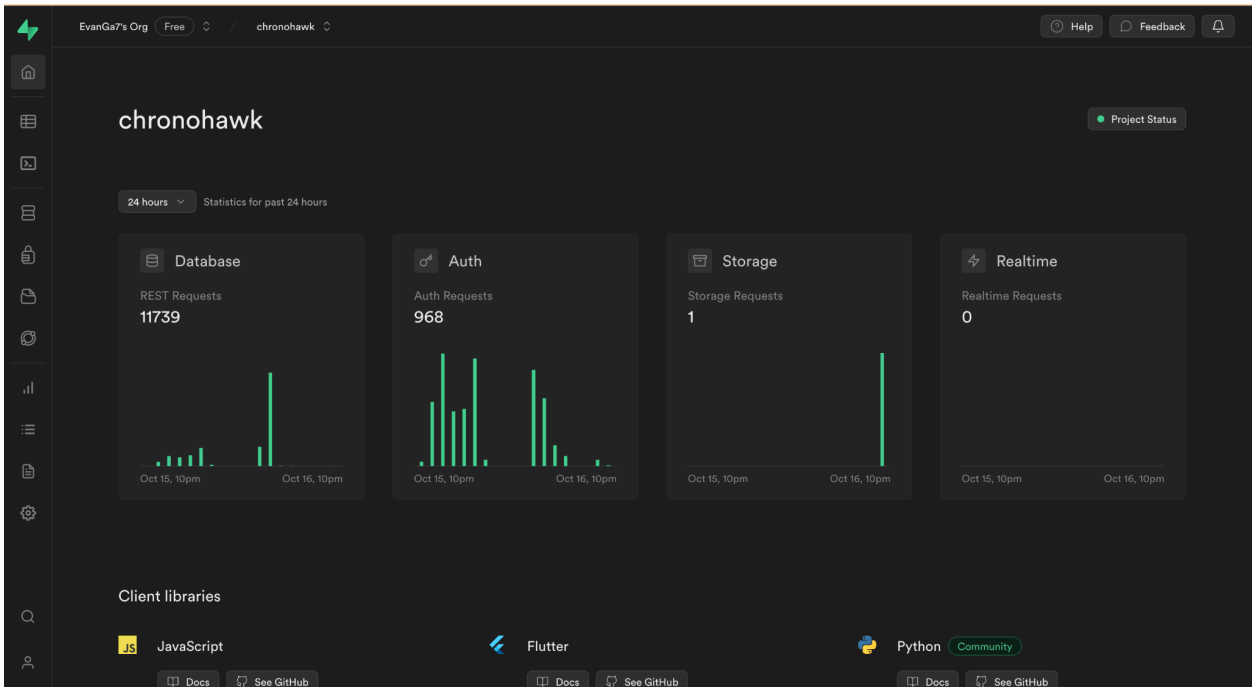
3.4 Database Design

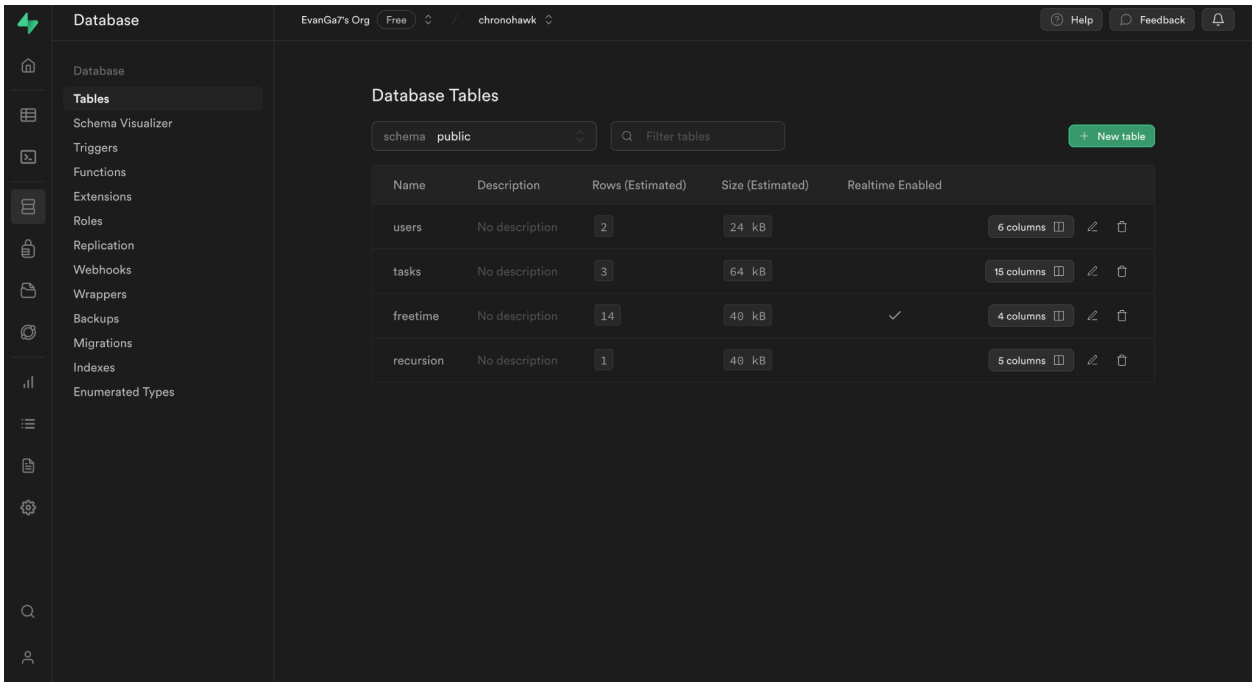


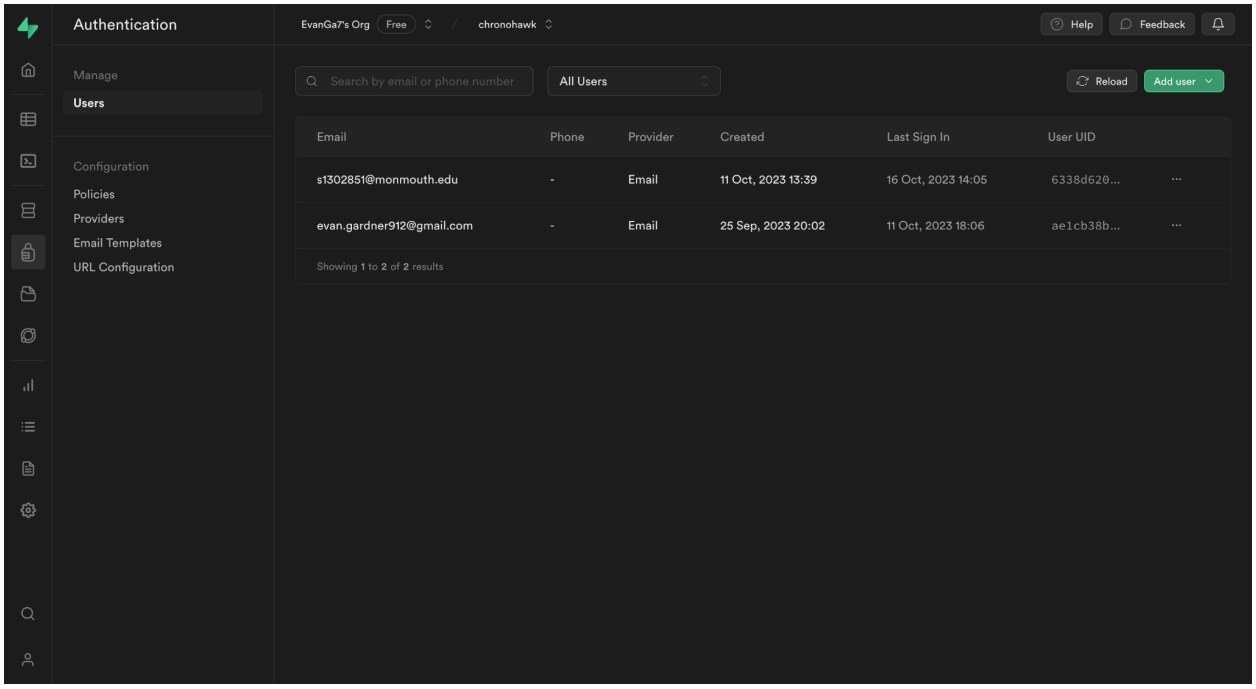

```
-- Creating the users table
CREATE TABLE users (
userid uuid PRIMARY KEY, --Supabase id that stores a list of numbers and letters that
are specific to each user
firstName VARCHAR(25) NOT NULL, --uses 25 characters to store the users first name
lastName VARCHAR(25) NOT NULL, --uses 25 characters to store the users first name
phone VARCHAR(15) NOT NULL, --Stores the user's phone number within 15 characters
email VARCHAR(50) NOT NULL, --This stores the user's email that has to be within 50
characters
password VARCHAR(100) NOT NULL --Stores the password of the user that is within 100
characters
);


-- Creating the tasks table
CREATE TABLE tasks (
taskid SERIAL PRIMARY KEY, --Autoincrementing value to create a unique task id
userid uuid REFERENCES users(userid), --Supabase id that stores a list of numbers and
letters that are specific to each user that references the user id from the users
table
taskname VARCHAR(100) NOT NULL, --Holds the name of the task that can hold up to 100
character
tasktype VARCHAR(15) NOT NULL, --Holds the type of task that can be up to 15
```

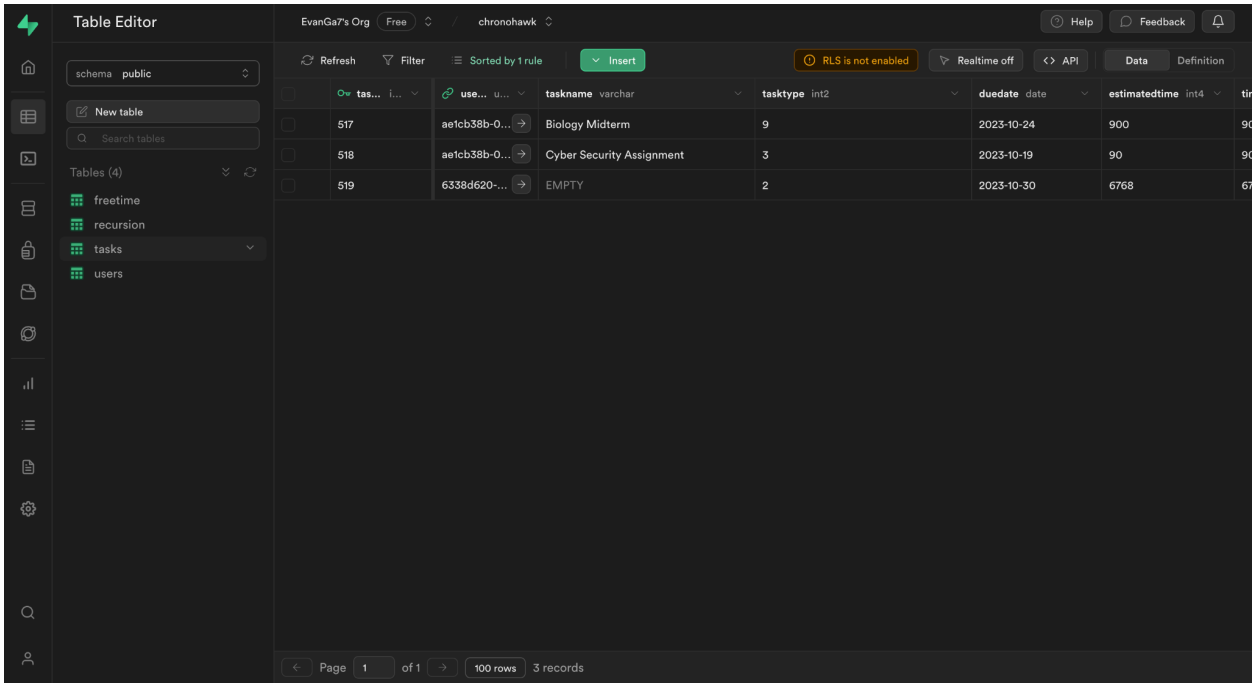
```
characters
duedate DATE NOT NULL, --This stores when the assignment is due with the value type of
date
estimatedtime INT NOT NULL, --This stores the estimated time to completion using the
value type of int
timeleft INT NOT NULL, --Holds the amount of time left that the user has not completed
priorityof SMALLINT NOT NULL, --Stores the priority of each task as a small integer
statusof VARCHAR(15) NOT NULL --Holds the if the task is completed or in progress of
the tasks using 15 characters
recursion BOOLEAN NOT NULL, --Uses a boolean to determine if it's recuring or not
importance FLOAT(4) NOT NULL, --This holds a value that shows how important the
assignment is
numdays SMALLINT NOT NULL, --Holds the number of days needed to complete using a small
int
startdate DATE, --Holds the start date that the user wants to start the preperation
for the task and this is a date type value
enddate DATE, --Holds when the preperateion time scheduled is over using a date value
isrecurringadded BOOLEAN --For recursion to determine if duplicate tasks have been
created for the current cycle so that it doesn't create duplicates
);

-- Creating the freeTime table
CREATE TABLE freeTime (
freetimeid SERIAL PRIMARY KEY, --Autoincrementing value to create a unique free time
id
userid uuid REFERENCES users(userid), --Supabase id that stores a list of numbers and
letters that are specific to each user that references the user id from the users
table
dayoffree SMALLINT NOT NULL, --An integer that represents what day of the week the
free time is for
minutesavailable INT NOT NULL, --Minutes available for each day that is stored as an
int
);

-- Creating the recursion table
CREATE TABLE recursion (
recursionid SERIAL PRIMARY KEY, --Autoincrementing value to create a unique free
recursion id
taskid INTEGER REFERENCES tasks(taskid),--Autoincrementing value to create a unique
task id that references taskid from the tasks table
frequencycycle SMALLINT, --Holds the number of days between each recursion using a
small integer
```

```
repetitioncycle SMALLINT, --Holds how long the repetitions will go until it ends using a small integer
cycleStartdate DATE, --This contains when the recursions begin using a date
);
```









3.5 Algorithms and Functions

3.5.1 Retrieve The Free Time For Each Day Of The Week

```
useEffect(() => {
const freeTimeObj = freeTime.reduce((acc, curr) => {
```

```
acc[curr.dayoffree] = curr.minutesavailable;
return acc;
}, {});
setFreeTimeByDay(freeTimeObj);
}, [freeTime]);

function getNextDateForDay(dayNum: number): Date {
const today = new Date();
const resultDate = new Date(today);
resultDate.setDate(today.getDate() + (dayNum + 7 - today.getDay()) % 7);
return resultDate;
}

const freeTimeEvents = [];

freeTime.forEach(freeTimeEntry => {
const nextDate = getNextDateForDay(freeTimeEntry.dayoffree);
for (let i = 0; i < 52; i++) { // Assuming you want to create events for 52 weeks
freeTimeEvents.push({
title: `Free Time: ${freeTimeEntry.minutesavailable} mins`,
start: new Date(nextDate),
end: new Date(nextDate),
color: '#FF3352', // Ensure this line is present and correct
allDay: true,
});
nextDate.setDate(nextDate.getDate() + 7); // Move to the same day of the next week
}
});
```

3.5.2 Calculate Task Duration

```
function calculateTaskDuration(taskDuration, startDate, dueDate, numdays) {
  let remainingDuration = taskDuration;
  let currentDate = new Date(startDate);
  let endDate = new Date(startDate);

  let daysScheduled = 0;

  while (daysScheduled < numdays) {
      const dayOfWeek = currentDate.getDay();
      const availableTime = freeTimeByDay[dayOfWeek] || 0;

      if (dueDate && currentDate > dueDate) {
```

```
            console.error("Task cannot be scheduled within the due date!");
            return null;
        }

        if (remainingDuration > 0) {
            if (availableTime >= remainingDuration) {
                endDate.setMinutes(endDate.getMinutes() + remainingDuration);
                remainingDuration = 0;
            } else {
                endDate.setMinutes(endDate.getMinutes() + availableTime);
                remainingDuration -= availableTime;
            }
        }

        // If there's no remaining duration, break out of the loop
        if (remainingDuration <= 0) {
            break;
        }

        currentDate.setDate(currentDate.getDate() + 1);
        endDate = new Date(currentDate);
        daysScheduled++;
    }

    return {
        start: startDate,
        end: endDate,
    };
}
```

3.5.3 Update Tasks With Dates

```
async function updateTasksWithDates() {
    // Sort tasks based on priority
    const sortedTasks = [...tasks].sort((a, b) => b.priorityof - a.priorityof);

    let currentDate = new Date(); // Start scheduling from today

    let i = 0;
    while (i < sortedTasks.length) {
        const task = sortedTasks[i];
        const result = calculateTaskDuration(task.estimatedtime, currentDate,
task.duedate, task.numdays);

        if (!result) {
            console.error("Couldn't schedule task:", task);
            continue;
        }

        const { start, end } = result;
```

```
      task.start = start;
      task.end = end;

      // Handle recursion
      if (task.recursion && task.recursionDetails ) {
        const { cyclestartdate, repetitioncycle, frequencycycle } =
task.recursionDetails;
        let recursionDate = new Date(cyclestartdate);
        const recursionEndDate = new Date(cyclestartdate);
        recursionEndDate.setDate(recursionEndDate.getDate() + repetitioncycle -
1); // Subtracting 1 to include the start date in the cycle
        while (recursionDate <= recursionEndDate) {
          // Schedule the recurring task
          const recurringTask = { ...task, start: recursionDate, duedate: new
Date(recursionDate) };
          const recurringResult =
calculateTaskDuration(recurringTask.estimatedtime, recursionDate,
recurringTask.duedate, recurringTask.numdays);

          if (recurringResult) {

              if (recurringTask.isrecurringadded == true) {
                  // Task exists, so update it
                  const { error: updateError } = await supabase
                      .from('tasks')
                      .update({
                        taskname: recurringTask.title,
                        duedate: recurringTask.duedate,
                        userid: userId,
                        priorityof: recurringTask.priorityof,
                        estimatedtime: recurringTask.estimatedtime,
                        tasktype: recurringTask.tasktype,
                        numdays: recurringTask.numdays,
                        importance: recurringTask.importance,
                        statusof: recurringTask.statusof,
                        timeleft: recurringTask.timeleft,
                        recursion: recurringTask.recursion,
                        startdate: recurringResult.start,
                        enddate: recurringResult.end
                      })
                      .eq('taskid',recurringTask.taskid);

                  if (updateError) {
                      console.error("Error updating recurring task:",
updateError.message);
                  }
              } else {
                  const { error: updateError } = await supabase
                  .from('tasks')
                  .update({
                    isrecurringadded: true
                  })
```

```
                    .eq('taskid',recurringTask.taskid);

                    if (updateError) {
                      console.error("Error updating recurring task:",
updateError.message);
                  }

                    // Task doesn't exist, so insert it
                    const { error: insertError } = await supabase
                        .from('tasks')
                        .insert({
                            taskname: recurringTask.title,
                            startdate: recurringResult.start,
                            enddate: recurringResult.end,
                            duedate: recurringTask.duedate,
                            userid: userId,
                            priorityof: recurringTask.priorityof,
                            estimatedtime: recurringTask.estimatedtime,
                            tasktype: recurringTask.tasktype,
                            numdays: recurringTask.numdays,
                            importance: recurringTask.importance,
                            statusof: recurringTask.statusof,
                            timeleft: recurringTask.timeleft,
                            recursion: recurringTask.recursion,
                            isrecurringadded: true
                        });

                    if (insertError) {
                        console.error("Error inserting recurring task:",
insertError.message);
                    }
                }
            }

            // Move to the next recursion date
            recursionDate.setDate(recursionDate.getDate() + frequencycycle);
        }}

        else{
        // Handle non-recurring tasks
        // Update the currentDate to the end date of the last scheduled task
        currentDate = new Date(end);

        // Check if there's any remaining time on the current day
        const dayOfWeek = currentDate.getDay();
        const availableTime = freeTimeByDay[dayOfWeek] || 0;
        const timeUsed = (end.getHours() * 60 + end.getMinutes()) -
(start.getHours() * 60 + start.getMinutes());
        const remainingTime = availableTime - timeUsed;

        // If the next task can fit into the remaining time, schedule it on the
same day
```

```
      if (i + 1 < sortedTasks.length && sortedTasks[i + 1].estimatedtime <=
remainingTime) {
        i++; // Move to the next task
        const nextTask = sortedTasks[i];
        nextTask.start = currentDate;
        nextTask.end = new Date(currentDate.getTime() + nextTask.estimatedtime *
60 * 1000);
      } else {
          currentDate.setDate(currentDate.getDate() + 1); // Move to the next day
          i++;
      }

      console.log("Task:", task);
      // Update the task in the database
      const { error } = await supabase
          .from('tasks')
          .update({
              startdate: task.start,
              enddate: task.end
          })
          .eq('taskid', task.taskid);

      if (error) {
          console.error("Error updating task:", error.message);
      }
  }
}
  window.location.reload();
}
```

3.5.4 Calculate Number Of Days Needed Based On Free Time

```
    //calculate the number of days needed to complete using the estimated time
and the free time
    const estimatedTime = parseFloat(formData.estimatedtime);
    const freeTimePerDay = freeTimeData;
    let numDays = 0;
    let timeLeft = estimatedTime;
    let day = 0;
    while (timeLeft > 0) {
      if (day > 6) {
        day = 0;
      }
      if (freeTimePerDay[day].minutesavailable > 0) {
        timeLeft -= freeTimePerDay[day].minutesavailable;
        numDays++;
      }
      day++;
    }
```

3.5.5 Determine The Priority Of The Task

```
function prioritizeTasks(
taskType: string,
dueDate: Date,
today: Date,
status: string,
freeTimePerDay: [
{
freetimeid: string,
userid : string,
dayoffree: number,
minutesavailable: number,
}
],
estimatedTimeNeeded: number,
daysThoughtNeeded: number,
importance: number,
timeLeft: number
): number {
const maxUrgency = 10;
const days = [0, 1, 2, 3, 4, 5, 6];

// 1. Task Type Urgency
const taskTypeWeights = {
```

```
1: 4, // Test
2: 4, // Quiz
3: 2, // Assignment
4: 3, // Project
5: 2, // Lecture
6: 1, // Reading
7: 2, // Discussion
8: 5, // Final
9: 5, // Midterm
10: 3, // Presentation
11: 3, // Paper
};
const taskTypeUrgency = taskTypeWeights[taskType] || 1;

// 2. Due Date Urgency
const daysLeft = Math.ceil((dueDate.getTime() - today.getTime()) / (1000 * 60 * 60 *
24));
const dueDateUrgency = daysLeft <= 1 ? 3 : 2/daysLeft;

// 3. Status Urgency
const statusWeights = {
'Not Started': 1,
'Completed': 0
};
const statusUrgency = statusWeights[status];

// 4. Free Time Urgency
// const freeTimeUrgency = estimatedTimeNeeded / (freeTimePerDay + 1); // +1 to avoid
division by zero
let freeTimeUrgency = 0;
freeTimePerDay.forEach(freeTime => {
if (days.includes(freeTime.dayoffree)) {
freeTimeUrgency += freeTime.minutesavailable;
}
});

// 5. Time Needed Urgency
const timeNeededUrgency = daysLeft <= daysThoughtNeeded ? 2 : 1;

// 7. Time Left Urgency
const timeLeftUrgency = (estimatedTimeNeeded - timeLeft + 1) / (estimatedTimeNeeded +
1); // +1 to avoid division by zero
```

```
// Maximum possible urgencies for each component
const maxTaskTypeUrgency = Math.max(...Object.values(taskTypeWeights));
const maxDueDateUrgency = 3; // Since daysLeft <= 1 is the maximum value
const maxStatusUrgency = Math.max(...Object.values(statusWeights));
const maxFreeTimeUrgency = 7 * 24 * 60; // Assuming a week's worth of minutes as the
maximum
const maxTimeNeededUrgency = 2; // Since daysLeft <= daysThoughtNeeded is the maximum
value
const maxImportanceUrgency = 1; // Since importance is squared and it's between 0 and
1
const maxTimeLeftUrgency = 1; // Since it's a fraction

const maxPossibleUrgency = maxTaskTypeUrgency + maxDueDateUrgency + maxStatusUrgency +
maxFreeTimeUrgency + maxTimeNeededUrgency + maxImportanceUrgency + maxTimeLeftUrgency;

// Normalize each urgency individually
const normalizedTaskTypeUrgency = taskTypeUrgency / maxTaskTypeUrgency;
const normalizedDueDateUrgency = dueDateUrgency / maxDueDateUrgency;
const normalizedStatusUrgency = statusUrgency / maxStatusUrgency;
const normalizedFreeTimeUrgency = freeTimeUrgency / maxFreeTimeUrgency;
const normalizedTimeNeededUrgency = timeNeededUrgency / maxTimeNeededUrgency;
const normalizedImportance = importance; // Since it's already between 0 and 1
const normalizedTimeLeftUrgency = timeLeftUrgency; // Since it's already between 0 and
1

// Combine normalized urgencies
const combinedUrgency = normalizedTaskTypeUrgency + normalizedDueDateUrgency +
normalizedStatusUrgency + normalizedFreeTimeUrgency + normalizedTimeNeededUrgency +
normalizedImportance + normalizedTimeLeftUrgency;

// Normalize the combined urgency to be between 1 and 10
const finalUrgency = Math.min(Math.max((combinedUrgency / 7) * 10, 1), 10); // Divided
by 7 because there are 7 factors

const urgency = Math.floor(finalUrgency);
return urgency;
}
```

3.5.6 Calculate Recursion Within Update Dates

```
// Handle recursion
if (task.recursion && task.recursionDetails ) {
const { cyclestartdate, repetitioncycle, frequencycycle } = task.recursionDetails;
let recursionDate = new Date(cyclestartdate);
const recursionEndDate = new Date(cyclestartdate);
recursionEndDate.setDate(recursionEndDate.getDate() + repetitioncycle - 1); //
Subtracting 1 to include the start date in the cycle
while (recursionDate <= recursionEndDate) {
// Schedule the recurring task
const recurringTask = { ...task, start: recursionDate, duedate: new
Date(recursionDate) };
const recurringResult = calculateTaskDuration(recurringTask.estimatedtime,
recursionDate, recurringTask.duedate, recurringTask.numdays);
if (recurringResult) {
if (recurringTask.isrecurringadded == true) {
// Task exists, so update it
const { error: updateError } = await supabase
.from('tasks')
.update({
taskname: recurringTask.title,
duedate: recurringTask.duedate,
userid: userId,
priorityof: recurringTask.priorityof,
estimatedtime: recurringTask.estimatedtime,
tasktype: recurringTask.tasktype,
numdays: recurringTask.numdays,
importance: recurringTask.importance,
statusof: recurringTask.statusof,
timeleft: recurringTask.timeleft,
recursion: recurringTask.recursion,
startdate: recurringResult.start,
enddate: recurringResult.end
})
.eq('taskid',recurringTask.taskid);
if (updateError) {
console.error("Error updating recurring task:", updateError.message);
}
} else {
const { error: updateError } = await supabase
.from('tasks')
.update({
isrecurringadded: true
```

```
})
.eq('taskid',recurringTask.taskid);

if (updateError) {
console.error("Error updating recurring task:", updateError.message);
}
// Task doesn't exist, so insert it
const { error: insertError } = await supabase
.from('tasks')
.insert({
taskname: recurringTask.title,
startdate: recurringResult.start,
enddate: recurringResult.end,
duedate: recurringTask.duedate,
userid: userId,
priorityof: recurringTask.priorityof,
estimatedtime: recurringTask.estimatedtime,
tasktype: recurringTask.tasktype,
numdays: recurringTask.numdays,
importance: recurringTask.importance,
statusof: recurringTask.statusof,
timeleft: recurringTask.timeleft,
recursion: recurringTask.recursion,
isrecurringadded: true
});
if (insertError) {
console.error("Error inserting recurring task:", insertError.message);
}
}
}
// Move to the next recursion date
recursionDate.setDate(recursionDate.getDate() + frequencycycle);
}}
```

3.6 Implementation

3.6.1 Supabase/ Database Setup

We used a UML diagram to design how the SQL database would look. Then we converted it to readable code and adjusted as needed, so that it would be as effective as possible. We then used supabase using the SQL setup database to create the database. We also used

authentication to authenticate new users using their email.

3.6.2 Create Task Code Segment

```
 <div className="main-bg relative min-h-screen flex items-center justify-center
text-buddha-950">
      <div className="bg-white p-8 rounded shadow-md w-96">
        <h2 className="text-2xl font-semibold mb-4 text-buddha-200">Create New
Task</h2>
        <form onSubmit={handleSubmit}>
          <div className="mb-4">
            <label htmlFor="taskname" className="block text-sm font-medium
text-buddha-200">
              Task Name
            </label>
            <input
              type="text"
              id="taskname"
              name="taskname"
              className="mt-1 p-2 w-full border border-buddha-950 rounded-md"
              placeholder="Enter task name"
              onChange={handleChange}
              required
            />
          </div>
          <div className="mb-4">
            <label className="block text-sm font-medium text-buddha-200">
                Task Type
            </label>
            <Dropdown>
                <DropdownTrigger>
                    <Button variant="flat" className='bg-buddha-500
text-buddha-950 w-full'>
                      {selectedTaskType ? selectedTaskType : 'Select Task Type'}
                    </Button>
                </DropdownTrigger>
                <DropdownMenu aria-label="Task Types">
                  <DropdownItem key="type1" onClick={() =>
handleTaskTypeChange('Test')}>Test</DropdownItem>
                  <DropdownItem key="type2" onClick={() =>
handleTaskTypeChange('Quiz')}>Quiz</DropdownItem>
                  <DropdownItem key="type3" onClick={() =>
handleTaskTypeChange('Assignment')}>Assignment</DropdownItem>
                  <DropdownItem key="type4" onClick={() =>
handleTaskTypeChange('Project')}>Project</DropdownItem>
                  <DropdownItem key="type5" onClick={() =>
handleTaskTypeChange('Lecture')}>Lecture</DropdownItem>
                  <DropdownItem key="type6" onClick={() =>
handleTaskTypeChange('Reading')}>Reading</DropdownItem>
                  <DropdownItem key="type7" onClick={() =>
handleTaskTypeChange('Discussion')}>Discussion</DropdownItem>
```

```
                  <DropdownItem key="type8" onClick={() =>
handleTaskTypeChange('Final')}>Final</DropdownItem>
                  <DropdownItem key="type9" onClick={() =>
handleTaskTypeChange('Midterm')}>Midterm</DropdownItem>
                  <DropdownItem key="type10" onClick={() =>
handleTaskTypeChange('Presentation')}>Presentation</DropdownItem>
                  <DropdownItem key="type11" onClick={() =>
handleTaskTypeChange('Paper')}>Paper</DropdownItem>
              </DropdownMenu>
            </Dropdown>
          </div>
          <div className="mb-4">
            <label htmlFor="duedate" className="block text-sm font-medium
text-buddha-200">
              Due Date
            </label>
            <input
              type="date"
              id="duedate"
              name="duedate"
              className="mt-1 p-2 w-full border border-buddha-950 rounded-md"
              placeholder="Enter task name"
              onChange={handleChange}
              required
            />
          </div>
          <div className="mb-4">
            <label htmlFor="estimatedtime" className="block text-sm font-medium
text-buddha-200">
              Estimated Time
            </label>
            <input
              type="number"
              id="estimatedtime"
              name="estimatedtime"
              className="mt-1 p-2 w-full border border-buddha-950 rounded-md"
              placeholder="Enter Estimated Minutes Needed"
              onChange={handleChange}
              required
            />
          </div>
          <div className="mb-4">
              <label className="block text-sm font-medium text-buddha-200">
                  Task Importance
              </label>
              <Dropdown>
                  <DropdownTrigger>
                      <Button variant="flat" className='bg-buddha-500
text-buddha-950 w-full'>
                          {selectedImportance ? selectedImportance : 'Select
Importance'}
                      </Button>
```

```
                </DropdownTrigger>
                <DropdownMenu aria-label="Task Importance">
                    <DropdownItem key="importance1" onClick={() =>
handleImportanceChange('Low Importance', 0.25)}>Low Importance</DropdownItem>
                    <DropdownItem key="importance2" onClick={() =>
handleImportanceChange('Medium Importance', 0.5)}>Medium
Importance</DropdownItem>
                    <DropdownItem key="importance3" onClick={() =>
handleImportanceChange('High Importance', 0.75)}>High Importance</DropdownItem>
                    <DropdownItem key="importance4" onClick={() =>
handleImportanceChange('ASAP', 1)}>Critical Importance</DropdownItem>
                </DropdownMenu>
            </Dropdown>
          </div>
          <div className="mb-4">
            <label htmlFor="recursion" className="block text-sm font-medium
text-buddha-200">
              Task Recursion
            </label>
            <Checkbox onChange={handleCheckboxChange} />
          </div>

          {isRecursive && (
            <>
              <div className="mb-4">
                <label htmlFor="cyclestartdate" className="block text-sm
font-medium text-buddha-200">
                  Recursion Start Date
                </label>
                <input
                  type="date"
                  id="cyclestartdate"
                  name="cyclestartdate"
                  className="mt-1 p-2 w-full border border-buddha-950 rounded-md"
                  placeholder="Enter Start Date"
                  onChange={handleChange}
                  required
                />
              </div>
              <div className="mb-4">
                <label htmlFor="repetitioncycle" className="block text-sm
font-medium text-buddha-200">
                  Recursion Cycle Period
                </label>
                <input
                  type="number"
                  id="repetitioncycle"
                  name="repetitioncycle"
                  className="mt-1 p-2 w-full border border-buddha-950 rounded-md"
                  placeholder="Enter Number Of Days In Period"
                  onChange={handleChange}
                  required
```

```
              />
            </div>
            <div className="mb-4">
              <label htmlFor="frequencycycle" className="block text-sm
font-medium text-buddha-200">
                Frequency of Recursions
              </label>
              <input
                type="number"
                id="frequencycycle"
                name="frequencycycle"
                className="mt-1 p-2 w-full border border-buddha-950 rounded-md"
                placeholder="Enter Number Of Repititions in Period"
                onChange={handleChange}
                required
              />
            </div>
          </>
        )}
        <div className="flex justify-end">
          <Button className="bg-buddha-500 text-buddha-950"
onPress={handleSubmit}>
            Create Task
          </Button>
        </div>
      </form>
    </div>
  </div>
```

3.6.3 Handles Changing The Task To Integer Code Segment

```
  const handleTaskTypeChange = (selectedType: string) => {
    setSelectedTaskType(selectedType);
    let typeValue;
    switch (selectedType) {
        case 'Test':
            typeValue = 1;
            break;
        case 'Quiz':
            typeValue = 2;
            break;
        case 'Assignment':
            typeValue = 3;
            break;
        case 'Project':
            typeValue = 4;
            break;
        case 'Lecture':
            typeValue = 5;
            break;
        case 'Reading':
```

```
            typeValue = 6;
            break;
        case 'Discussion':
            typeValue = 7;
            break;
        case 'Final':
            typeValue = 8;
            break;
        case 'Midterm':
            typeValue = 9;
            break;
        case 'Presentation':
            typeValue = 10;
            break;
        case 'Paper':
            typeValue = 11;
            break;
        default:
            typeValue = null;
    }
    setFormData(prevData => ({ ...prevData, tasktype: typeValue }));
};
```

3.6.4 Defining The Calendar Page Code Segment

```
    <>
    <div className="main-bg relative min-h-screen p-8 text-buddha-200">
      <main className="calendar-container">
        <div className="grid grid-cols-1 lg:grid-cols-2 h-full w-full
text-buddha-200">
          <div className="lg:col-span-2">
            <FullCalendar
              height="auto"
              plugins={[dayGridPlugin, interactionPlugin, timeGridPlugin]}
              headerToolbar={{
                left: 'prev,next today',
                center: 'title',
                right: 'dayGridMonth,dayGridWeek',
              }}
              events={[...freeTimeEvents, ...tasks]}
              eventContent={renderEventContent}
              eventClick={handleEventClick}
              initialView="dayGridMonth"
              nowIndicator={true}
              selectable={true}
              dayMaxEvents={true}
              weekends={true}
              select={handleDateSelect}
            />
          </div>
```

```
          <TaskModal
            isOpen={isTaskModalOpen}
              onClose={() => setIsTaskModalOpen(false)}
              selectedTask={selectedTask}
              modalMode="view"
          />
          <InModal
            isOpen={isModalOpen}
            onClose={() => setIsModalOpen(false)}
            selectedDate={selectedDate}
          />
        </div>
      </main>
      <br />
      <br />
      <div className="flex justify-center items-center">
          <button className="bg-buddha-500 hover:bg-buddha-200 text-buddha-950
py-4 px-6 rounded-full" onClick={updateTasksWithDates}>
              Update Tasks
          </button>
      </div>
      <br />
      <br />
      <Taskmenu />
    </div>
    </>
```

3.6.5 Defining The Interfaces Of Information Code Segment

```
interface Task {
  title: string;
  duedate: Date;
  start: Date;
  end: Date;
  priorityof: number;
  estimatedtime: number;
  taskid: string;
  tasktype: number;
  numdays: number;
  eventHeight: number;
  color: string;
  recursion: boolean;
  recursionDetails?: Recursion;
  importance: number;
  statusof: string;
  timeleft: number;
  isrecurringadded: boolean;
}

interface Recursion {
  recursionid: number;
```

```
  taskid: number;
  frequencycycle: number;
  repetitioncycle: number;
  cyclestartdate: Date;
}
```

3.6.6 Open Source Full Calendar API

```
  <FullCalendar
              height="auto"
              plugins={[dayGridPlugin, int eractionPlugin, timeGridPlugin]}
              headerToolbar={{
                left: 'prev,next today',
                center: 'title',
                right: 'dayGridMonth,dayGridWeek',
              }}
              events={[...freeTimeEvents, ...tasks]}
              eventContent={renderEventContent}
              eventClick={handleEventClick}
              initialView="dayGridMonth"
              nowIndicator={true}
              selectable={true}
              dayMaxEvents={true}
              weekends={true}
              select={handleDateSelect}
            />
```

3.6.7 Next.UI Button Component

```
import React from "react";
import { Link, Button as NextUIButton } from "@nextui-org/react";
import { useRouter } from 'next/navigation'; // Corrected the import

export function Button() {
//function to route to the add tasks page

return (
<div className="flex justify-end gap-4 items-center w-full"> {/* Added justify-end and
w-full */}
<NextUIButton className='bg-buddha-500' variant="flat">
<Link href="/calendar/tasks" aria-current="page" className='text-buddha-950'>
Add Task
</Link>
</NextUIButton>
</div>
```

```
);
}
```

3.6.8 Next.UI Input Modal For Creating Task Component

```
export function inModel({ isOpen, onClose, selectedDate}) {
const [isRecursive, setIsRecursive] = useState(false); // New state for recursion
checkbox
const [selectedImportance, setSelectedImportance] = useState<string | null>(null);
const [importanceValue, setImportanceValue] = useState<number | null>(null);
const handleImportanceChange = (label: string, value: number) => {
setSelectedImportance(label);
setImportanceValue(value);
// Update formData or any other state if needed
setFormData(prevData => ({ ...prevData, importance: value }));
};

const [freeTimeData, setFreeTimeData] = useState([{
freetimeid: '',
userid : '',
dayoffree: 0,
minutesavailable: 0,
}]);

const retrievedFreeTime = async () => {
const session = await supabase.auth.getSession();
if (session && session.data.session) {
const { data: retrievedFreeTime, error: retrieveFreeTimeError } = await supabase
.from('freetime')
.select('*')
.eq('userid', session.data.session.user?.id)

if (retrieveFreeTimeError) {
alert('Error retrieving freetime data: ' + retrieveFreeTimeError.message);
return;
}
// Map over the retrieved data to transform it into the desired format
const freeTimes = retrievedFreeTime.map(freeTime => ({
freetimeid: freeTime.freetimeid,
userid: freeTime.userid,
dayoffree: freeTime.dayoffree,
minutesavailable: freeTime.minutesavailable,
```

```
}));

setFreeTimeData(freeTimes);
}
}

useEffect(() => {
retrievedFreeTime();
}, []);

const calculateNumDays = () => {
//calculate the number of days needed to complete using the estimated time and the
free time
const estimatedTime = parseFloat(formData.estimatedtime);
const freeTimePerDay = freeTimeData;
let numDays = 0;
let timeLeft = estimatedTime;
let day = 0;
while (timeLeft > 0) {
if (day > 6) {
day = 0;
}
if (freeTimePerDay[day].minutesavailable > 0) {
timeLeft -= freeTimePerDay[day].minutesavailable;
numDays++;
}
day++;
}
return numDays;
}

const handlePriority = (numdays) => {
const importance = parseFloat(formData.importance);
const timeNeeded = parseFloat(formData.estimatedtime);
const dueDate = new Date(selectedDate);
const today = new Date();
const status = formData.statusof;
const taskType = formData.tasktype;
const timeLeft = formData.timeleft;
const daysThoughtNeeded = numdays;
const freeTimePerDay = freeTimeData;
```

```
const calculatedUrgency = prioritizeTasks(taskType, dueDate, today, status,
freeTimePerDay, timeNeeded, daysThoughtNeeded, importance, timeLeft);

return calculatedUrgency;
}

let formattedDate;
if (selectedDate) {
formattedDate = selectedDate.toISOString().split('T')[0];
}
const [formData, setFormData] = useState({
taskname: '',
tasktype: '',
estimatedtime: '',
timeleft: '',
priorityof: '0',
statusof: 'Not Started',
numdays: 0,
recursion: false,
frequencycycle: '',
repetitioncycle: '',
cyclestartdate: '',
importance: 0,
});

const handleCheckboxChange = () => {
setIsRecursive(!isRecursive);
setFormData({ ...formData, recursion: true});
};

const handleChange = (e) => {
const { name, value } = e.target;
setFormData((prevData) => ({ ...prevData, [name]: value }));
};

const handleSubmit = async () => {

const numDays = calculateNumDays();

const urgency = handlePriority(numDays);
// Insert the new task into the 'tasks' table
const session = await supabase.auth.getSession();
```

```
if (session && session.data.session) {
const {data} = await supabase
.from('tasks')
.insert([{
userid: session.data.session.user?.id,
taskname: formData.taskname,
tasktype: formData.tasktype,
duedate: formattedDate,
estimatedtime: formData.estimatedtime,
timeleft: formData.estimatedtime,
priorityof: urgency,
statusof: formData.statusof,
numdays: numDays,
recursion: formData.recursion,
importance: formData.importance,
}]);
} else {
alert('Error inserting task data: ' + Error.prototype.message);
return; // Exit early if there's an error
}

// Retrieve the taskid of the newly inserted task
const { data: retrievedTask, error: retrieveError } = await supabase
.from('tasks')
.select('taskid')
.eq('taskname', formData.taskname)
.eq('tasktype', formData.tasktype)
.eq('duedate', formattedDate)
.eq('estimatedtime', formData.estimatedtime)
.eq('timeleft', formData.estimatedtime)
.eq('priorityof', urgency)
.eq('statusof', formData.statusof)
.eq('numdays', numDays)
.eq('recursion', formData.recursion)
.single();

if (retrieveError) {
alert('Error retrieving task data: ' + retrieveError.message);
return;
}

const taskid = retrievedTask.taskid;
```

```
if (isRecursive) {
// If the task is recursive, insert into the 'recursion' table
const { error: recursionError } = await supabase
.from('recursion')
.insert([{
taskid: taskid,
frequencycycle: formData.frequencycycle,
repetitioncycle: formData.repetitioncycle,
cyclestartdate: formData.cyclestartdate,
}]);
if (recursionError) {
alert('Error inserting recursion data: ' + recursionError.message);
return; // Exit early if there's an error
}
}
alert('Task Created Successfully!');
onClose();
//refresh the page
window.location.reload();
}

const [selectedTaskType, setSelectedTaskType] = useState<string | null>(null);
const handleTaskTypeChange = (selectedType: string) => {
setSelectedTaskType(selectedType);
let typeValue;
switch (selectedType) {
case 'Test':
typeValue = 1;
break;
case 'Quiz':
typeValue = 2;
break;
case 'Assignment':
typeValue = 3;
break;
case 'Project':
typeValue = 4;
break;
case 'Lecture':
typeValue = 5;
break;
case 'Reading':
```

```
typeValue = 6;
break;
case 'Discussion':
typeValue = 7;
break;
case 'Final':
typeValue = 8;
break;
case 'Midterm':
typeValue = 9;
break;
case 'Presentation':
typeValue = 10;
break;
case 'Paper':
typeValue = 11;
break;
default:
typeValue = null;
}
setFormData(prevData => ({ ...prevData, tasktype: typeValue }));
};

return (
<>
{isOpen && (
<Modal
isOpen={isOpen}
onClose={onClose}
placement="top-center"
>
<ModalContent>
<>
<ModalHeader className="flex flex-col gap-1">Create A New Task</ModalHeader>
<ModalBody>
<label htmlFor="taskname" className="block text-sm font-medium text-buddha-950">
Task Name
</label>
<input
type="text"
id="taskname"
name="taskname"
```

```
className="mt-1 p-2 w-full border border-buddha-950 rounded-md"
placeholder="Enter task name"
onChange={handleChange}
required
/>
<label className="block text-sm font-medium text-buddha-950">
Task Type
</label>
<Dropdown>
<DropdownTrigger>
<Button variant="flat" className='bg-buddha-500 text-buddha-950'>
{selectedTaskType ? selectedTaskType : 'Select Task Type'}
</Button>
</DropdownTrigger>
<DropdownMenu aria-label="Task Types">
<DropdownItem key="type1" onClick={() =>
handleTaskTypeChange('Test')}>Test</DropdownItem>
<DropdownItem key="type2" onClick={() =>
handleTaskTypeChange('Quiz')}>Quiz</DropdownItem>
<DropdownItem key="type3" onClick={() =>
handleTaskTypeChange('Assignment')}>Assignment</DropdownItem>
<DropdownItem key="type4" onClick={() =>
handleTaskTypeChange('Project')}>Project</DropdownItem>
<DropdownItem key="type5" onClick={() =>
handleTaskTypeChange('Lecture')}>Lecture</DropdownItem>
<DropdownItem key="type6" onClick={() =>
handleTaskTypeChange('Reading')}>Reading</DropdownItem>
<DropdownItem key="type7" onClick={() =>
handleTaskTypeChange('Discussion')}>Discussion</DropdownItem>
<DropdownItem key="type8" onClick={() =>
handleTaskTypeChange('Final')}>Final</DropdownItem>
<DropdownItem key="type9" onClick={() =>
handleTaskTypeChange('Midterm')}>Midterm</DropdownItem>
<DropdownItem key="type10" onClick={() =>
handleTaskTypeChange('Presentation')}>Presentation</DropdownItem>
<DropdownItem key="type11" onClick={() =>
handleTaskTypeChange('Paper')}>Paper</DropdownItem>
</DropdownMenu>
</Dropdown>
<label htmlFor="duedate" className="block text-sm font-medium text-buddha-950">
Due Date
</label>
```

```
<div className="mt-1 p-2 w-full border border-buddha-950 rounded-md">
{selectedDate.toISOString().split('T')[0]}
</div>
<label htmlFor="estimatedtime" className="block text-sm font-medium text-buddha-950">
Estimated Time
</label>
<input
type="number"
id="estimatedtime"
name="estimatedtime"
className="mt-1 p-2 w-full border border-buddha-950 rounded-md"
placeholder="Enter Estimated Minutes Needed"
onChange={handleChange}
required
/>
<label className="block text-sm font-medium text-buddha-950">
Task Importance
</label>
<Dropdown>
<DropdownTrigger>
<Button variant="flat" className='bg-buddha-500 text-buddha-950'>
{selectedImportance ? selectedImportance : 'Select Importance'}
</Button>
</DropdownTrigger>
<DropdownMenu aria-label="Task Importance">
<DropdownItem key="importance1" onClick={() => handleImportanceChange('Low
Importance', 0.25)}>Low Importance</DropdownItem>
<DropdownItem key="importance2" onClick={() => handleImportanceChange('Medium
Importance', 0.5)}>Medium Importance</DropdownItem>
<DropdownItem key="importance3" onClick={() => handleImportanceChange('High
Importance', 0.75)}>High Importance</DropdownItem>
<DropdownItem key="importance4" onClick={() => handleImportanceChange('ASAP',
1)}>Critical Importance</DropdownItem>
</DropdownMenu>
</Dropdown>
<label htmlFor="recursion" className="block text-sm font-medium text-buddha-950">
Task Recursion
</label>
<Checkbox onChange={handleCheckboxChange} />
{isRecursive && (
<>
<label htmlFor="cyclestartdate" className="block text-sm font-medium text-buddha-950">
```

```
Recursion Start Date
</label>
<input
type="date"
id="cyclestartdate"
name="cyclestartdate"
className="mt-1 p-2 w-full border border-buddha-950 rounded-md"
placeholder="Enter Start Date"
onChange={handleChange}
required
/>
<label htmlFor="repetitioncycle" className="block text-sm font-medium
text-buddha-950">
Recursion Cycle Period
</label>
<input
type="number"
id="repetitioncycle"
name="repetitioncycle"
className="mt-1 p-2 w-full border border-buddha-950 rounded-md"
placeholder="Enter Number Of Days In Period"
onChange={handleChange}
required
/>
<label htmlFor="frequencycycle" className="block text-sm font-medium text-buddha-950">
Frequency of Recursions
</label>
<input
type="number"
id="frequencycycle"
name="frequencycycle"
className="mt-1 p-2 w-full border border-buddha-950 rounded-md"
placeholder="Enter Number Of Repititions in Period"
onChange={handleChange}
required
/>
</> )}
</ModalBody>
<ModalFooter>
<Button color="danger" variant="flat" onPress={onClose}>
Close
</Button>
```

```
<Button className="bg-buddha-500 text-buddha-950" onPress={handleSubmit}>
Add
</Button>
</ModalFooter>
</>
</ModalContent>
</Modal>
)}
</>
);
}

export default inModel;
```

3.6.9 Next.UI Task Modal For View/ Edit/ Delete Component

```
export function taskModal({ isOpen, onClose, selectedTask, modalMode: initialModalMode
}) {
const [isRecursive, setIsRecursive] = useState(false);
const [isEditMode, setIsEditMode] = useState(false);
const [modalMode, setModalMode] = useState<'view' | 'edit' |
'delete'>(initialModalMode || 'view');
const [selectedImportance, setSelectedImportance] = useState<string | null>(null);
const [importanceValue, setImportanceValue] = useState<number | null>(null);
const [urgency, setUrgency] = useState<number>(0);
const handleImportanceChange = (label: string, value: number) => {
setSelectedImportance(label);
setImportanceValue(value);
// Update formData or any other state if needed
setFormData(prevData => ({ ...prevData, importance: value }));
};

const [freeTimeData, setFreeTimeData] = useState([{
freetimeid: '',
userid : '',
dayoffree: 0,
minutesavailable: 0,
}]);

const retrievedFreeTime = async () => {
const session = await supabase.auth.getSession();
if (session && session.data.session) {
```

```
const { data: retrievedFreeTime, error: retrieveFreeTimeError } = await supabase
.from('freetime')
.select('*')
.eq('userid', session.data.session.user?.id)

if (retrieveFreeTimeError) {
alert('Error retrieving freetime data: ' + retrieveFreeTimeError.message);
return;
}
// Map over the retrieved data to transform it into the desired format
const freeTimes = retrievedFreeTime.map(freeTime => ({
freetimeid: freeTime.freetimeid,
userid: freeTime.userid,
dayoffree: freeTime.dayoffree,
minutesavailable: freeTime.minutesavailable,
}));

setFreeTimeData(freeTimes);
}
}

useEffect(() => {
retrievedFreeTime();
}, []);

const calculateNumDays = () => {
//calculate the number of days needed to complete using the estimated time and the
free time
const estimatedTime = parseFloat(formData.estimatedtime);
const freeTimePerDay = freeTimeData;
let numDays = 0;
let timeLeft = estimatedTime;
let day = 0;
while (timeLeft > 0) {
if (day > 6) {
day = 0;
}
if (freeTimePerDay[day].minutesavailable > 0) {
timeLeft -= freeTimePerDay[day].minutesavailable;
numDays++;
}
day++;
```

```
}
return numDays;
}

const handlePriority = (numdays) => {
const importance = parseFloat(formData.importance);
const timeNeeded = parseFloat(formData.estimatedtime);
const dueDate = new Date(formData.duedate);
const today = new Date();
const status = formData.statusof;
const taskType = formData.tasktype;
const timeLeft = formData.timeleft;
const daysThoughtNeeded = numdays;
const freeTimePerDay = freeTimeData;
const calculatedUrgency = prioritizeTasks(taskType, dueDate, today, status,
freeTimePerDay, timeNeeded, daysThoughtNeeded, importance, timeLeft);

return calculatedUrgency;
}

const handleCheckboxChange = async () => {
if (!formData.recursion) {
setIsRecursive(true);
setFormData(prevData => ({ ...prevData, recursion: true }));
try {
await insertRecursionData();
} catch (error) {
alert('Error inserting recursion data: ' + error.message);
// Revert UI changes if there's an error
setIsRecursive(false);
setFormData(prevData => ({ ...prevData, recursion: false }));
}
} else {
setIsRecursive(false);
setFormData(prevData => ({ ...prevData, recursion: false }));
try {
await deleteRecursionData();
} catch (error) {
alert('Error deleting recursion data: ' + error.message);
// Revert UI changes if there's an error
setIsRecursive(true);
setFormData(prevData => ({ ...prevData, recursion: true }));
```

```
}
}
};
const insertRecursionData = async () => {
const { error } = await supabase
.from('recursion')
.insert([{
taskid: selectedTask,
frequencycycle: formData.frequencycycle,
repetitioncycle: formData.repetitioncycle,
cyclestartdate: formData.cyclestartdate,
}]);
if (error) throw error;
};
const deleteRecursionData = async () => {
const { error } = await supabase
.from('recursion')
.delete()
.eq('taskid', selectedTask);
if (error) throw error;
};

useEffect(() => {
setModalMode(initialModalMode);
}, [initialModalMode]);

const [formData, setFormData] = useState({
taskname: '',
tasktype: '',
duedate: '',
estimatedtime: '',
timeleft: '',
priorityof: '0',
statusof: 'Not Started',
numdays: 0,
recursion: false,
frequencycycle: '',
repetitioncycle: '',
cyclestartdate: '',
importance: 0
});
```

```
useEffect(() => {
const fetchTaskData = async () => {
const { data: taskData, error: taskError } = await supabase
.from('tasks')
.select('*')
.eq('taskid', selectedTask)
.single();

if (taskError) {
alert('Error fetching task data: ' + taskError.message);
return;
}

if (taskData) {
setFormData(taskData);
if (taskData.recursion === true) {
setIsRecursive(true);
const { data: recursionData, error: recursionError } = await supabase
.from('recursion')
.select('frequencycycle, repetitioncycle, cyclestartdate')
.eq('taskid', selectedTask);
if (recursionError) {
alert('Error fetching recursion data: ' + recursionError.message);
return;
}
if (recursionData && recursionData.length === 1) {
const recursionDetails = recursionData[0];
setFormData(prevData => ({
...prevData,
frequencycycle: recursionDetails.frequencycycle,
repetitioncycle: recursionDetails.repetitioncycle,
cyclestartdate: recursionDetails.cyclestartdate
}));
}
} else {
setIsRecursive(false);
}
}
};

if (selectedTask) {
fetchTaskData();
```

```
}
}, [selectedTask]);

const toggleEditMode = () => {
setIsEditMode(!isEditMode);
};

const handleInputChange = (field, value) => {
setFormData(prevData => ({
...prevData,
[field]: value
}));
};

const saveEditedTask = async () => {

// Calculate the number of days needed to complete the task
const numDays = calculateNumDays();

// Calculate the priority of the task
const urgency = handlePriority(numDays);

const { error } = await supabase
.from('tasks')
.update({
taskname: formData.taskname,
tasktype: formData.tasktype,
duedate: formData.duedate,
estimatedtime: formData.estimatedtime,
priorityof: urgency,
statusof: formData.statusof,
numdays: numDays,
recursion: formData.recursion,
importance: formData.importance
})
.eq('taskid', selectedTask);
if (error) {
alert('Error updating task: ' + error.message);
return;
}
// If there's recursion data, update that as well
if (isRecursive) {
```

```
const { error: recursionError } = await supabase
.from('recursion')
.update({
frequencycycle: formData.frequencycycle,
repetitioncycle: formData.repetitioncycle,
cyclestartdate: formData.cyclestartdate
})
.eq('taskid', selectedTask);
if (recursionError) {
alert('Error updating recursion data: ' + recursionError.message);
return;
}
}
// Close the modal and refresh the data (or however you want to handle it)
onClose();
window.location.reload();
// Optionally, you can fetch the tasks again or use any other method to refresh the
data
};

const handleDelete = async () => {
try {
// 1. Check if the task has recursion data
if (isRecursive) {
// 2. Delete the recursion data
const { error: recursionDeleteError } = await supabase
.from('recursion')
.delete()
.eq('taskid', selectedTask);
if (recursionDeleteError) throw recursionDeleteError;
}
// 3. Delete the task
const { error: taskDeleteError } = await supabase
.from('tasks')
.delete()
.eq('taskid', selectedTask);
if (taskDeleteError) throw taskDeleteError;
onClose();
window.location.reload(); // or use any other method to refresh the data
} catch (error) {
alert('Error deleting task: ' + error.message);
}
```

```
};

const [selectedTaskType, setSelectedTaskType] = useState<string | null>(null);
const handleTaskTypeChange = (selectedType: string) => {
setSelectedTaskType(selectedType);
let typeValue;
switch (selectedType) {
case 'Test':
typeValue = 1;
break;
case 'Quiz':
typeValue = 2;
break;
case 'Assignment':
typeValue = 3;
break;
case 'Project':
typeValue = 4;
break;
case 'Lecture':
typeValue = 5;
break;
case 'Reading':
typeValue = 6;
break;
case 'Discussion':
typeValue = 7;
break;
case 'Final':
typeValue = 8;
break;
case 'Midterm':
typeValue = 9;
break;
case 'Presentation':
typeValue = 10;
break;
case 'Paper':
typeValue = 11;
break;
default:
typeValue = null;
```

```
}
setFormData(prevData => ({ ...prevData, tasktype: typeValue }));
};

return (
<>
{isOpen && (
<Modal isOpen={isOpen} onClose={onClose} placement="top-center">
<ModalContent className="bg-clear p-6 rounded-lg shadow-lg w-full max-w-xl mx-auto">
{modalMode === 'view' && (
<>
<ModalHeader className="flex flex-col gap-1 text-buddha-950 border-b pb-2 mb-4">View
Task</ModalHeader>
<div className="mb-4 space-y-2">
<p><strong>Task Name:</strong> {formData.taskname}</p>
<p><strong>Task Type:</strong> {formData.tasktype}</p>
<p><strong>Due Date:</strong> {formData.duedate}</p>
<p><strong>Estimated Time:</strong> {formData.estimatedtime}</p>
<p><strong>Priority:</strong> {formData.priorityof}</p>
<p><strong>Status:</strong> {formData.statusof}</p>
<p><strong>Number of Days:</strong> {formData.numdays}</p>
<p><strong>Recursion:</strong> {formData.recursion ? 'True' : 'False'}</p>
{isRecursive && (
<>
<p><strong>Frequency Cycle:</strong> {formData.frequencycycle}</p>
<p><strong>Repetition Cycle:</strong> {formData.repetitioncycle}</p>
<p><strong>Cycle Start Date:</strong> {formData.cyclestartdate}</p>
</>
)}
</div>
<div className="flex justify-end space-x-2 mt-4">
<Button color="danger" variant="flat" onPress={() => setModalMode('delete')}
className="px-4 py-2">
Delete
</Button>
<Button variant='shadow' onPress={() => setModalMode('edit')} className="bg-buddha-500
text-buddha-950 px-4 py-2">
Edit
</Button>
</div>
</>
)}
```

```
{modalMode === 'edit' && (
<>
<ModalHeader className="flex flex-col gap-1 text-buddha-950 border-b pb-2 mb-4">Edit
Task</ModalHeader>
<div className="mb-4 space-y-4">
<div>
<label className="block text-sm font-medium text-gray-700 mb-1">Task Name:</label>
<input className="border p-2 rounded w-full" type="text" value={formData.taskname}
onChange={e => handleInputChange('taskname', e.target.value)} />
</div>
<div>
<label className="block text-sm font-medium text-buddha-950">
Task Type
</label>
<Dropdown>
<DropdownTrigger>
<Button variant="flat" className='bg-buddha-500 text-buddha-950 w-full'>
{selectedTaskType ? selectedTaskType : 'Select Task Type'}
</Button>
</DropdownTrigger>
<DropdownMenu aria-label="Task Types">
<DropdownItem key="type1" onClick={() =>
handleTaskTypeChange('Test')}>Test</DropdownItem>
<DropdownItem key="type2" onClick={() =>
handleTaskTypeChange('Quiz')}>Quiz</DropdownItem>
<DropdownItem key="type3" onClick={() =>
handleTaskTypeChange('Assignment')}>Assignment</DropdownItem>
<DropdownItem key="type4" onClick={() =>
handleTaskTypeChange('Project')}>Project</DropdownItem>
<DropdownItem key="type5" onClick={() =>
handleTaskTypeChange('Lecture')}>Lecture</DropdownItem>
<DropdownItem key="type6" onClick={() =>
handleTaskTypeChange('Reading')}>Reading</DropdownItem>
<DropdownItem key="type7" onClick={() =>
handleTaskTypeChange('Discussion')}>Discussion</DropdownItem>
<DropdownItem key="type8" onClick={() =>
handleTaskTypeChange('Final')}>Final</DropdownItem>
<DropdownItem key="type9" onClick={() =>
handleTaskTypeChange('Midterm')}>Midterm</DropdownItem>
<DropdownItem key="type10" onClick={() =>
handleTaskTypeChange('Presentation')}>Presentation</DropdownItem>
```

```
<DropdownItem key="type11" onClick={() =>
handleTaskTypeChange('Paper')}>Paper</DropdownItem>
</DropdownMenu>
</Dropdown>
</div>
<div>
<label className="block text-sm font-medium text-gray-700 mb-1">Due Date:</label>
<input className="border p-2 rounded w-full" type="date" value={formData.duedate}
onChange={e => handleInputChange('duedate', e.target.value)} />
</div>
<div>
<label className="block text-sm font-medium text-gray-700 mb-1">Estimated
Time:</label>
<input className="border p-2 rounded w-full" type="number"
value={formData.estimatedtime} onChange={e => handleInputChange('estimatedtime',
e.target.value)} />
</div>
<div className="mb-4 ">
<label className="block text-sm font-medium text-buddha-950">
Task Importance
</label>
<Dropdown>
<DropdownTrigger>
<Button variant="flat" className='bg-buddha-500 text-buddha-950 w-full'>
{selectedImportance ? selectedImportance : 'Select Importance'}
</Button>
</DropdownTrigger>
<DropdownMenu aria-label="Task Importance">
<DropdownItem key="importance1" onClick={() => handleImportanceChange('Low
Importance', 0.25)}>Low Importance</DropdownItem>
<DropdownItem key="importance2" onClick={() => handleImportanceChange('Medium
Importance', 0.5)}>Medium Importance</DropdownItem>
<DropdownItem key="importance3" onClick={() => handleImportanceChange('High
Importance', 0.75)}>High Importance</DropdownItem>
<DropdownItem key="importance4" onClick={() => handleImportanceChange('ASAP',
1)}>Critical Importance</DropdownItem>
</DropdownMenu>
</Dropdown>
</div>
<div className="mb-4">
<label htmlFor="recursion" className="block text-sm font-medium text-buddha-950">
Task Recursion
```

```
</label>
<Checkbox
id="recursion"
checked={formData.recursion}
onChange={handleCheckboxChange}
/>
</div>
{isRecursive && (
<>
<label className="block mt-4">
<span className="text-gray-700">Frequency Cycle:</span>
<input
className="border p-2 rounded w-full"
type="text"
value={formData.frequencycycle}
onChange={e => handleInputChange('frequencycycle', e.target.value)}
/>
</label>
<label className="block mt-4">
<span className="text-gray-700">Repetition Cycle:</span>
<input
className="border p-2 rounded w-full"
type="text"
value={formData.repetitioncycle}
onChange={e => handleInputChange('repetitioncycle', e.target.value)}
/>
</label>
<label className="block mt-4">
<span className="text-gray-700">Cycle Start Date:</span>
<input
className="border p-2 rounded w-full"
type="date"
value={formData.cyclestartdate}
onChange={e => handleInputChange('cyclestartdate', e.target.value)}
/>
</label>
</>
)}
</div>
<div className="flex justify-end space-x-2 mt-4">
<Button color="danger" variant="flat" onPress={() => setModalMode('view')}
className="px-4 py-2">
```

```
Cancel
</Button>
<Button variant='shadow' className="bg-buddha-500 text-buddha-950 px-4 py-2"
onPress={saveEditedTask}>
Save
</Button>
</div>
</>
)}
{modalMode === 'delete' && (
<>
<ModalHeader className="flex flex-col gap-1 text-buddha-950 border-b pb-2 mb-4">Delete
Task</ModalHeader>
<ModalBody className="mb-4 text-red-600">
Are you sure you want to delete this task?
</ModalBody>
<div className="flex justify-end space-x-2 mt-4">
<Button color="danger" variant="flat" onPress={handleDelete} className="px-4 py-2">
Confirm Delete
</Button>
<Button variant='shadow' onPress={onClose} className="px-4 py-2 bg-buddha-200">
Cancel
</Button>
</div>
</>
)}
</ModalContent>
</Modal>
)}
</>
);
}

export default taskModal;
```

3.6.10 Next.UI Task Menu Component

```
const columns = [
{name: "TASKID", uid: "taskid"},
{name: "TASKNAME", uid: "taskname"},
{name: "TASKTYPE", uid: "tasktype"},
{name: "DUEDATE", uid: "duedate"},
```

```
{name: "ESTIMATEDTIME", uid: "estimatedtime"},
{name: "TIMELEFT", uid: "timeleft"},
{name: "PRIORITY", uid: "priorityof"},
{name: "STATUS", uid: "statusof"},
{name: "NUMDAYS", uid: "numdays"},
{name: "RECURSION", uid: "recursion"},
{name: "STARTDATE", uid: "startdate"},
{name: "ENDDATE", uid: "enddate"},
{name: "ACTIONS", uid: "actions"},
];


const statusColorMap: Record<string, ChipProps["color"]> = {
active: "success",
paused: "danger",
vacation: "warning",
};

type Task = {
taskid: number;
userid: string | null;
taskname: string;
tasktype: number;
duedate: Date;
estimatedtime: number;
timeleft: number;
priorityof: number;
statusof: string;
numdays: number | null;
recursion: boolean;
startdate: Date | null;
enddate: Date | null;
};

export default function taskMenu() {
const [error, setError] = useState(null);
const [userId, setUserId] = useState(null);
const tasksRef = useRef<Task[]>([]);
const [, forceUpdate] = useState({});
const [modalMode, setModalMode] = useState<'view' | 'edit' | 'delete'>('view');
const [isModalOpen, setIsModalOpen] = useState(false);
const [selectedTask, setSelectedTask] = useState<Task | null>(null);
```

```
const closeModal = () => {
setIsModalOpen(false);
setSelectedTask(null);
};

const openModalWithMode = (mode: 'view' | 'edit' | 'delete', task: Task) => {
setSelectedTask(task); // Set the selected task directly
setModalMode(mode); // Set the modal mode
setIsModalOpen(true); // Open the modal
};


useEffect(() => {
const retrieveUser = async () => {
const session = await supabase.auth.getSession();
if (session && session.data.session) {
const userId = session.data.session.user?.id;
setUserId(userId);
}
}
retrieveUser();
}, []);

useEffect(() => {
const fetchTasks = async () => {
if (!userId) return; // Don't fetch if userId is not set

try {
const { data, error } = await supabase
.from('tasks')
.select('*')
.eq('userid', userId);

if (error) throw error;

const tasksArray = data.map(task => ({
taskid: task.taskid,
userid: task.userid,
taskname: task.taskname,
tasktype: task.tasktype,
duedate: task.duedate,
```

```
estimatedtime: task.estimatedtime,
timeleft: task.timeleft,
priorityof: task.priorityof,
statusof: task.statusof,
numdays: task.numdays,
recursion: task.recursion,
startdate: task.startdate,
enddate: task.enddate,
}));

tasksRef.current = tasksArray;
forceUpdate({}); // Force a re-render

} catch (error) {
setError(error.message);
}
};

fetchTasks();
}, [userId]);

if (error) return <div>Error: {error}</div>;

const renderCell = React.useCallback((task: Task, columnKey: React.Key) => {
const cellValue = getKeyValue<Task, string | number | boolean>(task, columnKey as
keyof Task);

switch (columnKey) {
case "name":
return (
<User
avatarProps={{ radius: "lg", src: task.avatar }}
description={task.email}
name={String(cellValue)}
>
{task.email}
</User>
);
case "role":
return (
<div className="flex flex-col">
<p className="font-bold text-sm capitalize">{String(cellValue)}</p>
```

```
<p className="font-bold text-sm capitalize text-gray-400">{task.team}</p>
</div>
);
case "status":
return (
<Chip className="capitalize" color={statusColorMap[task.status]} size="sm"
variant="flat">
{String(cellValue)}
</Chip>
);
case "actions":
return (
<div className="relative flex items-center gap-2">
<Tooltip content="Task Details">
<span
className="text-lg text-gray-400 cursor-pointer hover:opacity-50"
onClick={() => openModalWithMode('view', task.taskid)}
>
<EyeIcon />
</span>
</Tooltip>
<Tooltip content="Edit Task">
<span
className="text-lg text-gray-400 cursor-pointer hover:opacity-50"
onClick={() => openModalWithMode('edit', task.taskid)}
>
<EditIcon />
</span>
</Tooltip>
<Tooltip color="danger" content="Delete Task">
<span
className="text-lg text-red-500 cursor-pointer hover:opacity-50"
onClick={() => openModalWithMode('delete', task.taskid)}
>
<DeleteIcon />
</span>
</Tooltip>
</div>
);
case "recursion":
return cellValue ? "True" : "False";
default:
```

```
return String(cellValue);
}
}, []);

return (
<div>
<Table className="text-buddha-950" aria-label="Example table with custom cells">
<TableHeader columns={columns}>
{(column) => (
<TableColumn key={column.uid} align={column.uid === "actions" ? "center" : "start"}>
{column.name}
</TableColumn>
)}
</TableHeader>
<TableBody items={tasksRef.current}>
{(item) => (
<TableRow key={item.taskid}>
{(columnKey) => <TableCell>{renderCell(item, columnKey)}</TableCell>}
</TableRow>
)}
</TableBody>
</Table>
<br />
<Button />
<TaskModal isOpen={isModalOpen} onClose={closeModal} selectedTask={selectedTask}
modalMode={modalMode} />
</div>
);
}
```

3.6.10 Next.UI Navbar Component

```
export default function App() {
const [userName, setUserName] = useState(null);

// useEffect(() => {
// // Function to fetch user name
// const fetchUserName = async () => {
// const user = supabase.auth.getUser();
// if (user) {
// const userid = user.id;
// const { data, error } = await supabase
```

```
// .from('users')
// .select('firstname, lastname')
// .eq('userid', userid);
// const userFirstName = data.firstname;
// const userLastName = data.lastname;
// setUserName(userFirstName + ' ' + userLastName);
// }
// };

// fetchUserName(); // Fetch user name on initial load

// // You can refresh the user name periodically (e.g., every few hours) using a timer
if needed.
// const refreshInterval = setInterval(() => {
// fetchUserName();
// }, 3 * 60 * 60 * 1000); // Refresh every 3 hours (adjust as needed)

// // Cleanup the interval to avoid memory leaks
// return () => clearInterval(refreshInterval);
// }, []);

return (
<Navbar shouldHideOnScroll className='bg-buddha-950'>
<NavbarBrand>
<ChronoLogo />
<p className="text-buddha-200" style={{ fontSize: '25px', lineHeight: '14px', margin:
'0', padding: '10px'}}>ChronoHawk</p>
</NavbarBrand>
<NavbarContent className="hidden sm:flex gap-4" justify="center">
<NavbarItem>
<Link color="foreground" href="/" className='text-buddha-200'>
Home
</Link>
</NavbarItem>
<NavbarItem isActive>
<Link href="/calendar" aria-current="page" className='text-buddha-500'>
Calendar
</Link>
</NavbarItem>
<NavbarItem >
<Link color="foreground" href="/account" className='text-buddha-200'>
Account
```

```
</Link>
</NavbarItem>
<NavbarItem >
<Link color="foreground" href="/account/new/freetime" className='text-buddha-200'>
FreeTime
</Link>
</NavbarItem>
</NavbarContent>
<NavbarContent justify="end">
{userName == null ? (
<>
<NavbarItem className="hidden lg:flex">
<Link href="/account/existing" className='text-buddha-200'>Login</Link>
</NavbarItem>
<NavbarItem>
<Button as={Link} href="/account/new" variant="flat" className='text-buddha-800
bg-buddha-200'>
Sign Up
</Button>
</NavbarItem>
</>
):(
<>
<NavbarItem >
<Button className='text-buddha-950 bg-buddha-500' as={Link} href="/account">Welcome,
{userName}</Button>
</NavbarItem>
</>
)}
</NavbarContent>
</Navbar>
);
}
```

4. Discussions

4.1 Strengths

Entering this project, our combined expertise provided us with a firm foundation to tackle various challenges. One of the distinct advantages we held was during the developmental phase, particularly in crafting the UML diagram. Given our shared background in database systems, our understanding of database design was both deep and comprehensive. Drawing out the UML and transitioning to creating the database from scratch was a testament to our grasp of design and requirements. Furthermore, our proficiency in SQL was another notable strength. This experience allowed us to navigate through the project with more ease and efficiency, making complex tasks more manageable and streamlined.

4.2 Weaknesses

Despite our prior knowledge and competencies, we recognized certain areas of improvement as we progressed through the project. Our primary focus was on ensuring the functionality of the project, which inadvertently led to the oversight of vital security measures. As a result, our project remains more vulnerable to potential breaches. Additionally, our venture into using NextUI posed challenges. Both of us were navigating unfamiliar terrain since we hadn't previously worked with NextUI. This learning curve impacted our pace, and had we been more acquainted with NextUI from the onset, we could have expedited our progress. Further, our expertise in UI design could have been more robust, as certain aspects of design and layout presented hurdles. Routing was another area that required more attention and refinement. Reflecting on these experiences, we acknowledge the importance of balancing functionality with

security and the necessity of familiarizing oneself with tools and frameworks at the beginning of a project.

4.3 Niches

This project stood out not only for its ambitious goals but also for its unique niches that set it apart from typical software undertakings. The meticulous process of handcrafting the UML diagram and building a database from scratch underscored our commitment to precision and bespoke solutions. While our project combined conventional database design principles with modern UI frameworks, it also ventured into uncharted territories, experimenting with innovative approaches and tools. These niches added layers of complexity but also enriched our project with distinctive features, making it a testament to our adaptability, creativity, and drive to push boundaries.

4.4 Learning Experience

Embarking on this project was a profound journey of discovery and growth. Each challenge we faced presented an invaluable opportunity to expand our skill set and gain deeper insights into the complexities of software development. From navigating unfamiliar frameworks like NextUI to addressing the intricate aspects of security and UI design, every hurdle shaped our understanding and honed our problem-solving abilities. Moreover, collaborating closely allowed us to leverage our combined expertise, share diverse perspectives, and learn from each other's experiences. As we reflect upon our journey, it's evident that the knowledge and skills we've amassed far exceed the project's confines, preparing us for future endeavors in the ever-evolving tech landscape.

5. References

[1] Mozilla, “HTML basics - Learn web development,” *developer.mozilla.org*, 2019. [Online]. Available: https://developer.mozilla.org/en-US/docs/Learn/Getting_started_with_the_web/HTML_basics. [Accessed Sep. 27, 2023].

[2] Mozilla, “CSS: Cascading Style Sheets,” *developer.mozilla.org*, 2019. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/CSS. [Accessed Sep. 27, 2023].

[3] Mozilla, “JavaScript | MDN,” *developer.mozilla.org*, 2019. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/JavaScript. [Accessed Sep. 27, 2023].

[4] Microsoft, “TypeScript: JavaScript With Syntax For Types,” *typescriptlang.org*, 2015. [Online]. Available: https://www.typescriptlang.org/. [Accessed Sep. 27, 2023].

[5] Microsoft, “Documentation for Visual Studio Code,” *visualstudio.com*, Nov. 03, 2021. [Online]. Available: https://code.visualstudio.com/docs. [Accessed Sep. 26, 2023].

[6] Microsoft, “Setting up Visual Studio Code,” *visualstudio.com*, Nov. 03, 2021. [Online]. Available: https://code.visualstudio.com/docs/setup/setup-overview. [Accessed Sep. 26, 2023].

[7] Microsoft, “Using Git source control in VS Code,” *visualstudio.com*, Nov. 03, 2021. [Online]. Available: https://code.visualstudio.com/docs/sourcecontrol/overview. [Accessed Sep. 26, 2023].

[8] Git, “Git Basics Episode 1,” *git-scm.com*. [Online]. Available: https://git-scm.com/video/what-is-version-control. [Accessed Sep. 26, 2023].

[9] Visual Studio Code. "Using Git with Visual Studio Code (Official Beginner Tutorial)," *YouTube*, May 27, 2022 [Video file]. Available: https://www.youtube.com/watch?v=i_23KUAEtUM. [Accessed: Sep. 26, 2023].

[10] Microsoft, “Collaborate with Live Share,” *visualstudio.com*, Nov. 03, 2021. [Online]. Available: https://code.visualstudio.com/learn/collaboration/live-share. [Accessed Sep. 26, 2023].

[11] Meta Open Source, “React,” *react.dev*. [Online]. Available: https://react.dev/. [Accessed Sep. 27, 2023].

[12] Vercel, “Next.js by Vercel - The React Framework,” *nextjs.org*. [Online]. Available: https://nextjs.org/. [Accessed Sep. 27, 2023].

[13] J. Garcia, “Introduction,” *nextui.org*. [Online]. Available: https://nextui.org/docs/guide/introduction. [Accessed Sep. 27, 2023].

[14] N. Young, “The Pros and Cons of Tailwind CSS,” *webdesignerdepot.com*, Sep. 24, 2021. [Online]. Available: https://www.webdesignerdepot.com/2021/09/the-pros-and-cons-of-tailwind-css/. [Accessed Sep. 27, 2023].

[15] Adobe, “Buy Adobe Illustrator CC | Vector graphic design software,” *Adobe.com*, 2019. [Online]. Available: https://www.adobe.com/products/illustrator.html. [Accessed: Sep. 27, 2023]

[16] Learnit Training, “Illustrator Full Course Tutorial (6+ Hours),” *YouTube*, Aug. 22, 2022. [Video file]. Available: https://www.youtube.com/watch?v=3RTqLQ1MaQU. [Accessed: Sep. 27, 2023]

[17] Midjourney, “Midjourney Documentation and User Guide,” *docs.midjourney.com*. [Online]. Available: https://docs.midjourney.com/. [Accessed: Sep. 27, 2023]

[18] A. Mittal, “Mastering AI Art: A Concise Guide to Midjourney and Prompt Engineering - Unite.AI,” *www.unite.ai*, Jul. 27, 2023. [Online]. Available: https://www.unite.ai/mastering-ai-art-a-concise-guide-to-midjourney-and-prompt-engineering/. [Accessed: Sep. 27, 2023]

[19] Supabase, “Supabase Docs,” *supabase.com*. [Online]. Available: https://supabase.com/docs. [Accessed: Sep. 27, 2023]

[20] “What is SQL? - SQL - AWS,” *Amazon Web Services, Inc.* https://aws.amazon.com/what-is/sql/#:~:text=Structured%20query%20language%20(SQL)%20is

[21] “Documentation | FullCalendar,” *fullcalendar.io*. https://fullcalendar.io/docs

6. AI Usages

Leveraging the capabilities of AI greatly facilitated our development process. We seamlessly integrated AI-driven solutions to convert our preliminary functions into practical, implementable code. While we made significant alterations to adapt and refine the output, the AI-generated foundation served as a critical point of reference, shaping our vision for the final design. We often utilized prompts like "Show how to do this in next.js" to harness the AI's expertise in specific frameworks, enhancing our understanding and proficiency. Furthermore, AI became an indispensable resource for our research endeavors, helping us navigate the intricate web of contemporary web technologies and frameworks. Another pivotal aspect was the AI's error-checking capability. By feeding our code into the system with prompts such as "fix the errors," we were able to identify and rectify issues efficiently, ensuring our code remained robust and fault-free. This blend of human intuition and AI precision undoubtedly augmented the quality and efficiency of our work.

MidJourney Prompt For Background Image: a library setting filled with bright yellows and blacks with a focus on time that will be used for the background of my website and ensure there is a hawk flying around - Image #3