Stucado

NUS Orbital 2024

Splashdown Submission

Team 6006: Team April
Kong Hoi Tec
Ea Jing Le

Contents

Foreword

To whoever is reading this,

Thank you for taking the time to read our README report for Stucado, a study productivity app tailored for students, developed for Orbital 23/24. This report showcases the fruits of our labor and records the various hurdles and challenges we faced during the development of this project.

We begin with an overview of our motivation, vision, and aim for Stucado. User stories are included to help you understand our target audience and their needs. Following this, we present a gallery of our posters for each milestone.

Next, we delve into the technology and project management aspects of Stucado. Here, you will gain insight into many of our high-level decisions and the tools we used to make this project successful. The features section highlights our most prominent features, accompanied by our implementation philosophy and the challenges we faced during development.

We then discuss the software engineering principles we practiced and provide an in-depth look at our testing procedures, complete with case studies to illustrate our approach. Plenty of examples are given to demonstrate our points.

At the outset of this project, we encountered numerous challenges and often doubted ourselves. Countless nights were spent scouring the internet, reading old bug reports and documentation to fix persistent issues. Despite the difficulties, this has been an extremely fulfilling learning experience, and we are glad we persevered.

While Stucado may not be the most polished app, nor do we expect it to be an industry-changing product, it represents our first "large scale" coding project. We believe that the concept of a productivity app with machine learning features is novel and has the potential to be a brilliant productivity tool. We encourage others, whether fellow Orbitees or our future selves, to build upon what we have created.

Once again, thank you for reading our report. Special thanks to everyone behind the Orbital program, our advisor Aakash, our fellow Orbitees, and most importantly ourselves for making this project possible.

Austin and Jing Le,
NUS SoC Year 1 Students

Team Name

Team April

Proposed Level of Achievement

Apollo

Motivation

In today's fast-paced digital world, students face an array of challenges when it comes to managing their academic workload efficiently. With an ever-increasing list of tasks and responsibilities, the limited amount of time available often leads to stress and anxiety. The prevalence of social media and digital distractions only exacerbates these issues, making it easy for students to lose track of their priorities and fall into a cycle of procrastination. Furthermore, the pressure to excel academically adds to the burden, leaving many students feeling overwhelmed and unsure of where to begin.

Adding to the urgency of addressing these challenges is the concerning state of mental health among NUS students, as highlighted by recent news articles. The demands of academic life, coupled with the stressors of modern living, underscore the need for proactive solutions to support students in their academic journey while keeping them motivated.

While study apps are not uncommon, there remains a gap in the market for tools that provide personalized solutions to these issues. Despite the availability of many high-quality study apps, few effectively address the nuanced challenges faced by students in today's digital age.

This is where Stucado shines. As a data-driven productivity tool tailored to the needs of individual students, Stucado offers a unique solution to the challenges of time management and productivity.

Vision

By collecting and analyzing user data, Stucado provides personalized messages and recommendations to help students optimize their study habits and maximize their productivity.

With features such as customized to-do lists, timetable integration, and adaptive pomodoro timers, Stucado empowers students to take control of their academic journey with confidence. By leveraging technology to offer actionable insights and support, Stucado not only helps students excel academically but also promotes overall well-being.

Aim

Stucado aims to provide students a **data-driven** platform to optimize their daily productivity by constantly **collecting data and tweaking its algorithm**. Our goal is to transform the way students approach their studies by offering an intelligent, adaptive tool that not only helps them manage their time effectively but also enhances their overall learning experience.

We aim to alleviate the stress and anxiety associated with academic workload by providing a structured and supportive environment. By supercharging ordinary features such as timetables, to-do lists and pomodoro timers with the power of data, we aim to provide **actionable insights** and recommendations. This would help students **make informed decisions** about when and how to study, ensuring that they get the most out of their day.

Stucado is designed to be a reliable companion for students throughout their academic journey. By keeping all user data **stored locally** and ensuring **offline functionality**, the app respects users' privacy and provides uninterrupted support, regardless of internet connectivity. Our ultimate aim is to empower students to take control of their academic success with confidence, efficiency, and a sense of enjoyment.

User stories

1. As a user who enjoys personalizing my workspace, I want to be able to set up and customize my dashboard and make it visually appealing.
2. As a student who needs to stay on top of deadlines and tasks, I want to be able to create and manage a to-do list within the system, allowing me to prioritize and track my tasks effectively.
3. As a student at NUS who wants to efficiently manage my academic schedule and maximize my study time, I want to be able to import my NUS mods timetable into the system and fit all my tasks into available time slots, so I can easily view and organize my classes and tasks.
4. As someone who struggles with maintaining focus during work or study sessions, I want to be able to use a Pomodoro timer feature in the system to enhance productivity.
5. As a student who wants to optimize my schedule, I want the system to predict my productivity and provide me helpful insights and recommendations at different times of the day so I can plan my tasks for my most productive periods.
6. As a student who wants to quickly gauge my performance, I want short-term, specific statistics to be displayed on my dashboard, so I can see my recent study hours, productivity, and task completion at a glance.
7. As a student who wants to reflect on my study habits over time, I want to be able to view long term statistics so I can understand my productivity and study patterns over weeks.
8. As a student who wants to personalize my experience and manage my data, I want to be able to edit my settings and have the ability to delete my data.

Posters

Team 6006: Kong Hoi Tec & Ea Jing Le

Milestone 1 Poster

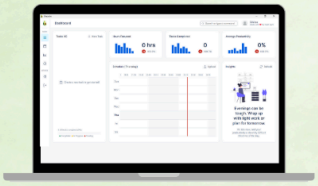
Stucado

Where study meets strategy

Team 6006: Team April
Kong Hoi Tec, Ea Jing Le

About

Stucado is a data-driven productivity tool tailored for NUS students to optimize their daily productivity by offering an intelligent, adaptive tool that helps them manage their time effectively and enhances their overall learning experience.



Problem Statement

Students today face major challenges in managing their academic workload effectively due to an increasing number of tasks, limited time, and digital distractions, leading to stress, anxiety, and decreased academic performance. Existing study apps fail to provide personalized support tailored to individual needs, leaving a gap in the market for tools that can help students prioritize tasks, manage time, and stay productive.

Objectives

- To provide a data-driven productivity tool for students.
- To offer personalized recommendations and optimize study habits.
- To enhance time management and productivity.
- To promote overall well-being and reduce academic stress.

Tech Stack

Front End
Back End
Database
Dev Tools
Testing

Software Engineering Principles

- Modularity and DRY: Reusable components and styles.
- Separation of Concerns: Distinct layers for presentation, service, application, data access, and data storage.
- Version Control: Git and GitHub for structured branching and collaboration.
- Code Quality: ESLint and Prettier for clean, consistent code.
- Code Testing: Jest and Cypress for unit, integration and system testing
- UI/UX Design: Wireframing and prototyping in Figma

Future Plans

Milestone 3:

- Fully implement remaining unit tests, integration and system testing
- Enhance adaptive features of the Pomodoro timer..
- Improve styling and personalization options for the timetable.
- Gamified experiences to engage users.

After Orbital:

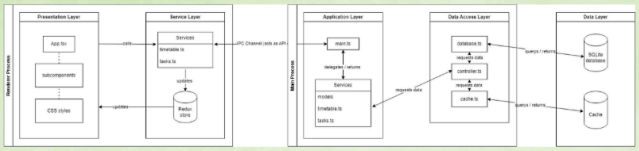
- Consider rewriting the app in React Native for mobile support and migrate the processing code to a dedicated backend server.

Testing

Unit tests, implemented for crucial frontend and backend logic using Jest. Core features working as expected, including task management, timetable integration, Pomodoro timer, and statistics visualization.

Integration and system testing, yet to be implemented as of Milestone 2.

User testing, collect feedback by reaching out to target audience (NUS students) outside of the development team. Identify and address usability issues and incorporate user suggestions to enhance user friendliness.



5 Layer System Architecture

Features

Integrated Timetable

Seamlessly integrate your NUS Mods timetable by pasting the URL into Stucado. Organize and optimize your study sessions with scrollable navigation, and utilize tools to create, edit, and clear timetable slots. Optimize your timetable to complete your daily tasks ASAP.

Descriptive Statistics

Track your productivity with heatmaps, line charts, and graphs showing focused hours, daily productivity scores, and tasks completed. View your current statistics and up to 30 days ago in the statistics page.

To-do List

Manage tasks effortlessly: add, edit or delete tasks, adjust estimated time with a user friendly time picker and keep track progress using an intuitive "traffic light" system. All tasks are auto-saved, and a progress bar shows task completion percentages.

Pomodoro Timer

Boost productivity with customizable session lengths and break durations. Receive system notifications for session transitions. Adaptive adjustments based on productivity will be implemented soon.

Onboarding

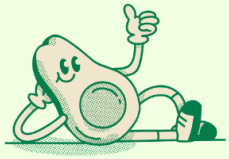
Uses are greeted with an onboarding page, where they can choose their app theme, upload a timetable from NUS Mods, and choose their study habit to initialize the model and enable insights from the start.

Settings Page

Customize your experience: switch themes, manage your data and test the app using experimental features such as generating random data and resetting the onboarding process.

Insights and Recommendations

Receive personalized motivational messages and data-driven suggestions. The system analyzes your productivity trends based on time of day, day of week, hours in classes and hours focused to offer actionable insights for you to optimize your study habits.



Milestone 2 Poster

Stucado

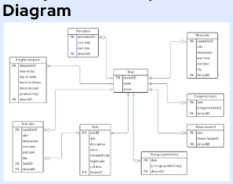
Where study meets strategy

Team 6006: Team April
Kong Hoi Tec, Ea Jing Le

About

Stucado is a data-driven productivity tool tailored for NUS students to optimize their daily productivity by offering an intelligent, adaptive tool that helps them manage their time effectively and enhances their overall learning experience.

Entity Relationship Diagram



Problem Statement

Students today face major challenges in managing their academic workload effectively due to an increasing number of tasks, limited time, and digital distractions, leading to stress, anxiety, and decreased academic performance. Existing study apps fail to provide personalized support tailored to individual needs, leaving a gap in the market for tools that can help students prioritize tasks, manage time, and stay productive.

Objectives

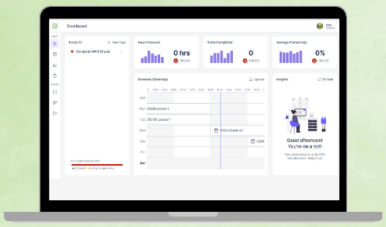
- To provide a data-driven productivity tool for students.
- To offer personalized recommendations and optimize study habits.
- To enhance time management and productivity.
- To promote overall well-being and reduce academic stress.

Tech Stack

Front End
Back End
Database
Dev Tools
Testing

Software Engineering Principles

- Modularity and DRY: Reusable components and styles.
- Separation of Concerns: Distinct layers for presentation, service, application, data access, and data storage.
- Version Control: Git and GitHub for structured branching and collaboration.
- Code Quality: ESLint and Prettier for clean, consistent code.
- Code Testing: Jest and Cypress for unit, integration and system testing
- UI/UX Design: Wireframing and prototyping in Figma



Testing

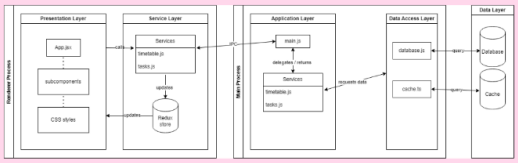
Unit Testing: Conducted with Jest to ensure the correctness of both frontend and backend components. This testing isolates individual features to verify their functionality and reliability, confirming that core aspects like task management and Pomodoro timers work as intended.

Integration Testing: Also performed with Jest, focusing on the interactions between different modules and services. It validates that components communicate effectively and data flows seamlessly across the application.

System Testing: Utilizes Playwright to simulate key user flows and test end-to-end functionality. This ensures that the application performs reliably with real user interactions.

User Testing: Involves collecting feedback from NUS students outside the development team. This feedback helps identify usability issues and drive improvements, enhancing the overall user experience.

System Architecture



Features

Onboarding

Uses are greeted with an onboarding page, where they can choose their app theme, upload a timetable from NUS Mods, and choose their study habit to initialize the productivity prediction model and enable insights from the start.

Integrated Timetable

Seamlessly integrate your NUS Mods timetable by pasting the URL into Stucado. Organize and optimize your study sessions with scrollable navigation, and utilize tools to create, edit, and clear timetable slots. Optimize your timetable to complete your daily tasks.

To-do List

Manage tasks effortlessly: add, edit or delete tasks, adjust estimated time with a user friendly time picker and keep track progress using an intuitive "traffic light" system. All tasks are auto-saved, and a progress bar shows task completion percentages.

Pomodoro Timer

Adaptive Pomodoro timer with adjusted session lengths based on productivity to manage your work efficiently. Boost productivity with customizable session lengths and break durations. Receive system notifications for session transitions.

Descriptive Statistics

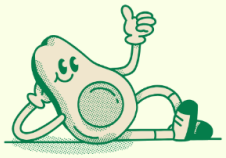
Track your productivity with heatmaps, line charts, and graphs showing focused hours, daily productivity scores, and tasks completed. View your short-term statistics in the homepage and long-term statistics and up to 30 days ago in the statistics page.

Settings Page

Customize your experience by tailoring your app's appearance, enhancing personal satisfaction, manage your data with option to delete all data. Link to Canvas or any website for quick access.

Insights and Recommendations

Receive personalized motivational messages and data-driven suggestions. The system analyzes your productivity trends based on time of day, day of week, hours in classes and hours focused to offer actionable insights for you to optimize your study habits.



Milestone 3 Poster

Technology & Project Management

Tech Stack

We chose to use Electron JS to develop our app as it provides a way to develop an application which can be deployed to Windows, Mac and Linux using web technologies. Since we both have minor experience using web technologies, we felt that this was a reasonable step to take.

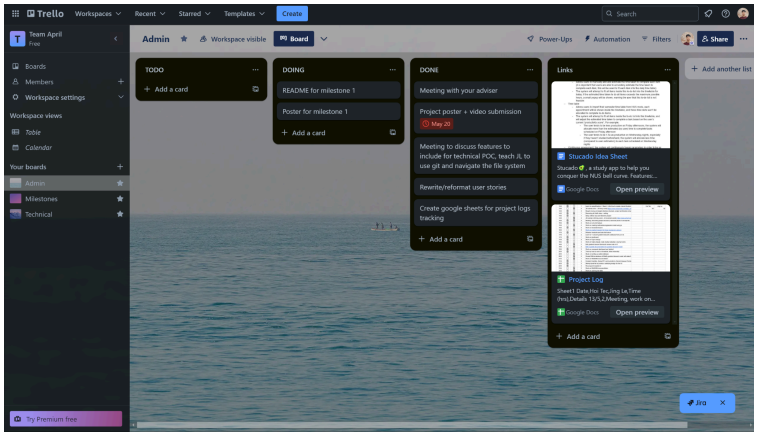
Initially, our choice was to use React and CSS for the frontend, and Typescript for the backend. This worked out fine, however deeper into the project we found that there were certain incompatibilities with Typescript, our database and some packages. That is why we decided to pivot and use plain Javascript instead. To ensure consistency, we have been using ESLint and Prettier to enforce some basic rules and code consistency

Moreover, we have chosen to use SQLite as our database. This is because it provides a lightweight and portable local database that can run fully offline in the users' machine. Since we also have experience working with SQL in the past, this is also another justified step in our eyes.

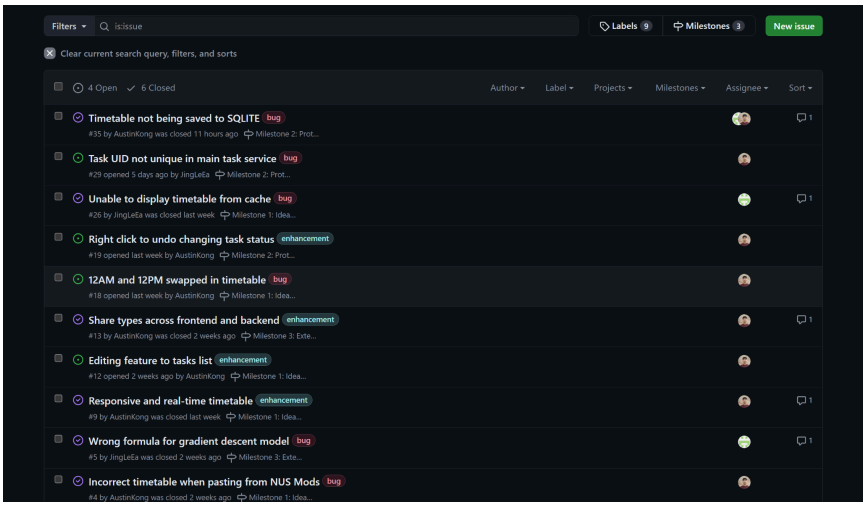
To write tests, we use Jest as our testing library for unit & integration testing. Originally, we planned to use Cypress as our system/e2e testing library since we have had minor experience using the library. However, we later found out that Electron support for Cypress has been discontinued. As such, we have pivoted to using Playwright instead.

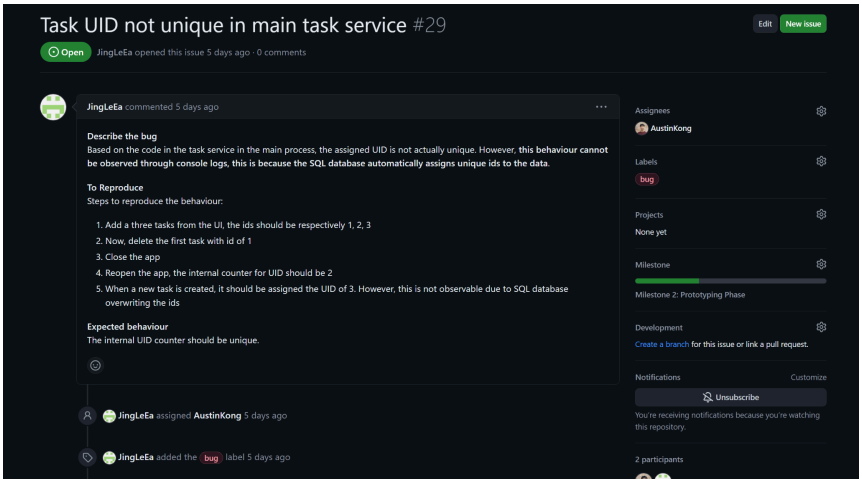
Finally, we use Vite as our development tool as it supports HMR (Hot Module Reloading). This allows us to rapidly prototype our app during development without the need to recompile at every change.

Project Management

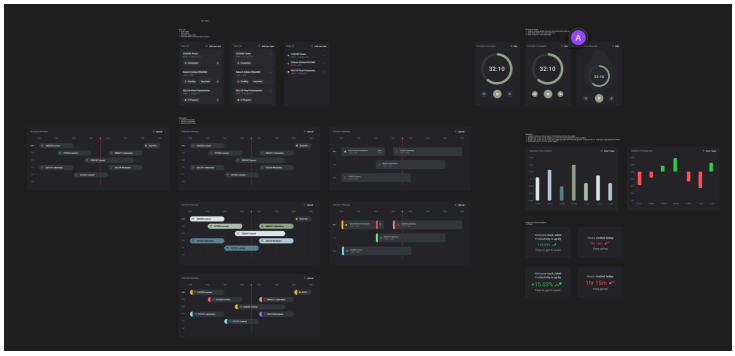


Our team uses Trello as our Kanban board to manage the progress of admin work (such as working on READMEs, posters etc.) and 10,000 feet milestone goals.

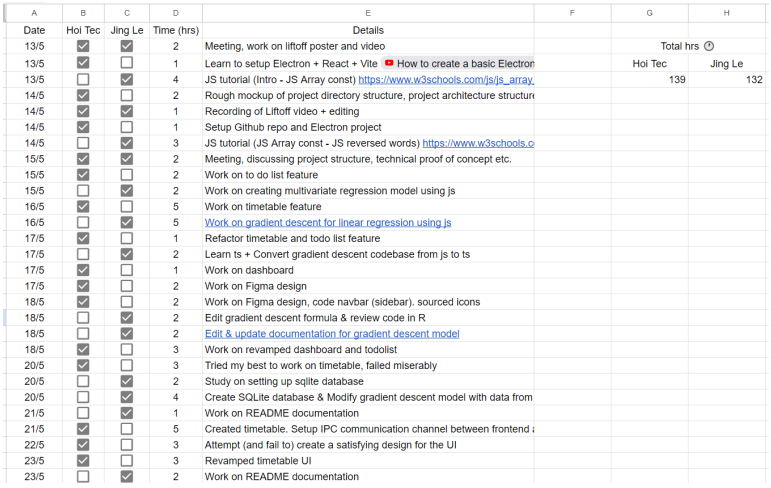




For more detailed features and bug reports, we use Github issues as our primary channel for communication. We chose to use Github's default templates for issues, which has worked well for us.



We use Figma to prototype the UI and design, allowing us to settle on a good looking design before moving on.



Date	Hoi Tec	Jing Le	Time (hrs)	Details		Total hrs	
13/5	☑	☑	2	Meeting, work on liftoff poster and video			
13/5	☑	☐	1	Learn to setup Electron + React + Vite	How to create a basic Electron	Hoi Tec	Jing Le
13/5	☐	☑	4	JS tutorial (Intro - JS Array const)	https://www.w3schools.com/js/js_array	139	132
14/5	☑	☐	2	Rough mockup of project directory structure, project architecture structur			
14/5	☑	☑	1	Recording of Liftoff video + editing			
14/5	☑	☐	1	Setup Github repo and Electron project			
14/5	☐	☑	3	JS tutorial (JS Array const - JS reversed words)	https://www.w3schools.c		
15/5	☑	☑	2	Meeting, discussing project structure, technical proof of concept etc.			
15/5	☑	☐	2	Work on to do list feature			
15/5	☐	☑	2	Work on creating multivariate regression model using js			
16/5	☑	☐	5	Work on timetable feature			
16/5	☐	☑	5	Work on gradient descent for linear regression using js			
17/5	☑	☐	1	Refactor timetable and todo list feature			
17/5	☐	☑	2	Learn ts + Convert gradient descent codebase from js to ts			
17/5	☑	☐	1	Work on dashboard			
17/5	☑	☐	2	Work on Figma design			
18/5	☑	☐	2	Work on Figma design, code navbar (sidebar), sourced icons			
18/5	☐	☑	2	Edit gradient descent formula & review code in R			
18/5	☐	☑	2	Edit & update documentation for gradient descent model			
18/5	☑	☐	3	Work on revamped dashboard and todolist			
20/5	☑	☐	3	Tried my best to work on timetable, failed miserably			
20/5	☐	☑	2	Study on setting up sqlite database			
20/5	☐	☑	4	Create SQLite database & Modify gradient descent model with data from			
21/5	☐	☑	1	Work on README documentation			
21/5	☑	☐	5	Created timetable. Setup IPC communication channel between frontend			
22/5	☑	☐	3	Attempt (and fail to) create a satisfying design for the UI			
23/5	☑	☐	3	Revamped timetable UI			
23/5	☐	☑	2	Work on README documentation			

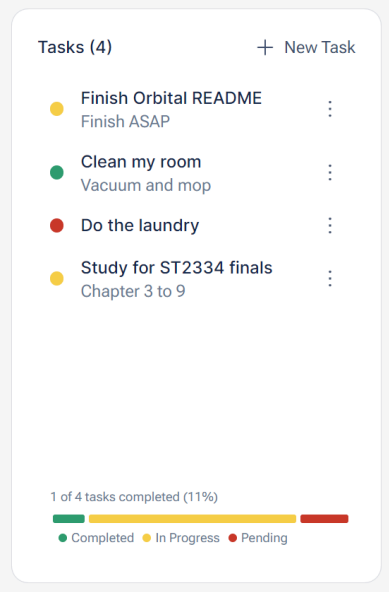
Finally, we have a Google Docs document to act as our “idea dump”, where we store all ideas and rough sketches. We also have a Google Sheets document to act as our Project Log and track our hours.

Features

Daily to-do List [Completed]

Description

The daily to-do list feature allows users to plan their day by creating, editing and deleting tasks. This to-do list resets on logout, removing completed and in-progress tasks, and keeps unstarted tasks.



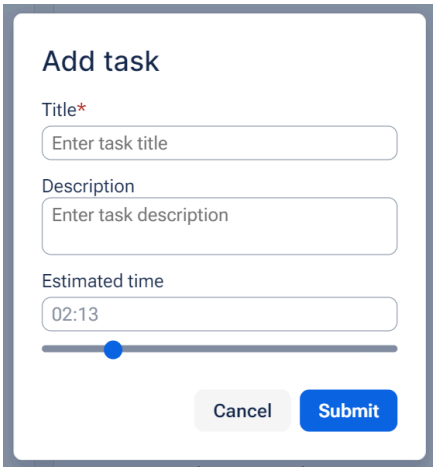
Within this list, users are able to keep track of the completion status of their tasks. Tasks can be one of three statuses: completed, in-progress and pending. A helpful tracker allows users to quickly visualize the total completion status of their tasks.

Implementation Philosophy

A user-friendly interface is crucial to ensure consistent use of the to-do list, which is critical for accurate data collection. As such, we have kept the to-do list minimal and intuitive, while not losing out on any core-features.

An example of this is our completion status tracking system: an intuitive “traffic light” system is used to allow users to track their current progress on a task, whereby the colors red, yellow and green are used to represent the “pending” (not started), “in-progress” and “done” state respectively. This status tracking system not only helps the user keep track of their tasks, but also serves as a means for the system to collect data, which will be used in training the linear regression model and displayed as statistics.

In order to update the completion status of any task, users may simply left-click anywhere on the task to “cycle” to the next status. Right clicking on the task will instead revert to the previous status. Users are able to cycle in order of: pending (default) -> in-progress -> completed.



Moreover, we need to collect a user estimated time taken to complete any task when it is created. This is required to make use of the timetable optimization feature (which we will go into detail in a future section). To make this process seamless and accessible, users may choose to type the time directly, or even drag and scroll to pick a time. Of course, relevant input validation methods are put in place.

We are using Redux stores to manage the state of the to-do list, and any update to the state of the list will also be reflected in the backend, which will eventually be stored in the database. The state of the to-do list is also automatically requested from the backend during startup of the app.

Implementation Challenges

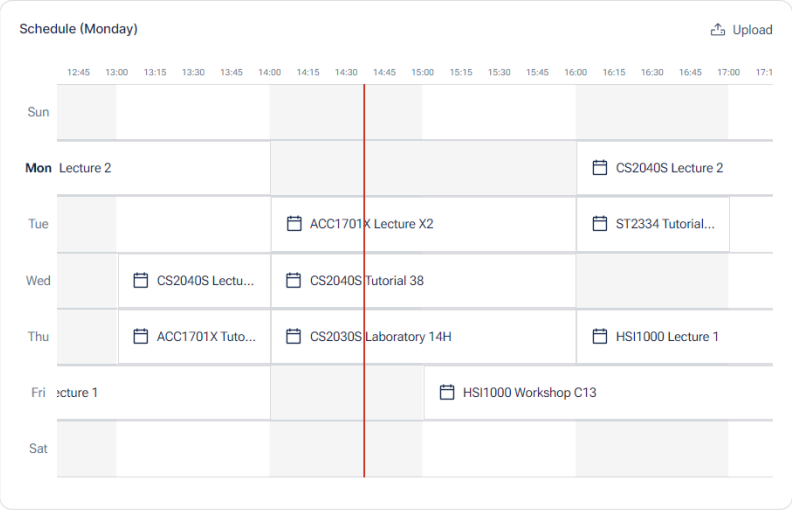
One great challenge when creating the to-do list is the user experience of using the app. Originally, we considered making the users input the task title, description, estimated time and priority (importance) during task creation. However, we feel that doing so would make the process of creating a new task cumbersome and unintuitive.

As such, we decided to remove the priority selection in favor of a simpler task creation and management process. But we had to change the timetable optimization algorithm as a result.

Timetable with NUS Mods Integration and Optimization [Completed]

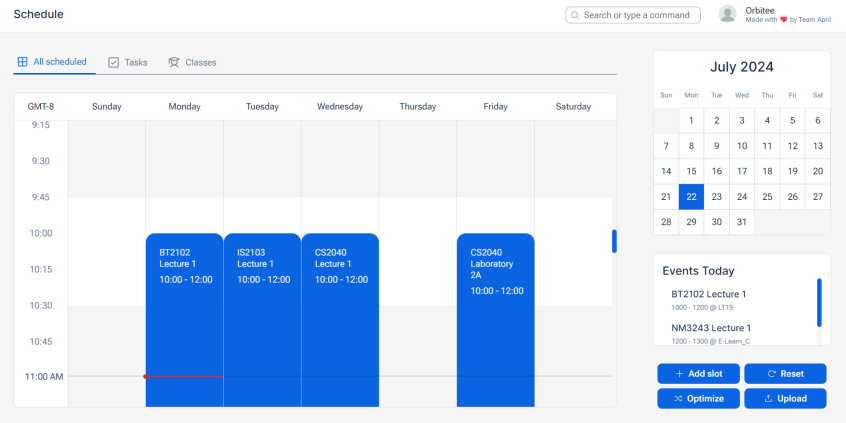
Description

The timetable system is designed to help users manage and visualize their weekly routines by integrating seamlessly with NUS Mods. This feature allows users to import their semester timetable directly from NUS Mods using a URL, ensuring that their academic schedule is accurately reflected within the application.



Once imported, each class is displayed within the timetable, occupying the appropriate time slots. These slots are then reserved, preventing to-do items from being scheduled during class times. This ensures that users have a clear view of their available time for completing tasks and prevents conflicts between academic commitments and other activities.

In addition, the system attempts to fit all to-do list items into the user's schedule. This is achieved through an algorithm that optimizes tasks in the to-do list based on the user-estimated times given.



Users can interact with the timetable through a dashboard widget and a dedicated schedule page. The dashboard widget provides a quick overview, while the schedule page offers more detailed functionality, including the ability to optimize the timetable, reupload the timetable, create, edit and delete pre-existing timetable slots.

Implementation Philosophy

A seamless integration with NUS Mods is vital to provide users with an accurate and up-to-date timetable. This system aims to reduce the cognitive load on users by automating the process of importing and organizing their schedules, allowing them to focus more on completing tasks.

We use the Best Fit Decreasing (Bin-Packing) algorithm to optimize the timetable. This algorithm prioritizes tasks with longer estimated times, fitting them into the timetable before shorter tasks. This method ensures efficient use of available time slots, maximizing productivity.

The integration with NUS Mods involves extracting data from the provided URL, making API calls to retrieve class details (refer to Implementation Challenges section below), and storing this information in the database. The frontend updates dynamically through Redux, reflecting any changes made to the timetable.

The schedule page also has some supplementary features, such as a filter which allows users to filter between tasks (slots added through the “optimize” feature) and classes (slots imported via NUS Mods). Moreover, an “events today” tab lets users view upcoming items in their timetable.

Implementation Challenges

When choosing to share a NUS Mods timetable using the “Share/Sync” button via URL, we would notice that the URL is formatted in a specific way. Consider the example of the timetable below:

```
https://nusmods.com/timetable/sem-2/share?CS1010E=SEC:2,TUT:18&CS1010R=&CS1010S=TUT:12
                    ,REC:03,LEC:1&CS2030S=LAB:16E,REC:03,LEC:2
```

Since exact class start times, end times and days are not given in the NUS Mods URL, we have to extract the relevant data from this URL, and make specific requests to the NUS Mods API in order to get all the data we need.

Evidently, the relevant data we extract from the URL would include: current semester, module codes of classes taken and lectures/tutorials/recitation/laboratory codes of classes taken. This is done using some string manipulation, which is the job of this function:

```
// Extracts relevant information used to make API request
function extractURL(url) {
  // e.g. [ 'CS1010=TUT:06,SEC:1', 'CS1010E=SEC:2,TUT:18' ]
  const allEnrolledLessons = url.substring(url.indexOf('?') + 1).split('&');
  const enrolledLessons = allEnrolledLessons.filter((lesson) => lesson.substring(0, 6) != 'hidden');

  // e.g. 2021-2022
  const date = new Date();
  const academicYear =
    date.getMonth() < 8
      ? `${date.getFullYear() - 1}-${date.getFullYear()}`
      : `${date.getFullYear()}-${date.getFullYear() + 1}`;

  // Semester 1 | 2
  const semester = +url.split('/')[4][4];

  return { enrolledLessons, academicYear, semester };
}
```

Once we have the relevant data, we can finally retrieve and process our specific mod class timings. This is done using the Axios package. Which gets us a JSON object with the relevant data, of which we filter through and transform into our desired format.

Besides that, we have encountered issues with the reliability of NUS Mods in general. First, we cannot be sure when NUS Mods will receive new semester data, as they themselves rely on NUS for their API's data source. This might cause issues around the months of July and August, where we cannot be sure if the data received is outdated.

Moreover, we have encountered issues where the data shown in the NUS Mods website and data return from their API differ. However, this is rather uncommon, and can be remedied by the user manually editing the time of the affected timetable slot.

Since the NUS Mods API is the only reliable source of data regarding NUS module schedules, we can only make do with what we have.

Finally, another issue we have encountered is regarding timetable optimization. Originally, we had planned to optimize the timetable taking into consideration the **predicted** productivity at any point in time. However, due to reasons which we will go through below, we have decided to optimize based purely on the users' estimated timing.

To optimize timetables with predicted productivity, we need to have the necessary variables, which include time of day, day of the week, hours in classes, and hours focused. We will need to divide the estimated time by productivity to obtain the potential actual time. However, all of the variables affecting productivity differ at different times.

For methods like the Best Fit Decreasing model and the knapsack problem, we first need to sort the tasks based on their potential actual time. Unfortunately, this is extremely tedious, as we need to attempt to fit the tasks into all slots, calculate the productivity to obtain the potential

actual time in each slot, store the schedule in each iteration of the loop, and only then find the best solution.

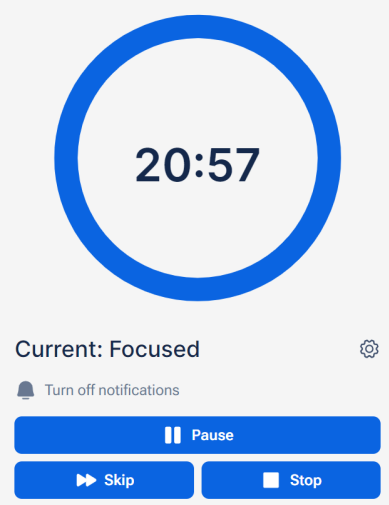
At the same time, we also considered that when inputting the estimated time, users themselves may have already taken productivity into consideration. For example, if they are in the mood to do a particular task or if they are feeling productive, they might input a shorter estimated time.

Therefore, we have decided to optimize the timetable using only the users' inputted estimated time, using the Best Fit Decreasing model.

Adaptive Pomodoro Timer [Completed]

Description

The Pomodoro technique traditionally involves working for a set period, followed by a short break, and after several cycles, taking a longer break. Our adaptive Pomodoro timer enhances this technique by using a linear regression model to adjust these durations in real-time according to the user's current productivity score.



The adaptive Pomodoro timer is designed to help users manage their work and break periods efficiently by dynamically adjusting session lengths based on their productivity, such that:

- Work session duration is increased, and break durations are decreased during high productivity sessions
- Work session duration is decreased, and break durations are increased during low productivity sessions

Users can manually set their preferred work session length, short break length, and long break length.

Also, the timer sends system-level notifications to alert users when a session ends, ensuring they take necessary breaks or resume work without constantly monitoring the timer. Users can opt to disable or enable the pomodoro notification in the settings page too.

Implementation Philosophy

A seamless and adaptive user experience is at the core of our implementation. We aim to provide an intuitive interface that allows users to focus on their tasks while the timer adjusts to their productivity levels. This ensures users can maintain high productivity without manual adjustments, making the process more efficient and less disruptive.

For example, during highly productive periods, the system may increase the work session length and decrease break durations to maximize focus time. Conversely, during less productive periods, the system may shorten work sessions and extend breaks to help users recharge. This adaptiveness is crucial for maintaining a balanced workflow and preventing burnout.

To do this, we have determined a baseline focus duration of 25 minutes, short break duration of 5 minutes and long break duration of 20 minutes. Then, the productivity score is calculated by our model. This score (in percentage) is divided against short and long break duration, and multiplied against focus duration.

```
export async function getPomodoroSettings() {
  const DEFAULT_FOCUS_TIME = 1500;
  const DEFAULT_SHORT_BREAK = 300;
  const DEFAULT_LONG_BREAK = 1200;
  // const PRODUCTIVITY_WEIGHT = 0.5;

  const productivity = await runModel();

  return {
    shortBreakDuration: roundToMinute((DEFAULT_SHORT_BREAK * 100) / productivity) || DEFAULT_SHORT_BREAK,
    longBreakDuration: roundToMinute((DEFAULT_LONG_BREAK * 100) / productivity) || DEFAULT_LONG_BREAK,
    workDuration: roundToMinute((DEFAULT_FOCUS_TIME * productivity) / 100) || DEFAULT_FOCUS_TIME,
  };
}
```

The effect of this is that when a users' productivity is up, the length of their breaks are shortened, and focus duration is lengthened. This will maximize their focus "uptime", allowing them to squeeze the last bit of productivity from their session.

Descriptive Statistics [Completed]

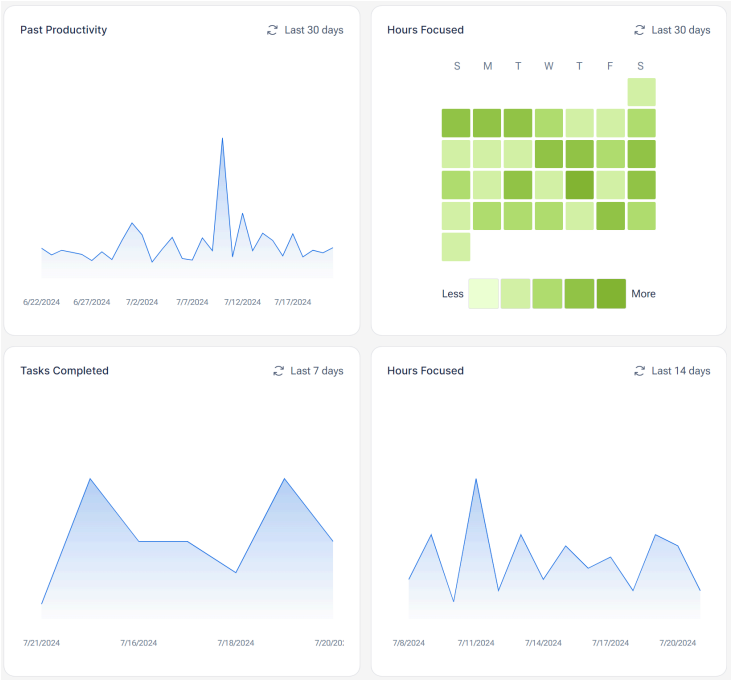
Description

The descriptive statistics feature allows users to visually track their productivity through various metrics, which include:

- Hours focused based on tasks and Pomodoro sessions
- Average productivity per day
- Number of tasks completed per day

Statistics are displayed on a dedicated statistics page for long-term, general statistics, and on the dashboard for short-term, specific statistics.

These statistics are designed to help users identify patterns in their productivity, such as peak focus times and days when they are most productive. By analyzing this data, users can make informed decisions to optimize their schedules and improve their efficiency.



The statistics page provides a long-term view of productivity data, allowing users to see trends and patterns over extended periods up to 30 days before. This page includes various graphs and charts for better visualization, such as heatmaps for hours focused, line charts for productivity trends, and bar charts for tasks completed. Users may toggle between seeing 7, 14, 30 days data.

Statistics shown on the dashboard offers a short-term, daily overview of specific statistics. This includes widgets that show daily hours focused, tasks completed, and average productivity. These widgets update in real-time as users complete tasks, providing immediate feedback on their progress. Bars beside the values show the statistics from the previous 7 days, with the highlighted bar representing the current (today) value.

The integration of these statistics into both the statistics page and the dashboard ensures that users have access to both detailed analyses and quick overviews, enhancing their ability to monitor and improve their productivity.

Implementation Philosophy

Providing users with a clear and insightful view of their productivity statistics is essential for fostering continuous improvement. The design focuses on making the data easy to understand and accessible from different parts of the app. The home page offers a quick overview of daily productivity, while the detailed statistics page provides in-depth analysis.

We implemented a user-friendly approach to generate test data for demonstration purposes. An 'Experimental' section under the settings page allows testers to populate the database with 30 days of randomly generated test data by clicking the 'Generate' button. This feature helps testers understand how their statistics will appear and function.

Implementation Challenges

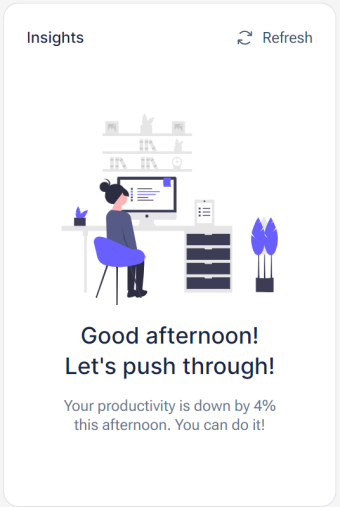
One of the main challenges was ensuring the statistics are both meaningful and easy to interpret. We needed to balance detailed data presentation with a clean and intuitive user interface. Additionally, updating statistics in real-time as users complete tasks requires efficient state management and data processing.

The actual implementation involves automatically converting daily data into statistics upon user logout. This ensures that users see updated statistics the next day without manual intervention.

Insights [Completed]

Description

The customized messages and recommendations feature aims to provide users with useful insights and motivation based on their predicted productivity data. By analyzing user activity and productivity patterns, this feature generates personalized messages that help users make informed decisions about their work habits and schedules. These messages are displayed on the dashboard and tailored to the user's current productivity status and patterns, providing real-time feedback and encouragement.



The messages are dynamic and change based on the user's current productivity data. This ensures that the feedback is always relevant and timely, helping users to adjust their activities to maintain or improve productivity.

Example messages include:

- Burning the midnight oil? Hang in there! Your productivity is down by 10% during these hours. Take a break if needed. (Between 12am - 6am)
- Good evening! Great job today! Your productivity is up by 10% this evening. Time to wind down and celebrate your success. (Between 6pm - 12am)

Implementation Philosophy

The primary goal is to provide meaningful and motivational insights that help users optimize their productivity. By leveraging historical data and a predictive model, we can offer personalized recommendations that are both timely and relevant. This approach not only enhances user engagement but also encourages consistent use of the app.

To achieve this, we developed a regression model using the gradient descent algorithm. This iterative optimization algorithm helps find the minimum of the loss function, allowing us to fine-tune the model's parameters, including the intercept and weights of the variables. This

ensures accurate productivity predictions. We will then use the best parameters of a model, including the intercept and the weights of the variables to predict productivity.

To predict a user's productivity at any time of the day, we collect data on various factors that may affect productivity, including:

- Time of day
- Day of the week
- Hours in classes
- Hours focused

To verify the correctness of the model, we've generated 100 random data points to compare the regression model generated using our model and the OLS model in R. We also compared the result of predicting multiple new data points using both models. (This can be seen in the Implementation Challenges section below).

Implementation Challenges

One of the challenges we faced was ensuring the accuracy and reliability of the regression model. During testing, we generated 100 random data points to compare our model with the OLS model in R. Although both models returned the same predicted values, the intercept and weights differed due to differences in handling categorical data.

```
Weights: [
   -8.126635642004786,   -6.06528408387496,
    16.70737615121238,   4.093556296368344,
   2.5600415469894773, -0.9220471383527968,
    11.21257244288304,  13.182710032895367,
   12.681845751313622,  6.2777278735492645,
   -5.467719334401787,  0.0856331013459517,
   17.149993482366156,  -26.09212530486928,
   28.006934036818745,  0.4257165021125505,
  -1.1961431415144903, -1.3756909885515578,
   11.250212778612058,  10.928362963477078,
   11.839404852573587
]
Intercept: 32.642289606129026
Prediction for new example1: 45.41429591204818
Prediction for new example2: 50.47606680399542
Prediction for new example3: 45.771572807544395
```

```
Coefficients:
                        Estimate Std. Error t value Pr(>|t|)
(Intercept)              35.8218    14.4132   2.485  0.01500 *
time_of_dayearly afternoon   2.0614    12.5663   0.164  0.87011
time_of_dayearly morning    24.8340    13.8395   1.794  0.07647 .
time_of_dayevening          12.2202    11.9910   1.019  0.31118
time_of_daylate afternoon   10.6867    11.0545   0.967  0.33656
time_of_daylate morning      7.2046    12.3819   0.582  0.56227
time_of_daymidnight         19.3392    12.6923   1.524  0.13148
time_of_daynight            21.3093    11.9653   1.781  0.07867 .
day_of_weekTuesday          -6.4041    11.5307  -0.555  0.58015
day_of_weekWednesday       -18.1496    10.8021  -1.680  0.09677 .
day_of_weekThursday        -12.5962    11.2539  -1.119  0.26633
day_of_weekFriday            4.4681    10.7952   0.414  0.68004
day_of_weekSaturday        -38.7740    12.4908  -3.104  0.00263 **
day_of_weekSunday           15.3251     9.6009   1.596  0.11433
hours_in_classes             0.4257     0.7406   0.575  0.56702
hours_focused               -1.1961     0.5974  -2.002  0.04861 *
weatherrainy                12.6259     8.4242   1.499  0.13782
weatherstormy               12.3041     8.1445   1.511  0.13475
weathercloudy               13.2151     8.1983   1.612  0.11087
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 27.55 on 81 degrees of freedom
Multiple R-squared:  0.3282,    Adjusted R-squared:  0.179
F-statistic: 2.199 on 18 and 81 DF,  p-value: 0.008806

       1        2        3
45.41430 50.47607 45.77157
```

In R, categorical data has reference levels (e.g., "dawn," "Monday," "sunny"), whereas our gradient descent model does not use reference levels. By aligning our model with these reference levels, we achieved consistent intercepts and weights with the R model. However, we chose to implement our model without reference levels to avoid excluding any vectors. This approach did not affect the correctness of our predictions.

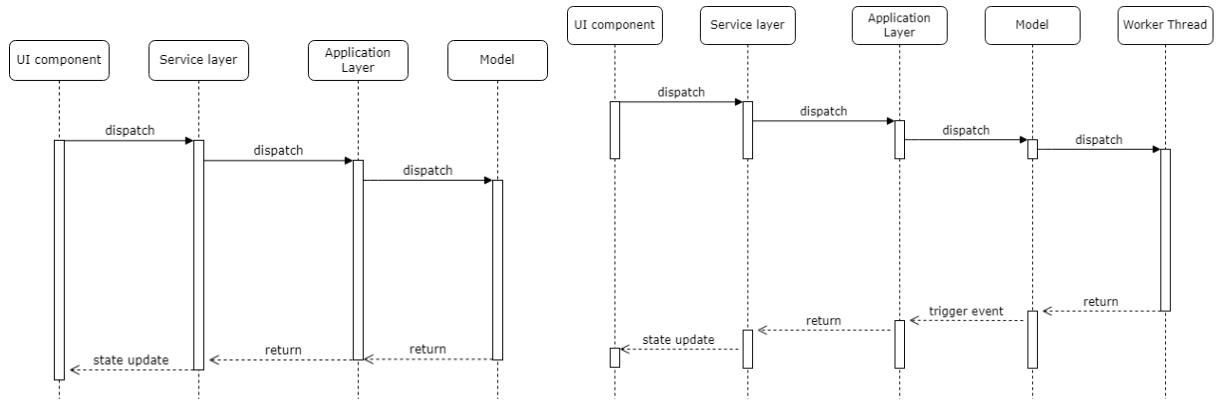
```
Weights: [
   2.061041918128529,  24.833687233168465,
  12.219927105109592,  10.686397785266431,
   7.204264859922245,   19.33889637390574,
   21.30904983849606,  -6.404260913347256,
  -18.14958717564117, -12.596278662228514,
    4.46808101690683, -38.774002938587245,
  15.325068837939595, 0.42571467273612473,
  -1.196147855460179,  12.625849167767848,
  12.304040310228192,  13.215058930583378
]
Intercept: 35.822193863054714
Prediction for new example1: 45.41431101505826
Prediction for new example2: 50.47638913734286
Prediction for new example3: 45.77162085574166
```

The intercept and weights now matches with the model in R

Moreover, it has been found that this model requires heavy computation, running it causes the app to freeze for a second before being able to interact with anything. As such, we had planned to change the implementation of the model to be run synchronously by creating Node JS worker threads.

However, one issue with using worker threads is that instead of returning the value of computation directly, we have to create a new set of events which will update after the computation is returned.

Below is the original vs proposed system visualized:



Original (left) vs proposed (right) system

This method has been implemented and tested to work well, however we have encountered issues which made us revert back to the original system. Namely, web worker support is poor in many Node packages, this includes the testing library we were using - Jest. Using web workers would cause our tests to break, and resources (documentation/bug reports) on the issue were next to nothing.

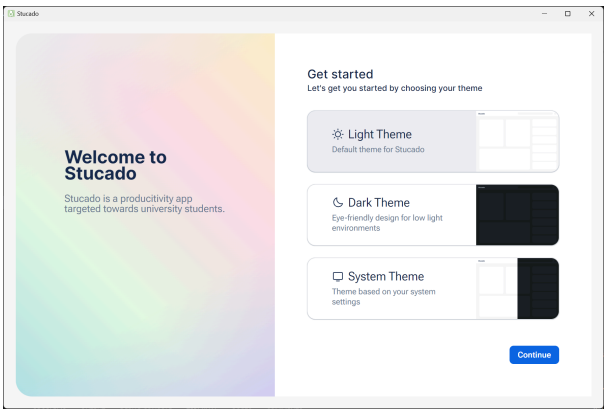
As such, our team has made a collective decision to maintain the testability of our app (since the gradient descent model is an extremely important part of the project, and we cannot afford to have any errors regarding this), in exchange for slightly decreased performance when running the model.

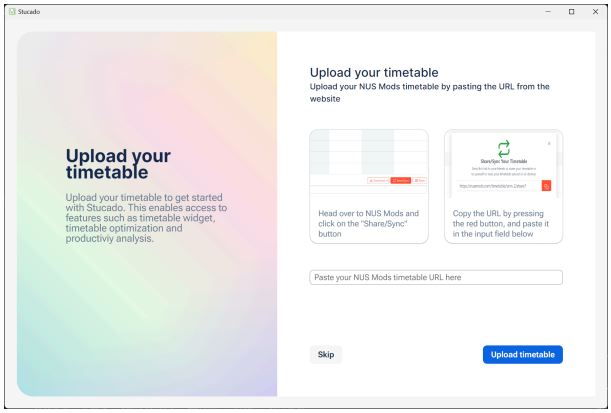
One compromise to minimize this negative impact is to decrease the number of iterations the model will run, this will of course decrease the overall model accuracy in exchange for performance. However, we have fine tuned the model to lose only ~1% in accuracy, which we feel is a good middle ground between accuracy and performance.

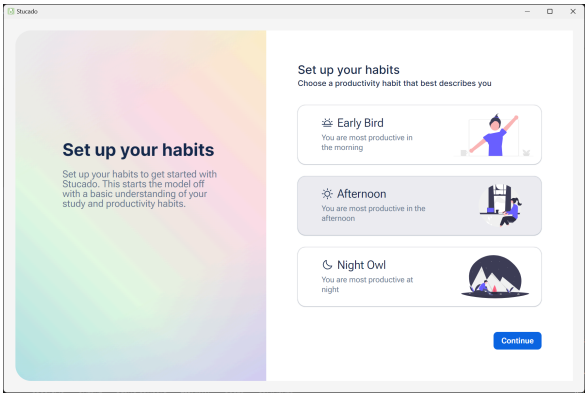
Onboarding Process [Completed]

Description

The onboarding process for new users is designed to help them set up and personalize their environment in the app. This process ensures a smooth and intuitive introduction to the app's features and functionalities. The onboarding will be displayed only during the first startup of the app and will consist of multiple steps to guide users through initial configuration and customization.







Implementation Philosophy

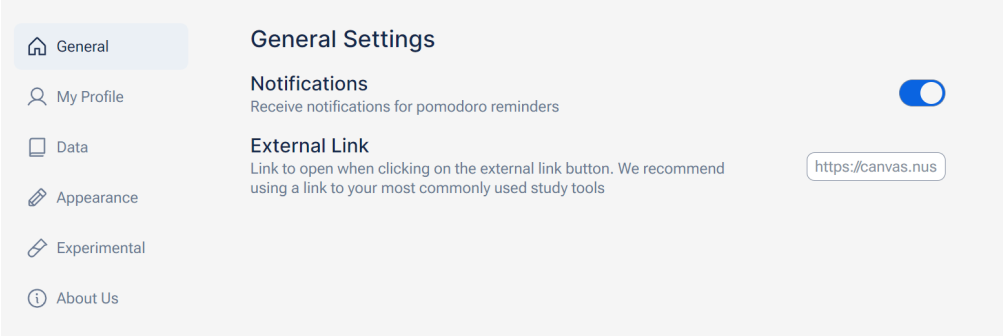
Key steps in the onboarding process include:

- **Welcome Page**: Users are welcomed to the app and given the option to choose their preferred display mode. They can select between dark mode, light mode, or using the system settings to match their device's theme. This step ensures that users can start with a comfortable visual experience.
- **Timetable Upload Page**: Users are given the opportunity to upload their NUS Mods timetable. This is crucial for enabling several of the app's features, including timetable optimization, productivity analysis, and the timetable widget. By integrating their academic schedule, users can make the most of the app's capabilities from the start.

- **Preliminary Data Collection Step**: Users select their study habits from three options: “early bird,” “afternoon,” or “night owl.” This selection populates the linear regression model with default data points, allowing the user to immediately benefit from personalized productivity insights and recommendations.

Settings and Personalization Features [Completed]

Description



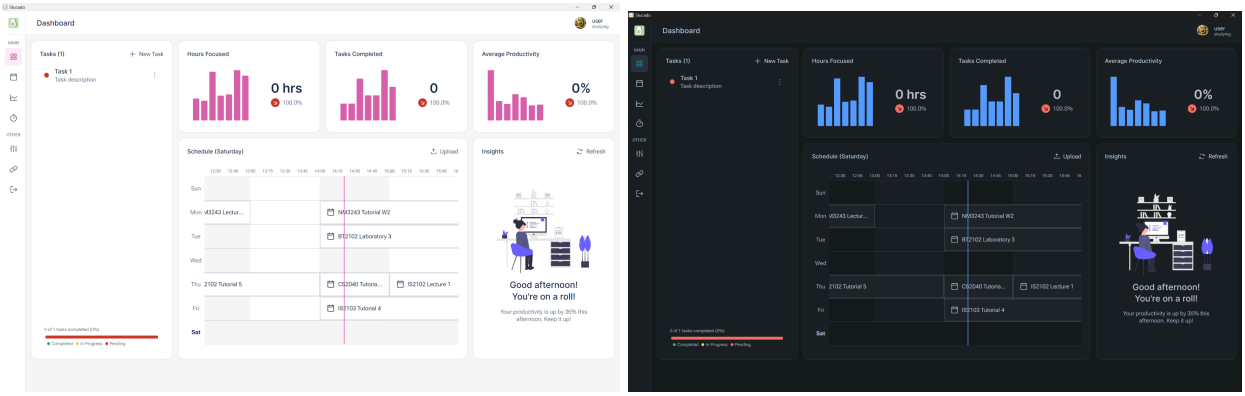
Settings and personalization features are designed to provide users with a high degree of control and customization over their app experience. These settings allow users to tailor the app's functionality and appearance to their preferences, enhancing usability and personal satisfaction. The settings are categorized into several sections, each focusing on different aspects of the user experience:

- **General**: Users can toggle notifications on or off, ensuring they receive alerts according to their preferences. Additionally, there is an "External Link" feature, which provides quick access to a specific website. By default, this is linked to Canvas (to tailor to NUS students), but users can customize this link to any website they choose, making it a versatile tool for quick access to frequently used resources.
- **My Profile**: Users can personalize their profile by changing their username, status, and selecting from one of three profile pictures. These profile settings are purely decorational and are only visible to the user, adding a personal touch to their app experience.
- **Data & Security**: This section includes a button to clear all data. This feature is crucial for users who wish to reset their app data, ensuring that they can manage their information securely and start fresh if needed.
- **Appearance**: Users can customize the visual theme of the app, choosing between light mode, dark mode, or system settings. Additionally, they can select from different color schemes (blue, green, purple, magenta) to further personalize the app's appearance, creating a visually pleasing and comfortable user experience.
- **Experimental**: This section includes tools for testing purposes, such as generating test data, resetting onboarding, and tearing down the app environment. These features are intended for users who are involved in testing and development, providing them with the necessary tools to experiment with different scenarios and settings.

Implementation Philosophy

The implementation of the settings and personalization features is guided by the philosophy of user empowerment and flexibility. The goal is to create an environment where users feel in

control of their app experience, allowing them to customize both functionality and aesthetics to suit their individual needs and preferences.



Different app themes

By offering a range of customizable options, from practical settings like notifications and external links to more personal touches like profile customization and color schemes, the app ensures that users can create an experience that feels uniquely their own. This level of personalization not only enhances user satisfaction but also promotes consistent and prolonged use of the app.

Software Engineering Principles

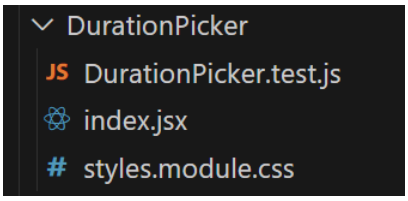
Modularity and DRY

In our project, we aim to make our code as modular as possible. This can be seen in an example in our UI elements, in which we separate our React components into individual “units” which can be reused across the code. For example, this DurationPicker component:


```
 */
const DurationPicker = ({ label, min = 0, max = 1440, name, value = min, onChange }) => {
  const convertToDuration = (minutes) => {
    const hours = String(Math.floor(minutes / 60));
    const remainingMinutes = String(minutes % 60);
    return `${hours.padStart(2, '0')}:${remainingMinutes.padStart(2, '0')}`;
  };

  const convertToMinutes = (duration) => {
    const [hours, minutes] = duration.split(':');
    return hours * 60 + +minutes;
  };

  const validateDuration = (duration) => {
    const regex = /^(0[0-9]|1[0-9]|2[0-3]):[0-5][0-9]$/;
    return regex.test(duration) && convertToMinutes(duration) >= min && convertToMinutes(duration) <= max;
  };

  const formatDuration = (value) => {
    const [hours, minutes] = value.split(':').map((part) => part.padStart(2, '0'));
    return `${hours}:${minutes}`;
  };
```

This component’s only function is to provide an interface for users to select a time, either through typing input or dragging the slider. As such, it is easy to reuse this component anywhere in the code without having to write repetitive code and styles.

Moreover, the styles of each component is placed in the same folder as said component, alongside with its unit test:



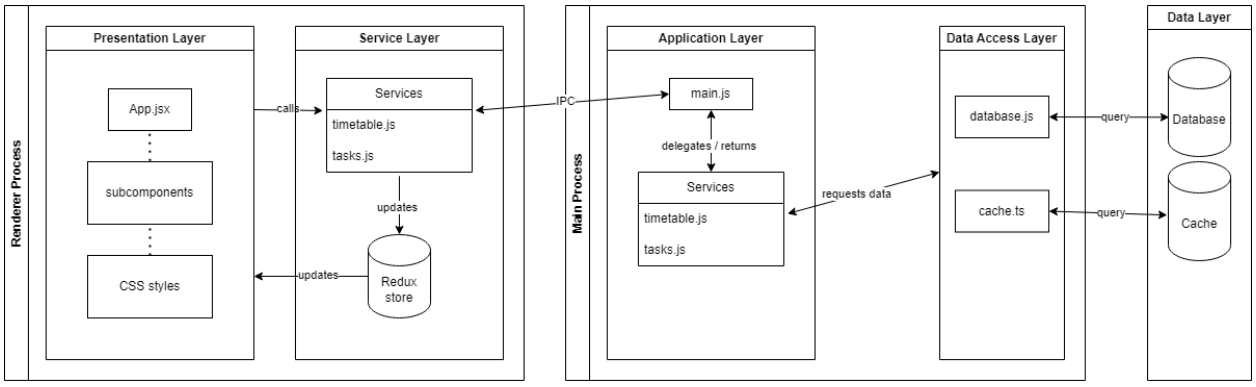
This allows us to make changes to both the functionality, and styling of a component without having to navigate through a complex file structure.

Separation of Concerns (SoC)

We have developed a system which allows the separation of concerns between each module in our codebase. This system utilizes Electron's framework to structure our application into distinct layers: Presentation, Service, Application, Data Access, and Data Layer:

- **The Presentation Layer** is responsible for the user interface and user experience. This layer includes all the components and styles that users interact with, such as the dashboard, timetable, and settings pages. By isolating the presentation logic, we ensure that changes to the user interface do not affect the underlying business logic or data handling processes.
- **The Service Layer** acts as an intermediary between the Presentation Layer and the Application Layer. It contains simple business logic, and is in charge of managing the state of the app and making API calls to the backend. Services are designed to be reusable and modular, handling specific areas such as tasks, timetable, settings etc.
- **The Application Layer** coordinates the overall workflow of the system and contains the bulk of the business logic. It manages the overall lifecycle of the app (creation of windows etc.), handles the machine learning models, interacts with native system functions (such as sending system notifications), making external API calls (to NUS Mods API) etc.
- **The Data Access Layer** is responsible for communication with local (SQLite) databases and manages the actual storage and retrieval of data. It handles the structure and organization of data, ensuring that it is stored efficiently and can be accessed quickly.
- **The Data Layer** consists of the actual database, where all application data is stored. This includes user information, task details, timetable schedules, settings, and any other relevant data.

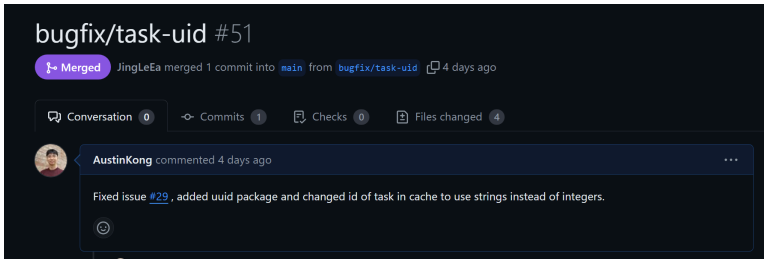
This system can be seen below, providing a clear representation of how each layer interacts and contributes to the overall architecture. By structuring our application in this layered manner, we achieve a high level of modularity and separation of concerns, leading to a maintainable, scalable, and robust system.



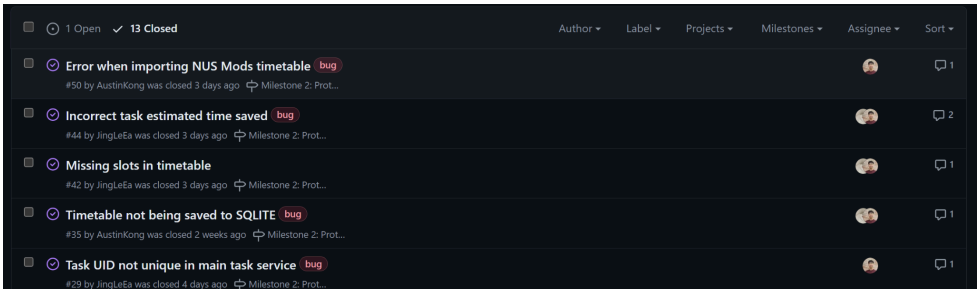
(Slice of) 5 layer system architecture

Version Control

We utilize Git for version control to manage the development of our application. Our Git workflow follows a structured process to ensure smooth collaboration and maintain high code quality.



We use a branching strategy to manage the code. The main branch contains stable code, and no changes are made to the branch directly. Instead, we create feature, bugfix or refactor branches to work on the code. Once a branch is complete, a pull request is created and a review request is sent to the other team member. We also made sure to stick to a consistent naming convention for branches, e.g. using the prefix “feature/”, “bugfix/” and “refactor/” to keep the repository organized.



Moreover, we use Github issues extensively to track bugs and future features that do not require our immediate attention. Once a bug is solved or a feature is implemented, we will reference this in the description of a corresponding pull request, and then close the issue. Tags were also used to identify the type of issue it is.

Code Quality and Standards

Maintaining high code quality and adhering to coding standards are crucial aspects of our development process. To ensure our codebase remains clean, readable, and consistent, we utilize tools such as ESLint and Prettier. Moreover, we aim to comment any ambiguous pieces of code to facilitate understanding and maintainability.

We also aim to adhere to best practices such as creating frequent commits with descriptive messages following a style guideline. Each commit focuses on a single logical change, making it easier to understand the purpose of each update and to identify potential issues.

Code Testing

Our testing approach for Stucado is designed to ensure that the application is reliable, functional, and user-friendly. Our comprehensive testing strategy includes multiple steps.

Unit Testing

Unit testing is fundamental to our development process. We use Jest to verify that individual components and functions work as intended. For instance, we have developed unit tests for key features like the Gradient Descent Productivity Model.

Integration Testing

Integration testing ensures that various modules and services within our application interact correctly. Using Jest, we verify the seamless operation of integrated units.

System Testing

System testing involves evaluating the system as a whole, simulating real-world interactions. We use Playwright to perform tests covering complete user workflows. Planned system tests include user onboarding, task management, timetable and scheduling, Pomodoro timer functionality, statistics and insights accuracy, and settings adjustments.

User Testing

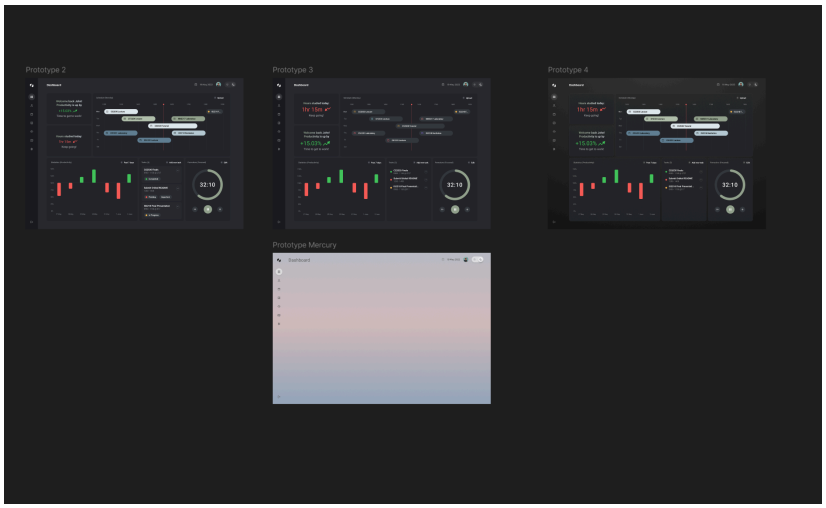
User testing gathers feedback from real users to identify usability issues and gather improvement suggestions. Our plan includes peer evaluations and direct user feedback via surveys and forms, focusing on Peter Morville's 7 Factors of User Experience: Useful, Usable, Findable, Credible, Accessible, Desirable, and Valuable.

For detailed information, please refer to the Testing section below.

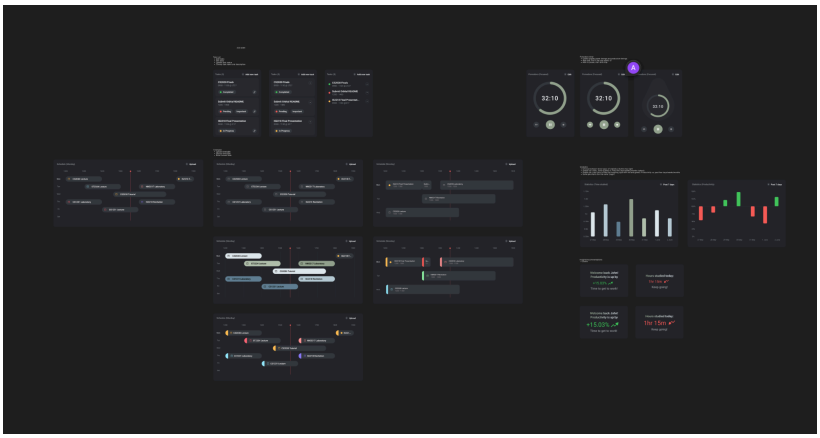
UI/UX Design

User interface (UI) and user experience (UX) design are critical components of Stucado. We aim to create an intuitive, aesthetically pleasing, and user-friendly interface that enhances the overall experience for our users. Our design process involves several key steps.

Wireframing and Prototyping



Prototyping overall app feel



Prototyping widget designs

Using Figma, we create wireframes and prototypes to visualize the layout and flow of the application. This helps us iterate on the design quickly and gather feedback before moving on to development.

Consistency and Branding

We ensure consistency in our design by adhering to a cohesive color scheme, typography, and branding elements. This creates a unified and professional look, enhancing the overall user experience.

This is done by adhering to a design system, for this project we chose to adhere to the Atlassian design system. However, if the app were to be developed further in the future, we would develop our own design system.

Responsive Design

We design our application to be responsive, ensuring that it works well on various devices and screen sizes. Since Stucado is aimed to be a desktop only app, responsive work is minimized. However, we have still designed the app to feel the same across the three most common screen sizes used as of today (Source):

- 1920x1080
- 1366x768
- 1536x864

Moreover, we have also made the app accessible for smaller (vertical) resolutions. Especially useful when users wish to multitask and use Stucado next to other apps in a split screen manner. This is all done using CSS media queries.

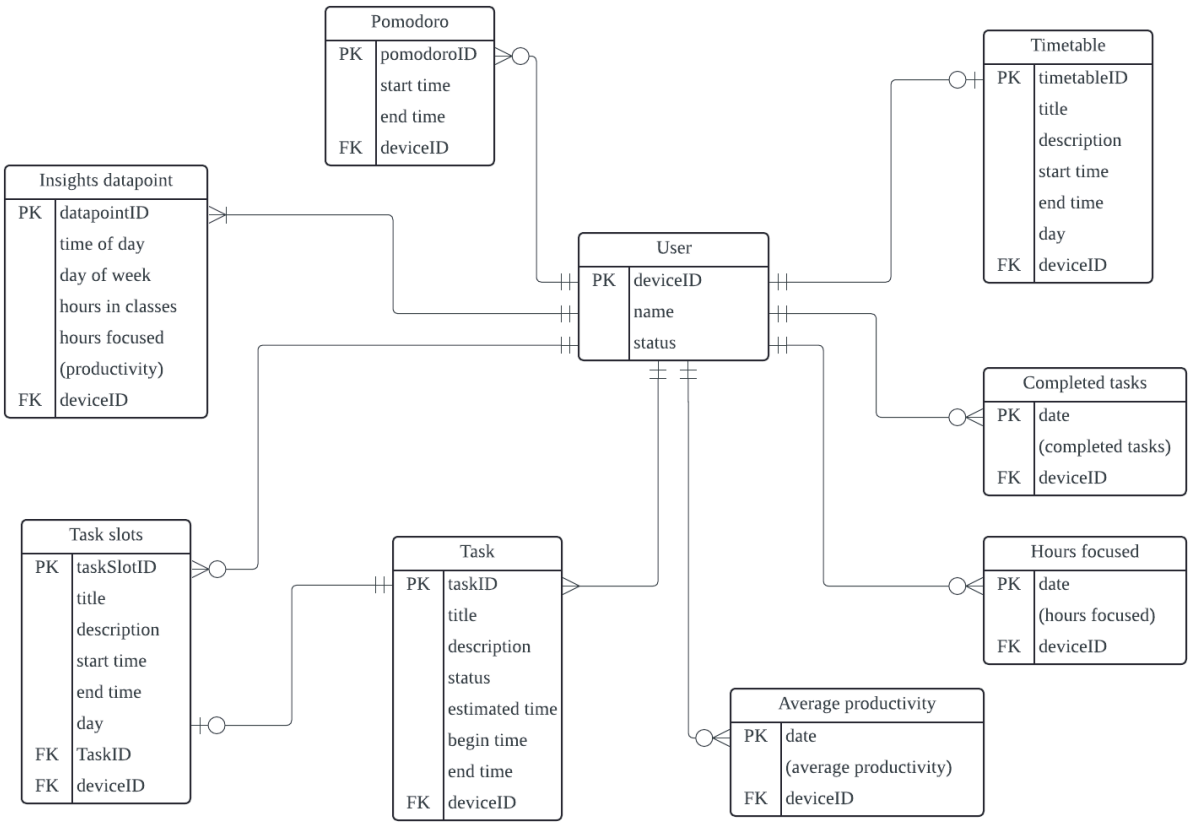
Accessibility

We prioritize accessibility by following best practices and guidelines, such as the Web Content Accessibility Guidelines (WCAG).

We have carefully selected colors with high contrast ratios, ensuring that text and important UI elements stand out clearly against their backgrounds. For example, we use dark text on light backgrounds and vice versa, adhering to WCAG guidelines for contrast ratios.

Furthermore, we have also implemented dark and light mode support to improve accessibility and user comfort, especially in low light environments (for dark mode users). Dark mode provides some benefits, which includes reduced eye strain and energy efficiency.

Entity Relationship Diagram (ERD)

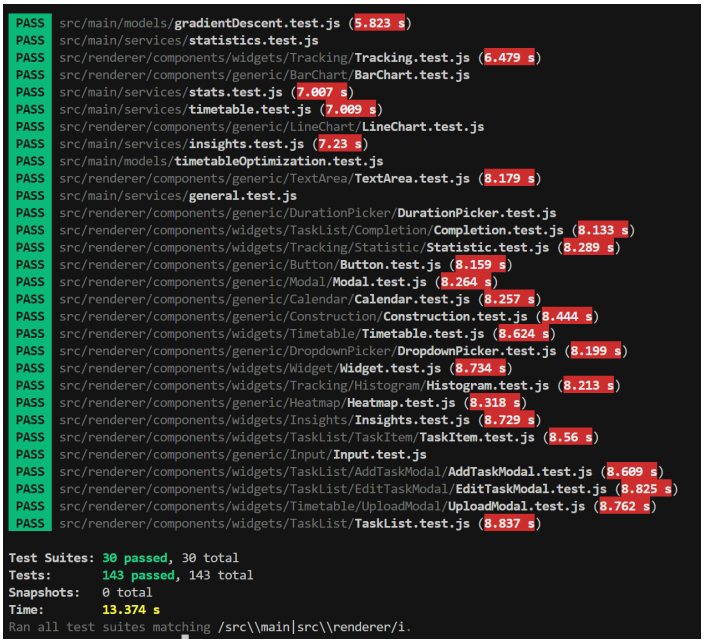


Testing

Unit Testing

Overview

Unit testing is fundamental to our development process, ensuring that individual components and functions work as intended. We use Jest to run our unit tests, a versatile JavaScript testing framework that integrates seamlessly with our tech stack.



All unit (and integration) tests passing

Unit testing involves isolating and testing individual units of code to verify their correctness. These units can be functions, methods, or React components that represent the smallest testable parts of our application. By writing unit tests, we can detect bugs early in the development cycle, facilitating a robust and maintainable codebase.

Case Study

All unit tests passing is a testament to our rigorous testing process. For each generic component in our React frontend, we have written unit tests to verify its functionality. Additionally, we have developed unit tests for functions and business logic across our codebase.

To illustrate, we will demonstrate the unit tests we have written for one of our key features - the Gradient Descent Productivity Model:

To test the model, we first randomly generate three groups of raw data. This data is representative of different productivity scenarios that our model needs to handle.The generated

data is saved into a CSV file to be executed using R, where the model is replicated and run to produce the expected results.

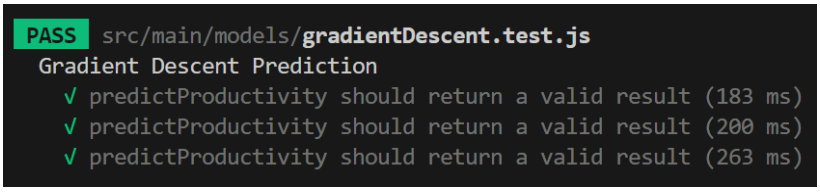
Below is the sample execution of predicting the productivity of a datapoint in R:

```r
rawdata1 <- read.csv("raw data.csv")

rawdata1$TimeOfDay <- factor(rawdata1$TimeOfDay, levels = c("dawn", "earlyAfternoon", "earlyMorning",
"evening", "lateAfternoon", "lateMorning", "midnight", "night"))

rawdata1$DayOfWeek <- factor(rawdata1$DayOfWeek, levels = c("Monday", "Tuesday", "Wednesday", "Thursday",
"Friday", "Saturday", "Sunday"))

# Run the linear regression
model1 <- lm(Productivity ~ TimeOfDay + DayOfWeek + HoursInClasses + HoursFocused, data = rawdata1)

# Predict data
new_data <- data.frame(
  TimeOfDay = "evening",
  DayOfWeek = "Friday",
  HoursInClasses = 3,
  HoursFocused = 7
)

predict(model1, new_data)
```

```
       1
48.97736
```

The produced output has the value of 48.98. This obtained answer will be the expected value to be returned in our unit test, with precision to three digits.

With the expected output determined, we write the unit test in Jest to validate our JavaScript implementation of the model. This test ensures that our model produces results consistent with the R implementation, verifying the accuracy and precision of our algorithm.



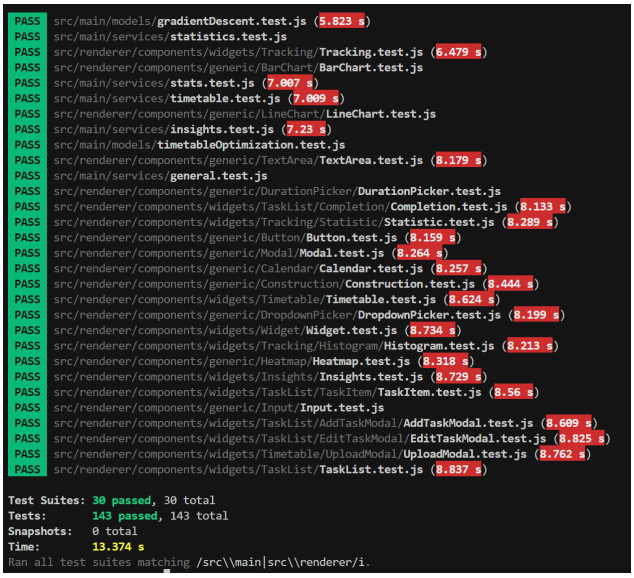
```
PASS src/main/models/gradientDescent.test.js
  Gradient Descent Prediction
    √ predictProductivity should return a valid result (183 ms)
    √ predictProductivity should return a valid result (200 ms)
    √ predictProductivity should return a valid result (263 ms)
```

Gradient descent model passes unit tests

Integration Testing

Overview

Integration testing ensures that various modules and services within our application interact correctly. Our integration testing plan aims to verify the seamless operation of integrated units, and we use Jest for these tests as well.



All integration (and unit) tests passing

Integration testing involves testing the interactions between integrated units or modules to detect any issues arising from their combined operation. This type of testing focuses on ensuring that the interfaces and communication between modules function correctly, thus maintaining the application's reliability and functionality.

Case Study

An example of our integration testing process would be for the TaskList widget. This widget is a critical component of our application, allowing users to manage their tasks efficiently. The TaskList widget consists of multiple subcomponents, each with its own responsibilities and unit tests to ensure their functionality.

Said widget is composed of the following subcomponents:

- AddTaskModal: A modal dialog that appears when a user wants to add a new task
- Completion: Indicates the completion status of all tasks, giving users a high level overview of their progress
- EditTaskModal: A modal dialog that appears when a user wants to edit a task
- TaskItem: The component that actually displays the content of a task

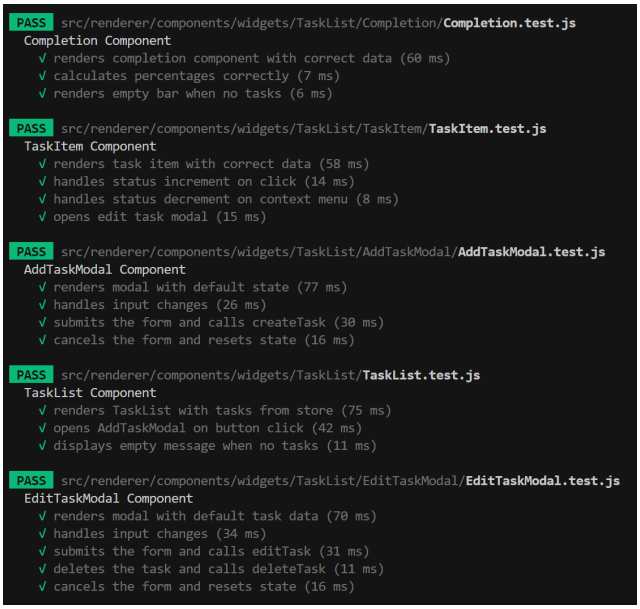
These subcomponents can be further broken down into a series of generic components such as an Input, Modal, TextArea, Button etc.

In the integration test for this widget, we test for general situations such as:

- Renders task items correctly with tasks from a mock Redux store
- Opens AddTaskModal correctly on button click
- Renders “Create a new task to get started!” message when no tasks are present

More specific test cases are tested in the unit tests of each individual subcomponent, such as:

- Handles status increment on click (TaskItem)
- Opens edit task modal (TaskItem)
- Calculates percentages correctly (Completion)
- Handles input changes (AddTaskModal)



All unit and integration tests for TaskList pass

System Testing

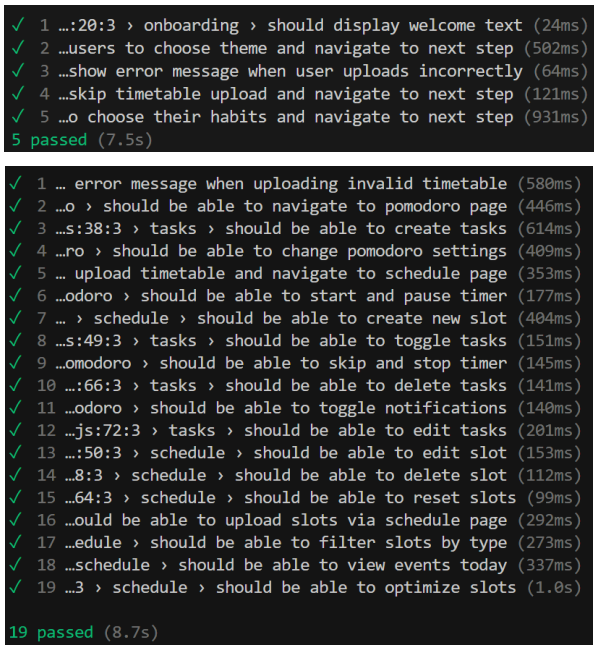
Overview

System testing involves testing the system as a whole, simulating real-world interactions with the app to identify potential issues. This phase of testing ensures that the application functions correctly in an environment that closely mirrors how it will be used in production.

We use Playwright to perform our system testing, executing tests that cover complete user workflows from start to finish. This comprehensive approach ensures that all integrated components work together seamlessly and meet the expected performance and usability standards.

We have implemented system tests for four user flows:

- **User onboarding:** Test the complete flow of user onboarding
- **Task management**: Verify the creation, editing, deletion, and toggling of task status.
- **Timetable and scheduling**: Test the NUS Mods import feature, schedule optimization feature, and the addition, editing, and deletion of timetable slots
- **Pomodoro timer**: Test the basic controls of the Pomodoro timer, including play, stop, and skip



All system tests passing

Focus on User Testing

While we acknowledge the importance of testing various features such as statistics, settings, and insights, these areas typically involve minimal user interaction. As a result, our primary

focus will be on user testing to ensure a seamless and intuitive user experience. These tests will be conducted through controlled user sessions and feedback collection, allowing us to refine and improve the application's usability.

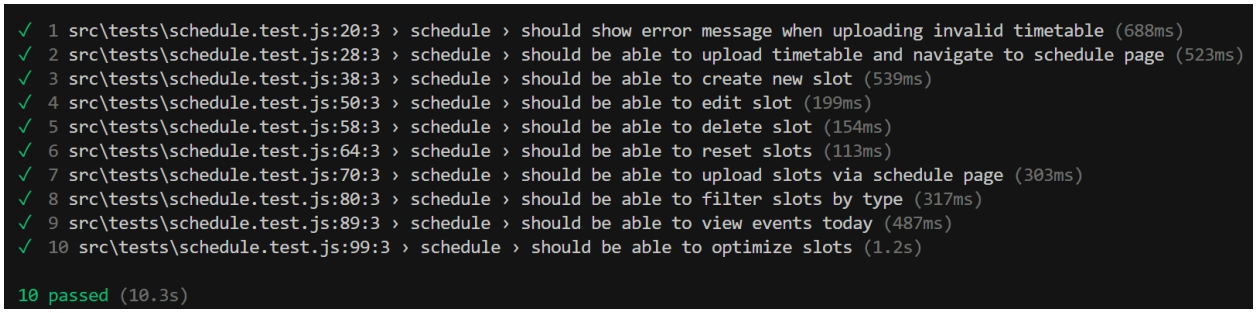
Case Study

An example system test would be one for the timetable and schedule:

Test Case	Steps	Expected Outcome
Uploading an invalid timetable	1. Launch the Electron app. 2. Click on the "Upload" button. 3. Fill the timetable URL input with an invalid URL. 4. Click "Submit" and check for the error message. 5. Click "Cancel" to dismiss the error.	An error message "Error uploading, please retry" should be displayed when an invalid URL is submitted.
Ensure the user can upload a valid timetable and navigate to the schedule page	1. Click on the "Upload" button. 2. Enter a valid NUS Mods timetable URL. 3. Submit the URL. 4. Navigate to the schedule page by clicking the schedule icon.	The timetable should be uploaded successfully, and the user should be able to navigate to the schedule page.
Verify that a user can create a new schedule slot	1. Click on "Add Slot". 2. Fill in the title, description, and time range. 3. Select the day of the week. 4. Submit the new slot.	The newly created slot should be visible in the schedule.
Ensure that a user can edit an existing slot	1. Select the slot to be edited. 2. Change the title of the slot. 3. Submit the changes.	The slot should be updated with the new title, and the old title should no longer be visible.
Verify that a user can delete a slot	1. Select the slot to be deleted. 2. Click the "Delete" button.	The slot should be removed from the schedule.
Ensure that a user can reset all slots	1. Click on the "Reset" button. 2. Confirm the reset action.	All slots should be cleared from the schedule.
Ensure that timetable slots can be uploaded from the schedule page	1. Click on the "Upload" button on the schedule page. 2. Enter a valid NUS Mods timetable URL. 3. Submit the URL.	The timetable should be uploaded successfully, and the slots should be visible.
Verify that the user can filter slots by type (Tasks,	1. Navigate to the schedule page. 2. Click on "Tasks" to filter the	The schedule should display the correct slots based on the selected

Classes, All scheduled)	slots. 3. Click on "Classes" to filter the slots. 4. Click on "All scheduled" to view all slots.	filter.
Verify that the user can view events scheduled for today	1. Add a new slot for today. 2. Check if the slot is visible in the schedule for today.	The slot should be visible under today's schedule.
Ensure that the user can optimize task slots within their schedule	1. Navigate to the dashboard. 2. Add a new task. 3. Navigate back to the schedule page. 4. Optimize the schedule. 5. Verify that the task is optimally placed in the schedule.	The task should be visible in the schedule after optimization.

Note that each test case is run consecutively one after the other, the app should only be launched once on the start of the first test case.



All timetable tests passing

User Testing

Overview

User testing is a vital component of our testing strategy, focusing on gathering feedback from real users to identify usability issues and gather suggestions for improvement. Our user testing plan involves reaching out to our target audience (NUS students) outside the development team to use the application in real-world scenarios.

Our strategy includes collecting feedback from peer evaluations and directly from users via surveys and feedback forms. This feedback helps us identify and address any usability issues and incorporate user suggestions to enhance the application's user-friendliness.

This is done by preparing a Google form with various questions (shown in the Results section below), providing a select group of users (NUS students from different faculties. Including non-computing students) with a simple walkthrough and the Milestone 3 build of our project. After which, we collate the reviews and ratings from each user and summarize them in the section below.

Each tester is only responsible for testing 3-4 features of the app, in order to not overwhelm them. However, we have given them the option to test more features if they wish to do so.

We chose to let users evaluate Stucado based on the 7 Factors of User Experience by Peter Morville.

The 7 factors include:

1. **Useful**: A feature is considered useful if it serves a purpose and provides value to the user. It should be relevant and fulfill a specific need or solve a problem.
    a. Questions:
        i. Does this feature meet a specific need or solve a problem you have?
        ii. Do you find the feature relevant to your tasks or goals?
2. **Usable**: A feature is usable if it is easy to use and helps users achieve their goals efficiently and effectively. There should be minimal hassle or confusion.
    a. Questions:
        i. How easy is it to use this feature?
        ii. Were there any difficulties or frustrations when using this feature?
        iii. How quickly can you complete your tasks using this feature?
3. **Findable**: A feature is findable if users can easily locate it and the information or tools they need within it.
    a. Questions:
        i. How easy is it to find this feature in the app?
        ii. Was there ever a time when you struggled to locate this feature?
        iii. Are the instructions or labels clear and helpful?

4. **Credible**: A feature is credible if users perceive it as trustworthy and reliable. It should appear professional and provide accurate and trustworthy information.
    a. Questions:
        i. Do you trust the information or functionality provided by this feature?
        ii. Have you encountered any errors or issues that affected your trust in this feature?
        iii. Does the feature appear professional and well-designed?
5. **Accessible**: A feature is accessible if it can be used by people with a wide range of abilities and disabilities.
    a. Questions:
        i. Can you use this feature comfortably without any accessibility tools or settings?
        ii. Are there any accessibility features or options that make using this feature easier for you?
6. **Desirable**: A feature is desirable if it is appealing to users.
    a. Questions:
        i. Do you find this feature visually appealing?
        ii. Does this feature make you enjoy using the app more?
7. **Valuable**: A feature is valuable if it delivers benefits to users and contributes to the overall success of the app. It should enhance the user's experience and provide tangible benefits.
    a. Questions:
        i. How does this feature add value to your overall experience with the app?
        ii. How does this feature impact your productivity or enjoyment?

Results

User story	Task	Discovery	Rating	Feedback
As a user who enjoys personalizing my workspace, I want to be able to set up and customize my dashboard and make it visually appealing.	Setup and personalize the dashboard.	1. Testers are able to choose the display mode of the dashboard. 2. Testers are able to upload their NUSMods timetable. 3. Testers are able to choose their study habits.	Useful: 4.5 Usable: 5 Findable: 4.5 Credible: 5 Accessible: 4 Desirable: 5 Valuable: 4	Useful: The onboarding feature serves users' need for a personalized and organized dashboard. It allows them to choose their preferred mode of viewing. Usable: Customizing the dashboard is straightforward and hassle-free. The icons are clear. Findable: The customization options are easy to locate and clearly labeled, making the process intuitive and efficient.

				Credible: The customization feature is reliable and functions as expected, reinforcing users' trust in the app's capabilities. Accessible: The feature is comfortable to use without needing any special accessibility tools. Users recommended adding shadows to make choice options stand out more. Desirable: The feature is visually appealing, the graphics in the options enhance users' overall experience and leaves a good first impression of the app. Users also stated that the dark mode settings help to keep their eyes comfortable. Valuable: Personalizing dashboard adds significant value to users' experience as it allows users to create an appealing and efficient study environment.
As a student who needs to stay on top of deadlines and tasks, I want to be able to create and manage a to-do list within the system, allowing me to prioritize and track my tasks effectively.	Create new tasks and manage the tasks in the to-do list.	1. Testers are able to create, edit and delete tasks. 2. Testers are able to adjust the estimated time for each task. 3. Testers are able to right click on the task button to update the status of the task. 4. Testers are able to left click to retract the status of the tasks.	Useful: 5 Usable: 4 Findable: 5 Credible: 5 Accessible: 4 Desirable: 4.5 Valuable: 5	Useful: The to-do list feature meets users' need to stay organized and on top of their deadlines. It's highly relevant and essential for students. Usable: The task management feature is user-friendly. Users can create, edit, and delete tasks easily. Also, the progress bar provides a clear overview of their task completion status. However, some users could not manage to figure out how to revert the task status using right click, clarity could be improved. Findable: Users can easily find and access the task management feature. The interface is intuitive, and everything is where they expect it to be.

		5. Testers are able to see the progress bar that shows the completion status of the task list.		Credible: The feature works reliably and provides accurate information about their tasks and status. Accessible: Users are able to use this feature comfortably without any accessibility tools, and it's easy to navigate and operate. Users have also stated that the traffic light system is intuitive. Desirable: The task management feature is well-designed and visually appealing. Valuable: This feature is one of the most significant features in the app, as it enhances users' productivity by helping them to prioritize and track their tasks effectively. Users have stated that it would help with the procrastination problem.
As a student at NUS who wants to efficiently manage my academic schedule and maximize my study time, I want to be able to import my NUS mods timetable into the system and fit all my tasks into available time slots, so I can easily view and organize my classes and tasks.	Import / update NUS Mods timetable. Optimize timetable after creating tasks.	1. Testers are able to upload / update the NUS Mods timetable in the homepage. 2. Testers are able to upload / update the NUS Mods timetable in the Timetable page. 3. Testers are able to edit the details of the classes in the Timetable page.	Useful: 5 Usable: 4 Findable: 4 Credible: 4.5 Accessible: 4 Desirable: 4.5 Valuable: 5	Useful: Importing NUS Mods timetable allows users to plan their schedule without the need to direct to NUS Mods (manual scheduling). Users also stated that it is convenient as it helps them in arranging their daily schedule. The timetable optimization function is relevant, but could add another option allowing users to add breaks between "optimized" tasks. Usable: Uploading and updating the timetable is easy and efficient. Users can quickly adjust and view their schedule without any hassle, such that they don't need to add classes one by one. However, users said that we could consider adding another "optimize" button in the homepage. One user had issues uploading their timetable. Findable: One user recommended

				making the upload timetable function (in the home page) more obvious/clear. Credible: The timetable accurately reflects users' schedule, and the 'Pending' tasks are added into the timetable clearly. One user felt that the text sizing/flow could be improved. Accessible: The feature is accessible and straightforward to use. Desirable: Having timetable integrated into the app is visually appealing and adds to the overall functionality, making it more enjoyable to use. Valuable: The timetable and timetable optimization feature adds great value by streamlining their academic schedule management, saving time and effort.
As someone who struggles with maintaining focus during work or study sessions, I want to be able to use a Pomodoro timer feature in the system to enhance productivity.	Start a Pomodoro timer	1. Testers are able to start, pause and stop a pomodoro timer. 2. Testers are able to edit the pomodoro timer to adjust the work duration, short break duration and long break duration. 3. Testers are able to turn on and turn off the notification of	Useful: 4 Usable: 4.5 Findable: 4 Credible: 4.5 Accessible: 4.5 Desirable: 4.5 Valuable: 5	Useful: The Pomodoro timer is very useful, customizing the duration of work adds more variation to the feature. Usable: Starting, pausing and stopping the timer is very easy. Customizing the duration of work and break periods is straightforward. Findable: The feature is easy to locate. The instructions are clear and helpful. Credible: The timer works reliably without any issues. Accessible: Users can use the Pomodoro timer comfortably, and the option to turn on and off the notification adds to its accessibility.

		the pomodoro timer.		Desirable: The feature is visually appealing and makes the app more engaging and enjoyable to use. Valuable: The Pomodoro timer allows users to stay focused, making it a valuable addition to the app.
As a student who wants to optimize my schedule, I want the system to predict my productivity and provide me helpful insights and recommendations at different times of the day so I can plan my tasks for my most productive periods.	View insights and predicted productivity in the homepage.	1. Testers are able to view the predicted productivity after selecting their study habits.	Useful: 4.5 Usable: 4.5 Findable: 4 Credible: 4.5 Accessible: 4.5 Desirable: 5 Valuable: 4.5	Useful:The predicted productivity value provided based on users' study habits helps users optimize their study schedule Usable: Viewing and understanding the insights is easy. The information is presented clearly and is actionable. Findable: The insights widget is easy to find in the homepage and is displayed prominently. Credible: The insights are quite reliable, as it will get more and more accurate in the future. Accessible: The insights are easy to read and understand. Desirable: The feature is visually appealing and adds a motivational aspect to the app, making it more engaging for the users. The illustration makes the app feel more interesting and playful. Valuable: The productivity insights help users make better decisions about their study habits, adding significant value to their study schedule.
As a student who wants to quickly gauge my performance, I want	View short-term statistics in the	1. Testers are able to view statistics on that day to	Useful: 4 Usable: 4 Findable: 4.5 Credible: 4.5	Useful: The short-term statistics give users a quick overview of recent performance, which is useful for immediate adjustments

short-term, specific statistics to be displayed on my dashboard, so I can see my recent study hours, productivity, and task completion at a glance. As a student who wants to reflect on my study habits over time, I want to be able to view long term statistics so I can understand my productivity and study patterns over weeks.	homepage. View long-term statistics in the Statistics page.	gauge the hours focused, tasks completed and average productivity in the homepage. 2. Testers are able to view previous statistics as bar charts in the homepage after generating test data. 3. Testers are able to view past long-term statistics in the Statistics page after generating test data. 4. Testers can customize the time frame for displayed statistics, opting to view data from the past 30, 14, or 7 days in the statistics widgets.	Accessible: 4 Desirable: 4 Valuable: 4.5	and planning. The long-term statistics help users understand their study patterns over time. Usable: The statistics are easy to view and very user intuitive. The bar charts are informative. The customizable time frame in the long-term statistics adds flexibility. Findable: The statistics are easy to find, the icons and labels are clear. It would be better to show the name of the feature when hovering over the icons in the sidebar. Credible: The data is accurate and trustworthy. The feature is professional and well-designed. Accessible: The statistics are accessible and easy to read, the detailed information that is shown in the long-term statistics after hovering to the charts is useful. However, the button to change the time frame looks like a refresh icon which may be a little confusing. Desirable: The feature is visually appealing, the bar charts in the home page are not too messy, and the line charts provide a comprehensive overview of users' study habits. Users have also stated that the statistics page looks appealing and well-structured. Valuable: The statistics feature provides valuable insights to users' study habits, helping them understand their productivity and study patterns.
As a student who wants to personalize my experience and manage my data, I want to be able to	Edit preferences and delete data	1. Testers are able to change the display theme and colors.	Useful: 4.5 Usable: 5 Findable: 3.5 Credible: 5 Accessible: 4	Useful: The ability to edit preferences and manage data meets users' need for personalization and control over their information. The hyperlink

edit my settings and have the ability to delete my data.		2. Testers are able to enable notification for pomodoro sessions 3. Testers are able to delete all data in the app. 4. Testers are able to direct to other websites in one click by keying in the desired link in the settings page.	Desirable: 4.5 Valuable: 5	function was useful. However, could consider allowing users to upload their profile picture. Usable: Changing settings and managing data is straightforward and easy to do.Instead of allowing users to only add one external link, could consider allowing users to add more links so that they can direct to more websites. Findable: One user suggested changing the settings tab icon to a classic “gear” instead, which would improve discoverability. Credible: The settings work reliably, and the ability to delete data builds trust in the app’s respect for user privacy. Accessible: The settings are easy to use and include accessibility options that enhance the user experience. Desirable: The feature is well-designed and visually appealing, adding to the overall enjoyment of using the app. Valuable: The ability to customize settings and manage data adds significant value by giving users control over their experience and information.

Some general feedback:

- Users suggested showing the name of a page when hovering over the icon in the sidebar to improve clarity
- The dashboard looks slightly messy, but still acceptable

Unfortunately, one of our testers had issues uploading their timetable. Despite our efforts to recreate the bug, we could not do so. We suspect that this issue is caused by the NUS Mods API we are using to get class information, as we have encountered similar issues before which were solved the week after without our interference.

We suspect that the data that NUS Mods has in their servers are unavailable/outdated, since module information for the semester has just been freshly released. As such, we will continue to monitor the situation, but we will not be able to fix their issue for now.

Conclusion

Overall, the app has received positive feedback for its functionality, ease of use, accessibility and value. Users find the features useful and relevant to their needs, with an intuitive and appealing interface that enhances their productivity and overall experience.

User guide

Installation

Unfortunately, this current version is only available for Windows. There are two ways to install:

Self-contained (recommended, can be removed by deleting the folder):

1. Download Stucado-Unpacked.zip here
2. Extract (unzip) into a folder
3. Open stucado.exe
4. Everything is self-contained in this folder, just delete the folder after done

Installer (can be uninstalled by add/remove programs in settings):

1. Download Stucado-Installer.exe here
2. Start the installer and run through the installation process
3. Open stucado.exe
4. To uninstall, use the "Add or remove programs" feature from Windows

Development Timeline

MS	Tasks	Description	In Charge	Date
1	Setup project	Install necessary packages and setup Github	HT, JL	15-17 May
		Rough mockup of project directory structure, project architecture structure		
	Ideation and research	Familiarize with necessary technologies (Electron, React, Typescript, SQLite)	HT, JL	15-20 May
	Prototype core features	Multivariate regression model implementation (gradient descent)	JL	17-23 May
		To-do list	HT	
		Timetable	HT	
	Expand prototype core features	Database and cache using SQLite	JL	23 May-03 June
		Get NUS Mods API timetable	HT	
		Integrate regression model with frontend	HT, JL	

Evaluation Milestone 1:
- Ideation
- Proof-of-concept:
  - Tasks list with updating task status, saving to cache
  - Timetable with NUS Mods import, saving to cache
  - Basic demonstration of regression model prediction				

2	User interface	Mockup final Figma design of UI	HT	03-10 June
		Implement UI designs in app		
	Database	Format and transfer data from cache to database	JL	03-10 June

<table>
<tr><td rowspan="6"></td><td rowspan="2">Expansion features</td><td>Pomodoro timer widget, and data caching</td><td rowspan="2">HT, JL</td><td>11-18 June</td></tr>
<tr><td>Statistics widget, and statistics generation</td><td></td></tr>
<tr><td>Timetable optimization</td><td>Machine learning model for optimizing timetable slots</td><td>JL</td><td>18-25 June</td></tr>
<tr><td rowspan="2">Pages and onboarding</td><td>Create individual pages (settings, insights etc.)</td><td rowspan="2">HT</td><td rowspan="2">18-25 June</td></tr>
<tr><td>New user onboarding process and basic personalization features</td></tr>
<tr><td>Testing and debugging</td><td>Unit tests, system tests and basic user testing</td><td>HT, JL</td><td>25-30 June</td></tr>
<tr><td colspan="5">Evaluation Milestone 2: Semi-polished prototype
- Polished UI
- Core widgets completed:
  - Tasks List
  - Timetable
  - Pomodoro
  - Statistics
  - Insights
- Backend completed:
  - Database, cache
  - Machine learning models (linear regression and timetable optimization)</td></tr>
<tr><td rowspan="4">3</td><td>Testing</td><td>Extensive system and user testing</td><td>HT, JL</td><td>01-15 July</td></tr>
<tr><td>System documentation</td><td>Code refactor and organization, redo Github README</td><td>HT, JL</td><td>16-29 July</td></tr>
<tr><td>Extra features</td><td>Minigame widgets, personalization widgets</td><td>HT</td><td>01-29 July</td></tr>
<tr><td>Polish</td><td>Draw app icon, logo, mascot etc.</td><td>JL</td><td>01-29 July</td></tr>
</table>

Evaluation Milestone 3: Fully working app

- Fully polished app
- Some extra features:
    - Minigame widgets
    - Personalization widgets
- Fully documented system

Relevant Links

Project Log: A detailed record of all project activities, progress, and hour tracking.

Project Poster: A visual summary showcasing the key aspects and achievements of the project.

Project Video: A short video presentation highlighting the project's goals, development, and features.

Github repository: Stucado’s source code. Download links for the installer and executable can be found here.