UNSW

SCIENTIA

MANU ET MENTE

COMP3900 Computer Science Project

Final Report

2021 Term 2

By: Team Placeholder

James Dang z5209597 (Frontend)

John Dao z5258962 (Scrum Master and Backend)

Jake Edwards z5114769 (Frontend)

Edward Gauld z5246767 (Backend)

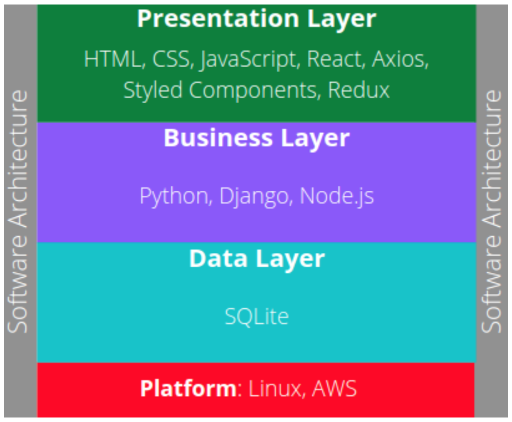
Jaydon Tse z5214494 (Backend)

The final submission of the Task Master project is intended to be run on selection two of the two given environments (Lubuntu 20.4.1 LTS virtual machine image).

1: Architecture and design of the overall system and functionalities

1.1: The Tech Stack

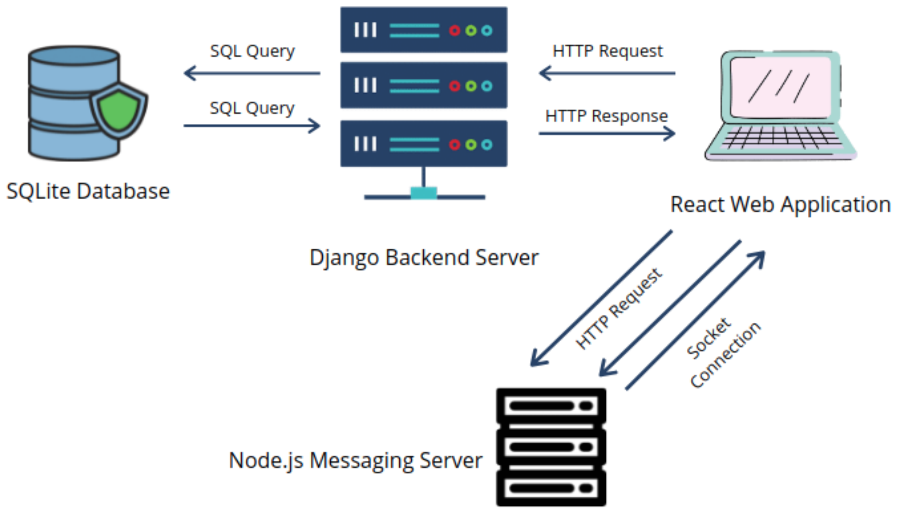
TaskMaster was built as a full-stack solution with React as the frontend, Django and Node.js as the backend, and SQLite as the database. To perform CRUD operations, we used the REST API design, where the frontend calls endpoints from the backend to retrieve and update data.



Software Architecture of the TaskMaster system

In essence, React (Presentation Layer) would be the visual layer users interact with, in the form of a web application. Here, end-users will be able to enter their input to either delete, update, retrieve and create data. Moving down the layer, a Django server (Business Layer) would handle user input and handle the operations intended by the end-user. This layer will then interact with the SQLite database (Data Layer), to produce a final output to the end-user.

In addition to the business layer, we had a Node.js server that would handle real-time messaging with other users. In contrast to the Django server, it does not interact with the Data Layer. Instead, it directs messages with different users by utilising web sockets and establishing connections between them. More information can be found in the frontend third-party library functionalities.



TaskMaster’s technology ecosystem

1.2: Frontend Structure

The frontend would be a web application using HTML, CSS, JavaScript and React.js, where users will be able to access it from their web browsers. This was initialized using React’s most popular framework Create React App, which is ready to start from installation and allows us to kickstart development immediately.

For the frontend structure, we separated the code into four main components:

- **Pages** - Index pages for unique and individual routes (e.g /login)
- **Components** - Reusable components that will be shared across their parent components and pages (e.g button)
- **API** - Functions that call the backend to perform CRUD operations
- **Redux** - Manages the global states that will be shared across different pages and components

By doing so, we were able to decentralise the dependencies of different features, provide a balanced structure for the code and ensure its maintainability for future development.

Diving deeper into the frontend’s state and data management, localStorage and Redux was used to cache data and prevent extra congestion from being forwarded to the Django servers. For example, an end user’s username will never change, so we store it in localStorage until the user logs out. It also improves the user experience by not needing to poll the server whenever reused data was needed.

Overall, these design choices and considerations were made to ensure the maintenance and integration of new features to the codebase was seamless. By providing a framework for the team to follow, we were able to follow a structured pattern that was consistent throughout its development.

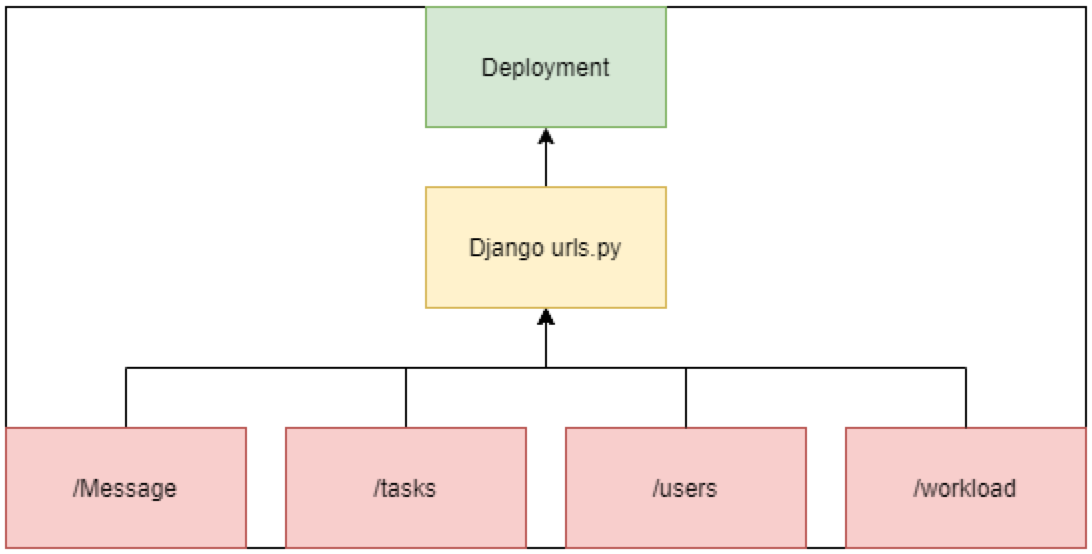
1.3: Backend

The chosen backend framework for Task Master has been Django and the Django Rest Framework. This was selected due to the high modularity of packages that accompany Django which allowed for shorter development times, increasing the efficiency of the backend development process. This was especially important as the frontend team relied on the capabilities of the backend during their development.

The basic structure of the backend follows a module system that congregates towards Django’s rest framework which broadcasts the features as backend endpoints. These modules are:

- Friendships
- Message
- Tasks
- Users

These modules are supported by an SQL database with all querying, item creating and table management is done by Django itself. These modules combine together to provide the functionality required to fully integrate Task Master’s frontend system. These modules are then combined together within Django’s ‘urls.py’ which routes the endpoints that are broadcasted by the Django server and allows the frontend development team to access their functionality. This congregation is represented in the diagram below.

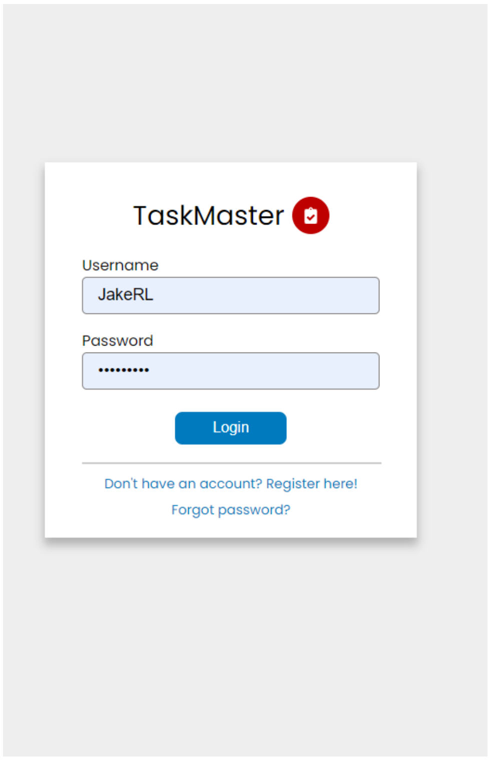


2: Descriptions of the functionalities developed by the team and how they map/address all project objectives

2.1: Profile Creation and Viewing Profiles

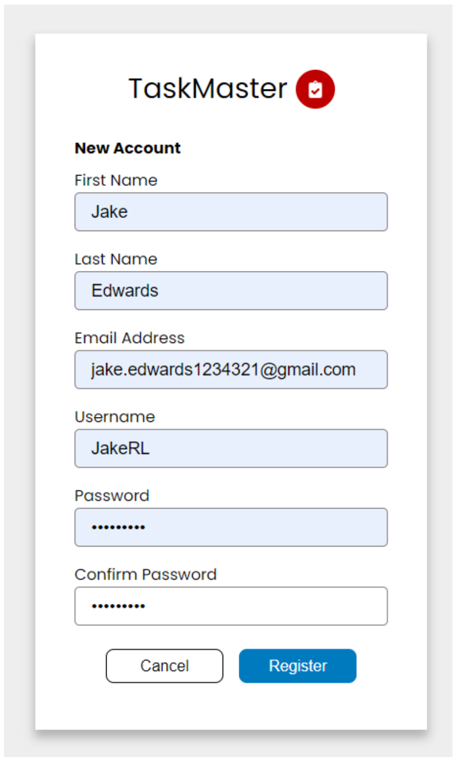
For users to create and maintain their own profile, we have implemented user registration and login form to put in their details, and to use the service respectively. For our login page, a user can choose to create an account, to log in using their username and password, or to fill in a form to reset their email in case they have forgotten it.

The task registration form has all the necessary details associated with an account. These include the user's name, email, username, and password. These fields will be used either for logging in, or for display on their user profiles. See the figures below the left showing the login form and the right the registration form.


TaskMaster

Username: JakeRL

Password: •••••••••


TaskMaster

New Account

First Name: Jake

Last Name: Edwards

Email Address: jake.edwards1234321@gmail.com

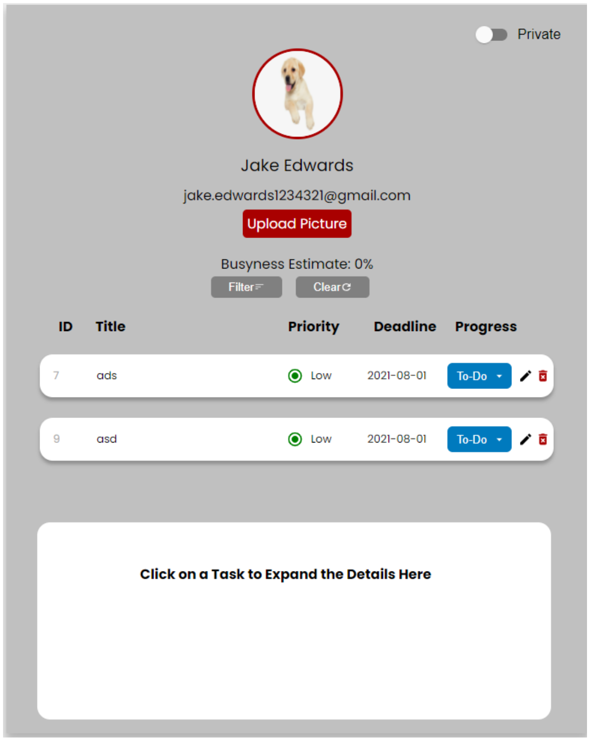
Username: JakeRL

Password: •••••••••

Confirm Password: •••••••••

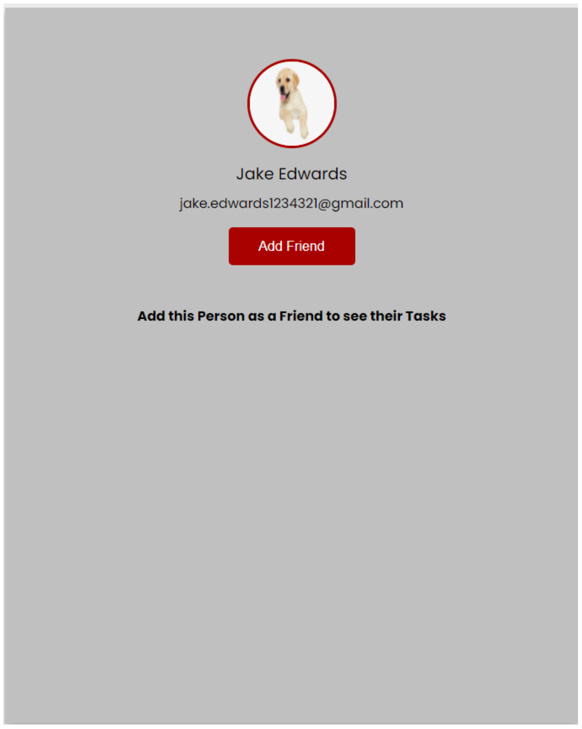
When a user is logged in, they can access their profile from the main page, which will be covered in more detail later. This profile clearly communicates the name, and email of the user. Through the search bar on the main page, users can search and view other profiles through the user's email address. The left figure below shows a user's own profile. Additional novelty implemented in the profile is the ability to make the profile private, as seen in the switch at the top right. This means that a users profile will not be shown when it is searched for. To add to this, we also implemented the ability for a user to upload a profile picture to identify themselves easier. Lastly, the profile shows the assigned tasks to the user, which will be discussed in detail below.

The figure to the right shows the view of a user's profile that has been searched for. This user’s details are shown, however you are not able to see their tasks until you add them as a friend. The process of adding connections will be discussed in the next section.



ID	Title	Priority	Deadline	Progress
7	ads	Low	2021-08-01	To-Do
9	asd	Low	2021-08-01	To-Do

Click on a Task to Expand the Details Here



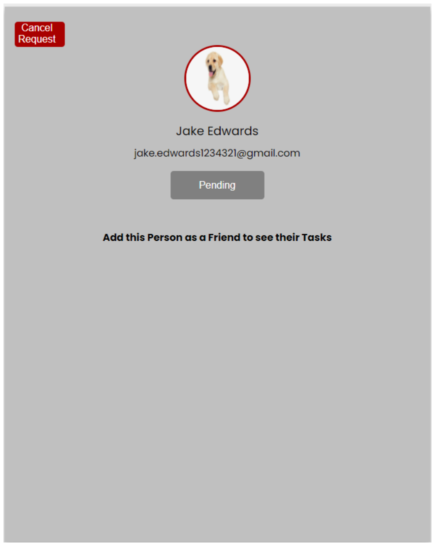
Add this Person as a Friend to see their Tasks

Overall, the login pages and profiles are designed to satisfy the requirement that - **Task masters must be able to maintain a profile where they specify their name and contact details (email address).**

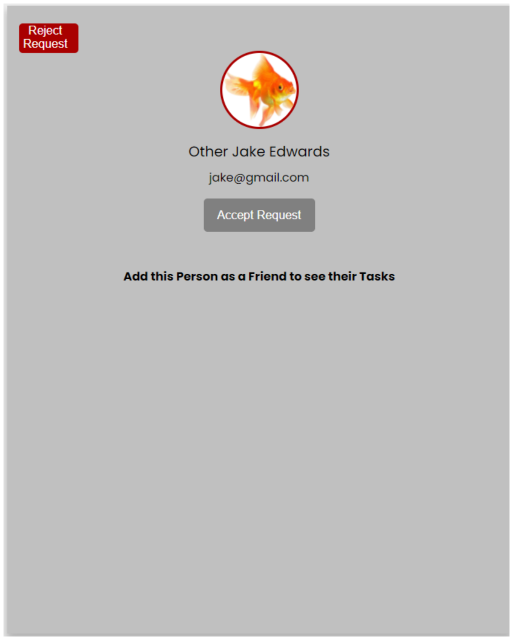
Connections

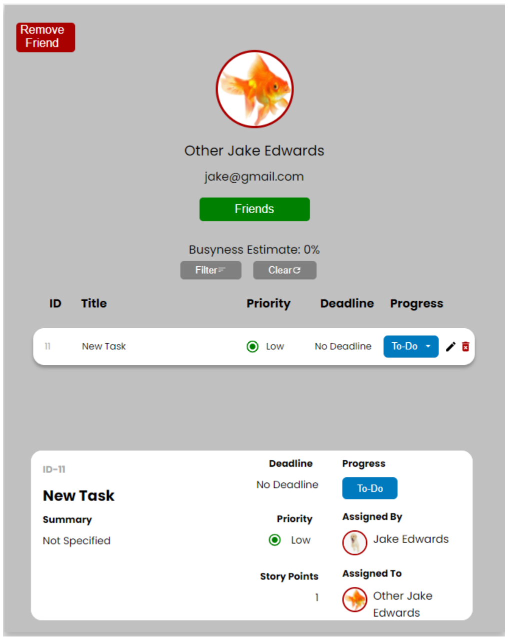
To manage connections in the app, we have users search for their desired connections through email on the main page. This will take them to the users profile for them to add them from there.

Once the ‘Add Friend” button is clicked, a friend request will be sent. The user can decide to cancel this friend request if they wish to, while it is in the pending state. This can be seen below.


Cancel Request
Jake Edwards
jake.edwards1234321@gmail.com
Pending
Add this Person as a Friend to see their Tasks

When a user has pending friend requests to handle, they can either manage this through the profile pages, or their friends tab on the main page. To accept a request through the profile page, a user must search for the profile page of the user that the request is sent from. There they will find a button to accept the friend request.

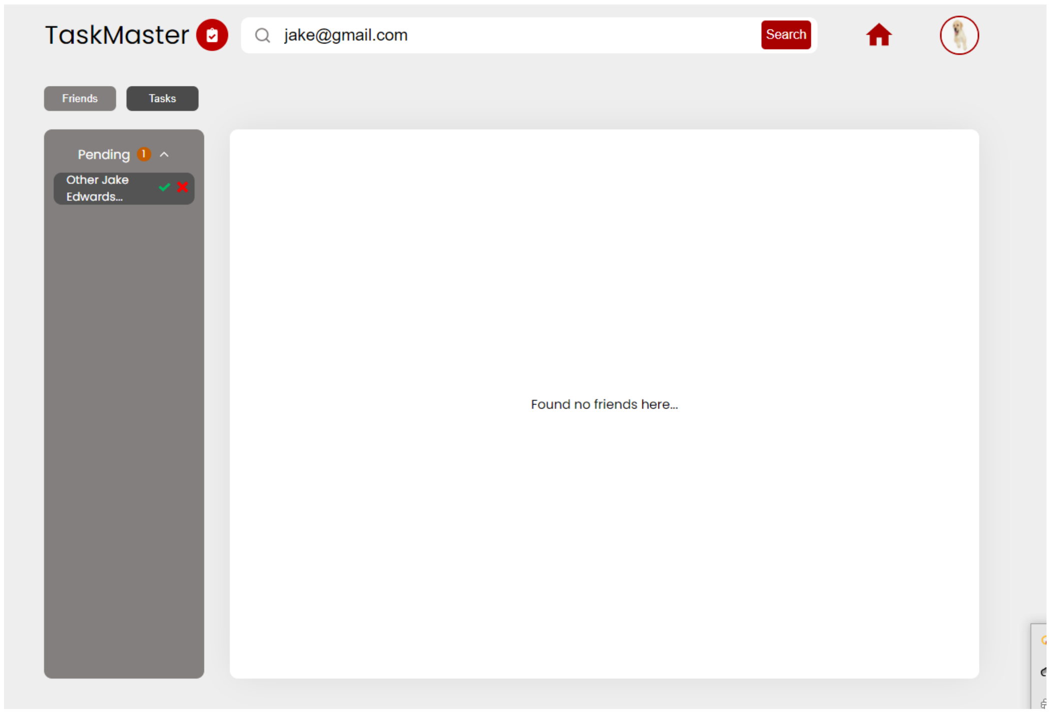




ID	Title	Priority	Deadline	Progress
11	New Task	Low	No Deadline	To-Do

ID-11
New Task
Summary
Not Specified
Deadline
No Deadline
Priority
Low
Story Points
1
Progress
To-Do
Assigned By
Jake Edwards
Assigned To
Other Jake Edwards

If a user wants to manage their friend requests from the main page, they can do it under the friends bar to the left. This shows a drop-down of pending friend requests, with the ability to accept from there. This feature was implemented to allow users to more easily manage their friend requests.

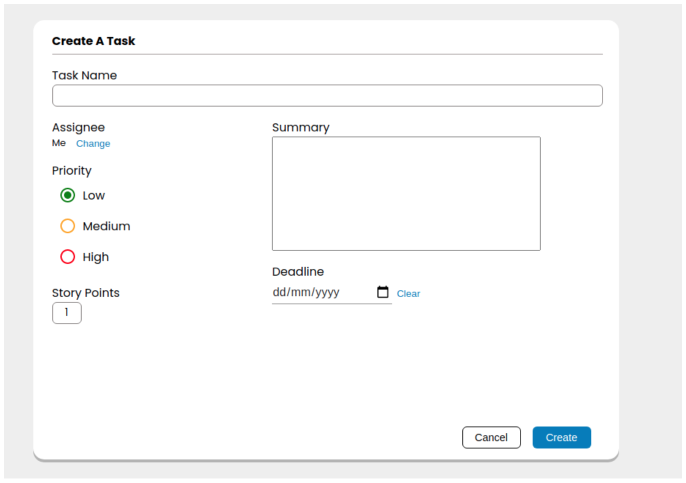


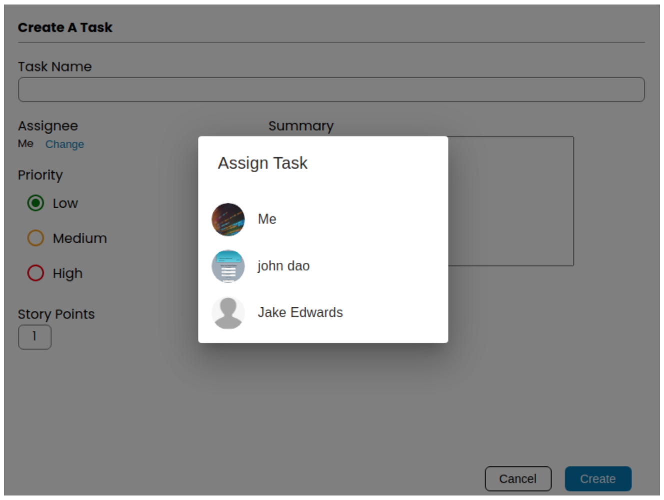
Found no friends here...

These connection features were implemented to satisfy the requirements that **Each Task Master must be able to request a connection with 1 or more other Task Masters (by email address), and also accept or decline connection requests from other Task Masters (two Task Masters are connected if one of them accepted a connection request from the other).** We additionally added extra functionality to this step including a more convenient method of managing friend requests from the main page and allowing users to cancel requests.

2.2: Task Creation

On the main page, users are given a 'Create A New Task' Button, where they will be redirected to the page to create a new task. On this page, users will be able to create a task including a title, description, story points and an optional deadline, which can be assigned to either themselves or one of the Task Masters they are connected with. This satisfies the objective - **Task Masters must be able to create a task, including a title, description, optional deadline, and assign it to either themselves or one of the Task Masters that they are connected with (each task must also have a system-assigned id that is shown and is to be assigned to the creating taskmaster if not assigned to anyone else).**

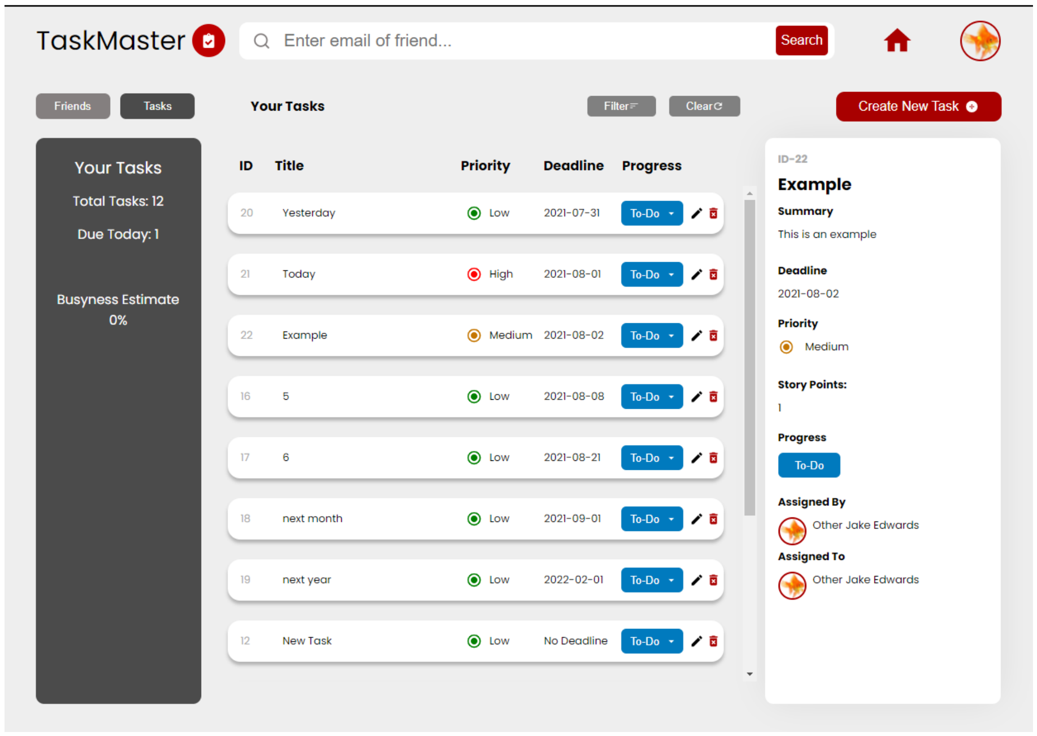




The first screenshot is the Create A Task form. The second screenshot shows the menu to assign tasks to connected users.

2.3: Viewing Tasks

On the main page, a user will be able to view all of their assigned tasks. On this page, you can easily see a summary of the task items, including the task id, title, priority, deadline, and state of progress. The tasks shown are all sorted by deadline. When a task is clicked, the details of the task will be expanded to the right of the screen, where additional information including the task summary, the story points, the assigner and the assignee are listed. Some statistics are additionally shown for the tasks to the left, including the total tasks due, and total tasks due today.



Your Tasks

Total Tasks: 12

Due Today: 1

Busyness Estimate
0%

ID	Title	Priority	Deadline	Progress
20	Yesterday	Low	2021-07-31	To-Do
21	Today	High	2021-08-01	To-Do
22	Example	Medium	2021-08-02	To-Do
16	5	Low	2021-08-08	To-Do
17	6	Low	2021-08-21	To-Do
18	next month	Low	2021-09-01	To-Do
19	next year	Low	2022-02-01	To-Do
12	New Task	Low	No Deadline	To-Do

ID-22

Example

Summary
This is an example

Deadline
2021-08-02

Priority
Medium

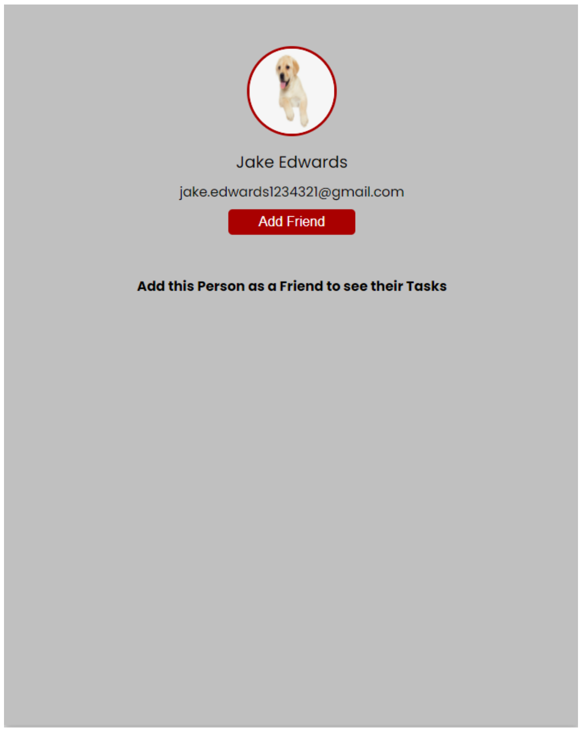
Story Points:
1

Progress
To-Do

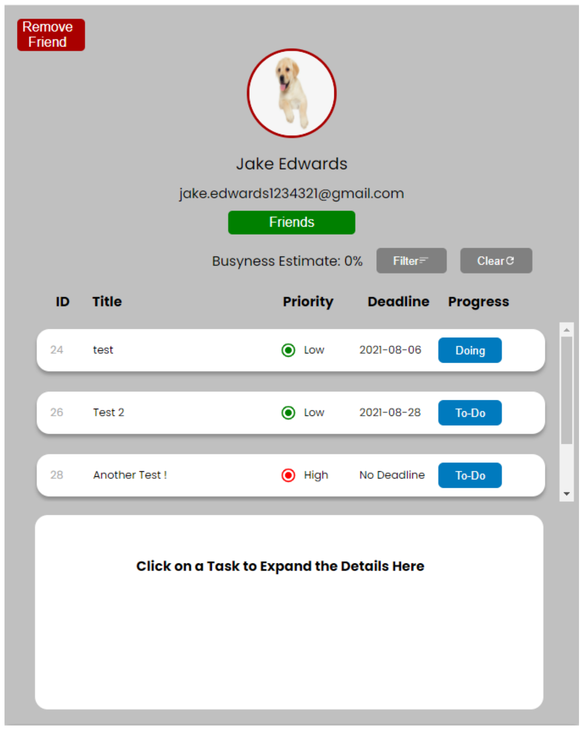
Assigned By
Other Jake Edwards

Assigned To
Other Jake Edwards

To complement this main task display, we also have the tasks shown on users profiles. Tasks are hidden by default if you are looking at a profile of someone you aren't a friend with. Once you are friends with them, you can see their tasks. You can always see your own tasks that are shown on your profile.



Add this Person as a Friend to see their Tasks



Busyness Estimate: 0%

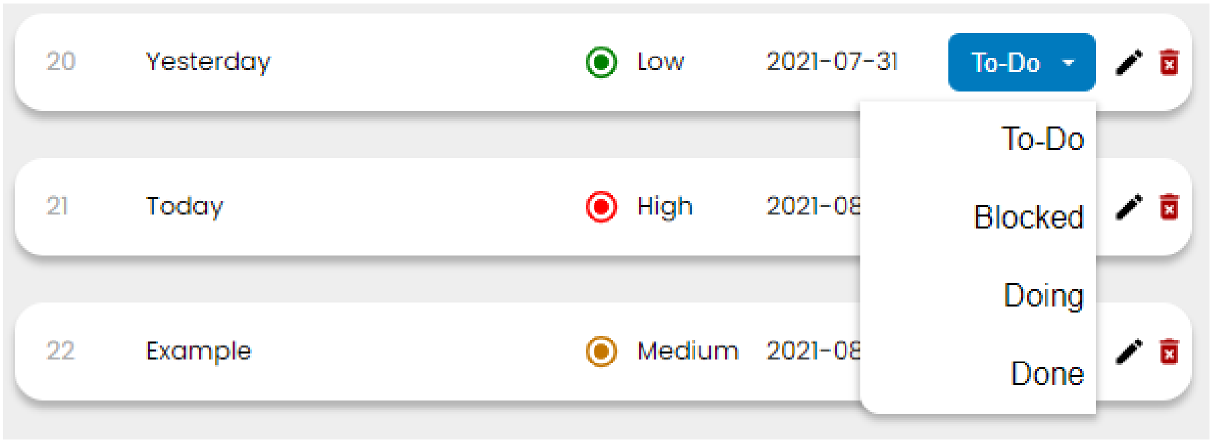
ID	Title	Priority	Deadline	Progress
24	test	Low	2021-08-06	Doing
26	Test 2	Low	2021-08-28	To-Do
28	Another Test !	High	No Deadline	To-Do

Click on a Task to Expand the Details Here

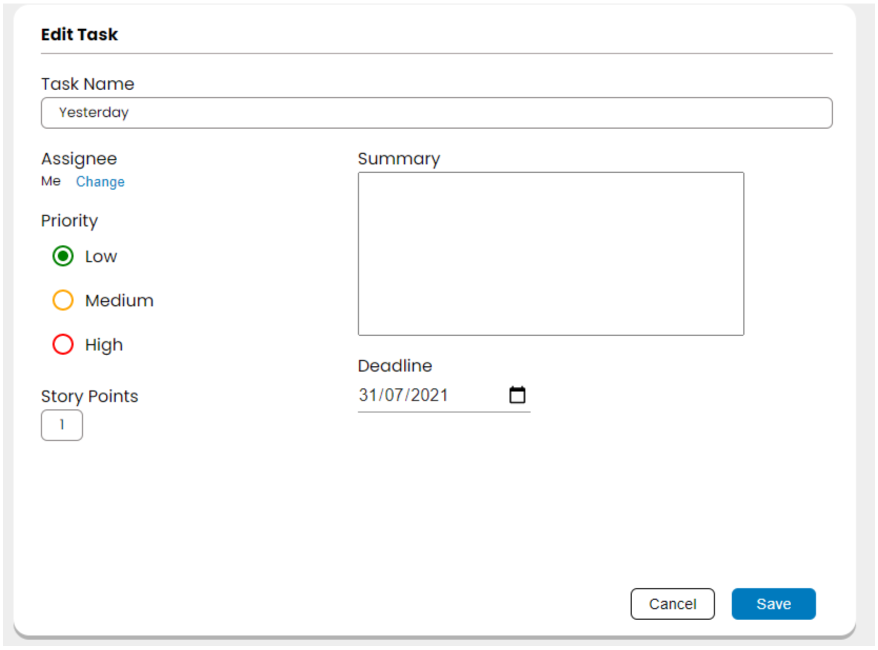
These functionalities address the project objective - **In order to view the work to be done by a Task Master, their profile must also include an "Assigned Tasks List", showing all of their assigned tasks (with each task showing summary details of the task ID, title, and deadline (if specified)), sorted by deadline (earliest to the latest deadline, with tasks not having a deadline appearing last). Each Task Master must be able to view details on the profiles of any other connected Task Masters.**

2.4: Update Task Details

Each task component can be edited either by toggling the state button or getting redirected to the edit page. The toggle state button allows users to easily update their tasks to reflect their current progress. This is done through a dropdown menu and is shown in the figure below.



Additionally, a user can click the edit button to the right of the state button and will get redirected to an edit page similar to the create a task page. This allows users to have more control over editing any field involved with the tasks.



Edit Task

Task Name: Yesterday

Assignee: Me Change

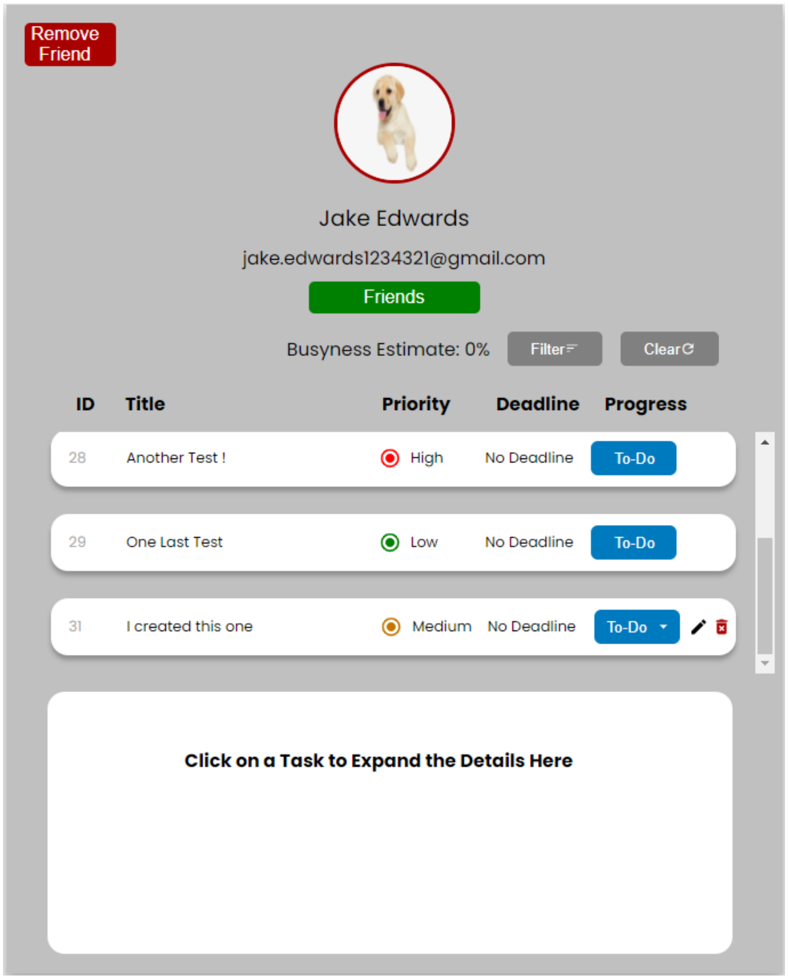
Summary:

Priority: [✓ Low] Medium High

Deadline: 31/07/2021

Story Points: 1

Lastly, when viewing the tasks assigned to those they are connected to, they can only edit the task if they were the ones to create it. An example of this is shown below.



Jake Edwards

jake.edwards1234321@gmail.com

Busyness Estimate: 0%

ID	Title	Priority	Deadline	Progress
28	Another Test !	High	No Deadline	To-Do
29	One Last Test	Low	No Deadline	To-Do
31	I created this one	Medium	No Deadline	To-Do

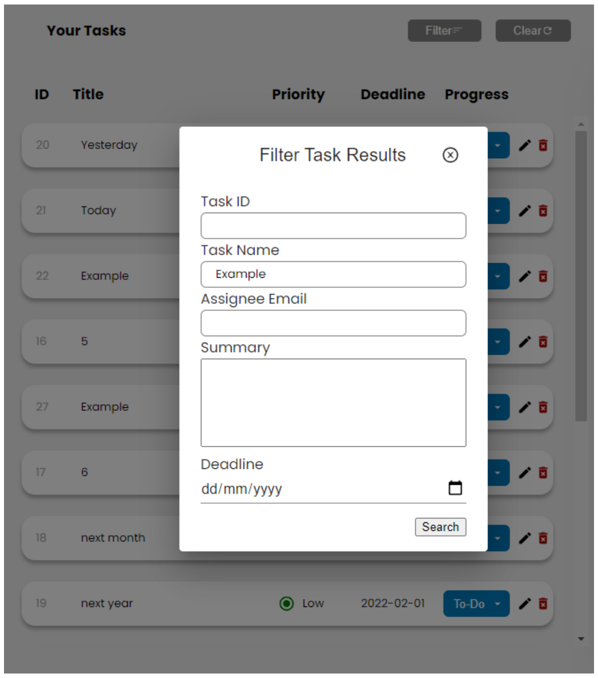
Click on a Task to Expand the Details Here

Overall, these functionalities are designed to address the project objective - **In order to progress the state of a task, a Task Master must be able to update the status of a task they have created or that they have been assigned (valid task states include "Not Started" (start state for new tasks), "In Progress", "Blocked", Completed").**

2.5: Search through Tasks

To allow users to organise and quickly find their desired task, a search feature has been implemented. This has been implemented using a form to search for the desired fields. This is accessed by the filter button on the main page. This will filter the total tasks being shown to correspond with the inputs you give the form. You can then click the clear button to reset the filter results.

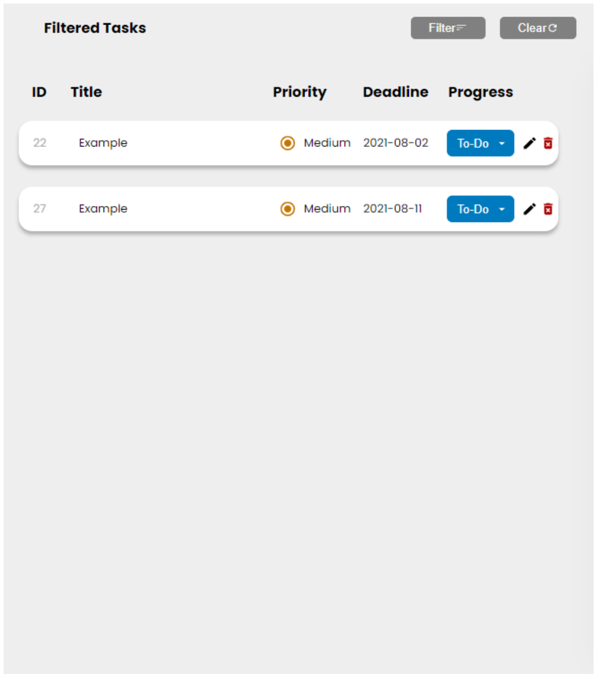
The task filter is additionally implemented on the profiles, however, the assignee email defaults to the profile's email.



ID	Title	Priority	Deadline	Progress
20	Yesterday			
21	Today			
22	Example			
16	5			
27	Example			
17	6			
18	next month			
19	next year	Low	2022-02-01	To-Do

Filter Task Results

Task ID	
Task Name	Example
Assignee Email	
Summary	
Deadline	dd/mm/yyyy

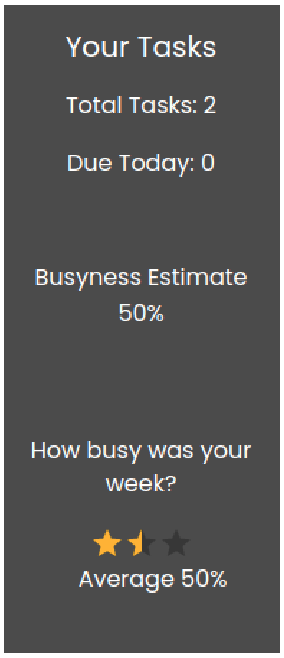


ID	Title	Priority	Deadline	Progress
22	Example	Medium	2021-08-02	To-Do
27	Example	Medium	2021-08-11	To-Do

These functionalities are implemented to satisfy the requirements **- Over time, the number of tasks existing in the system will build up, so it must be possible for a Task Master to search, (through all tasks assigned to themselves or any Task Master they are connected to), by any combination of id, name, description and/or deadline, and view its full details.**

2.6: Task Master Busyness

The busyness estimation will be displayed whenever a user's profile is accessed. It will be on a scale of 6 stages from 0% to 100+% (0%, 25%, 50%, 75%, 100%, 100+%). This is calculated by using the particular week's total tasks, deadlines, tasks in progress, tasks of high priority, and tasks completed. Machine learning is utilised with previous data and the user's feeling of busyness which will then calculate the estimated busyness of the user's week based on those aspects.


Your Tasks

Total Tasks: 2

Due Today: 0

Busyness Estimate
50%

How busy was your week?

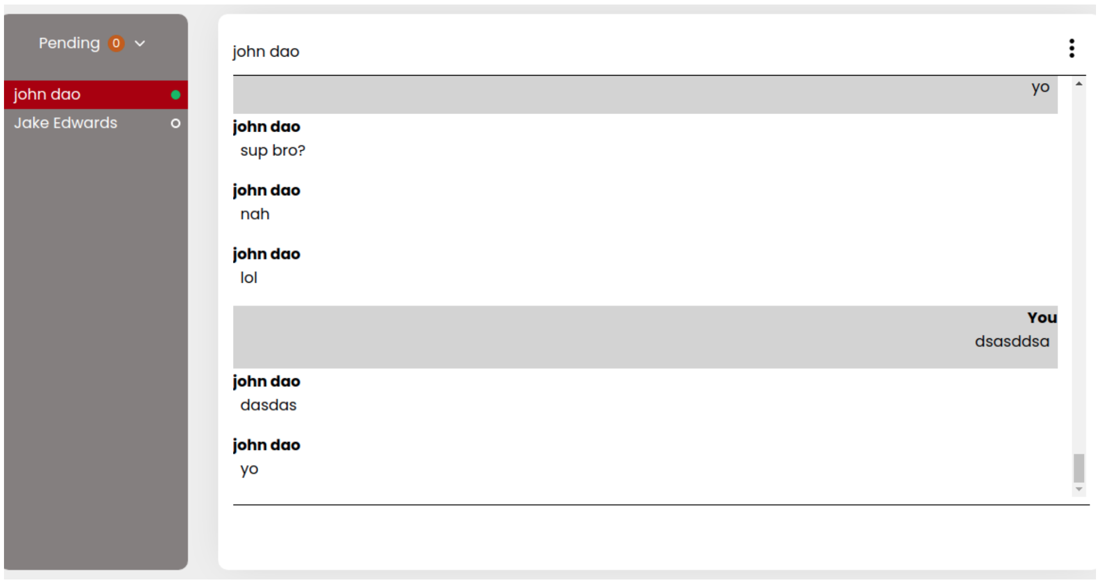
Average 50%

Users rate their busyness through the 3-star input field.

This busyness feature was implemented to satisfy the following objective - **The system must be able to show a Task Master an estimate of how busy each of their connected Task Masters will be over the next week, (0% (min possible) through to 100% (maximum possible), or "100%+ (overloaded)" for any Task Masters it thought were overloaded), where this value can be based on a combination of their assigned tasks, task states, task deadlines, how long similar tasks have taken to complete in the past, and/or any other variables that you wish to use/introduce for the purposes of estimating how busy each Task Master is**

2.7: Messaging (Functional Novelty)

On the main page, users can navigate to an in-built messenger, where they can message their connections in real-time. Connected users will also appear offline/online so users will be indicated if they will be able to receive replies quickly or not.



yo

john dao
sup bro?

john dao
nah

john dao
lol

You
dsasddsa

john dao
dasdas

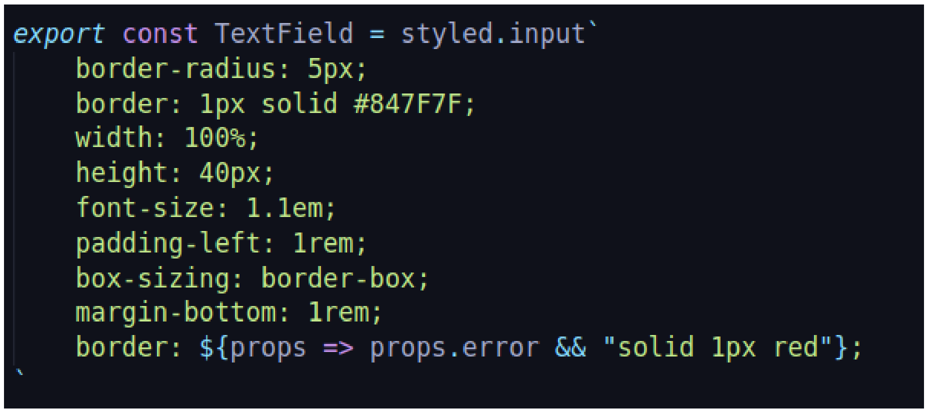
john dao
yo

Messages are displayed with the newest messages shown first. Users will also be shown to be offline/online on the friends list component.

3: Frontend third-party functionalities

3.1: Styled-components

An open-source CSS library used for JavaScript projects, which intuitively follows the React pattern of making components and reusing them. It also allows dynamic styling to be integrated a lot easier, which is common in our web application, hence why it was chosen.
(styled-components, 2021)



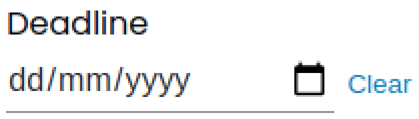
```
export const TextField = styled.input`
    border-radius: 5px;
    border: 1px solid #847F7F;
    width: 100%;
    height: 40px;
    font-size: 1.1em;
    padding-left: 1rem;
    box-sizing: border-box;
    margin-bottom: 1rem;
    border: ${props => props.error && "solid 1px red"};
`
```

An example of a styled-component that uses dynamic styling.

Here, dynamic styling is used to highlight an input box if an error occurs.

3.2: Material UI

A CSS library that provides ready to use components to be integrated. We used this in special cases where it would have been easier to use a pre-built component instead of making it ourselves. A good example is the deadline field in the ‘Create A Task’ form. Since we are using the Community Edition, no payments will be required to the creators.
(Material-UI, 2021)



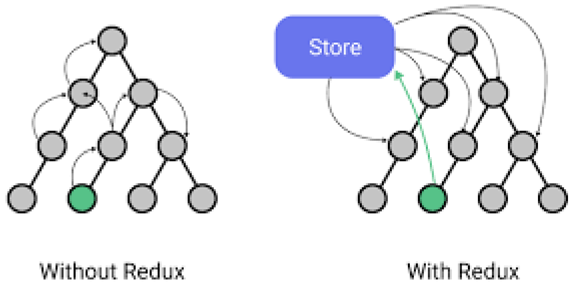
Users can click on the calendar button, which reveals a calendar.

A date can be selected which renders in dd/mm/yy format.

3.3: Redux

A state management library that allows global states to be shared across all components, which is essential for our web application that stores many global states. A good example is the user’s details, which include their id, token, and username, that is used in many of the components of the frontend. This also avoids prop drilling, where arguments are sent down a nested virtual DOM tree, which can create many errors if they are passed down incorrectly. Redux is also open-source, so licencing is not an issue.

(Redux, 2021)



With Redux, the state is retrieved from a global store. Without it, global states must be passed down the virtual DOM.

Image is taken from https://proxify.io/articles/using-redux-with-react

3.4 Socket.io

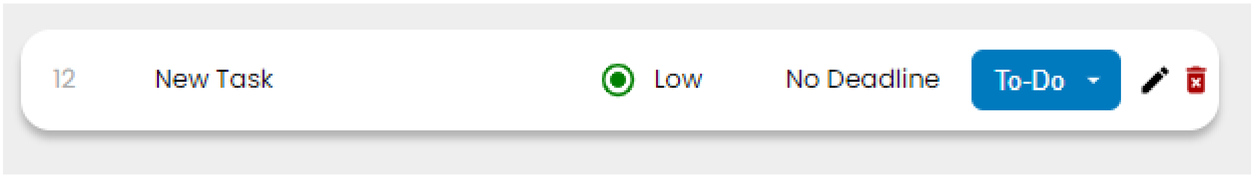
Allows real-time messaging across different users by creating new web sockets for each logged-in user. It is also open-source, so scaling with more users will only cost us extra with hardware upgrades.

(Socket.IO, 2021)

3.5: React-icons

Has a large collection of common images for use with React. This library is also open-source, and simple to install using npm. Some examples of icons in our applications include the refresh button on the clear tab, the tick and the cross for accepting and declining, and the priority, edit and delete button on the tasks.

(React Icons, 2021)



4: Backend third-party functionalities

Although Django allowed for a high level of modularity between functionality and packages that could be utilised, there were few third party functionalities used in the final build including:

- Django-friendship
- Django-apscheduler
- Django-rest-passwordreset

4.1: Django-friendship

Django-friendship is an open-source package that allows the basic implementation of friendship features that includes:

- Following
- Blocking
- Bi-direction friendships

Though the package allows for this functionality to be implemented, Its design was based upon vanilla Django and not Django's rest framework which was utilised in the Task Master project. This meant that for full friendship implementation, endpoint functionality or 'views' had to be purpose-built in order for the frontend team to access its functionality.
(django-friendship, 2021)

4.2: Django-apscheduler

The Django-apscheduler is an open-source Django implementation of python's Apscheduler that allows certain functionalities, tasks and logic to be scheduled for execution later.
Django-apscheduler was selected as it allowed the scheduling of tasks without the need for inter-process communication libraries such as Redis or Docker.

This functionality is utilised mainly in busyness and in updating the busyness prediction model and resetting the user busyness submission. The main function that executes the aforementioned functionality is called every Saturday at 12 AM.
(django-apscheduler, 2021)

4.3: Django-rest-passwordreset

Django-rest-passwordreset is an open-source package that enables the reset of passwords via an email confirmation with a Django rest application. It allows users of Task Master to send a request to reset their account's password and confirm the request through a unique code that is sent to an associated user's email.
(django-rest-passwordreset, 2021)

4.4: Pytorch

Pytorch is an open-source machine learning library. This is used in the creation of the neural network that estimates the busyness of the user based on their tasks.
(PyTorch, 2021)

5: Implementation Challenges

5.1: Real-time Messaging (Socket.io)

Initially, we had planned to poll the servers constantly to retrieve and send messages that would give the illusion of real-time. However, this would have heavily congested the server, even at a small scale, and could potentially show messages out of order. For example, if two users messaged at the same time, you might see on two different sides:

User 1 - I sent the message first

User 2 - No I did!

User 2 - No I did!

User 1 - I sent the message first

which ultimately creates user experience issues. So, we decided to use web sockets, which would allow users to direct messages without concurrency issues.

For its first implementation, we planned to use the Django channels package to implement messaging. It would also make sense to integrate messaging with the backend code as it would centralise components together. However, the package was unintuitive when it came to building the private messaging service, as it required many workarounds to get the foundations working. Also, the feature of showing users being online/offline also seemed too big of a challenge, which almost scrapped the idea of integrating real-time messaging.

Instead, we experimented with Socket.io, which would not be integrated with the Django server and instead act as its own standalone Node.js server. While it seemed sceptical to implement it this way, since it decentralises code, even more, it turned out to be the more optimal solution.

The Socket.io server was able to create socket ids that were uniquely assigned to an individual web browser. These socket ids allowed messages to be redirected to the correct users, which showed in real-time as well. Socket.io also allowed the broadcasting of currently online users, which was how we were able to implement users being online/offline.

5.2: Busyness estimates and the acquisition of data (novelty)

Although the specifications of the busyness estimate specified any type of implementation, we decided to incorporate machine learning into our busyness estimate. This included using existing user task data and form responses to predict the busyness of an individual. However, due to this increase in novelty complexity, there were a variety of challenges that had to be overcome in order for this to be successfully implemented.

The lack of realistic data reduced the capability and accuracy of the predictions that were made from the model. As the generation of the model depended upon large amounts of data to provide a higher level of accuracy. However, due to the unavailability of accurate task data within our project's ecosystem, the random generation of data was utilised in training the machine learning model. This meant that the generated busyness value was not accurate and resulted in irregular behaviour such as busyness being unresponsive to large changes in user workload.

5.3: LocalStorage vs Data Retrieval from Backend (Frontend)

In web applications, localStorage is an object in web browsers that allows data to be cached. In many scenarios, we had to consider storing data in localStorage for faster retrieval or retrieving data by polling the server. A good scenario is when we need to retrieve the user's username. We could retrieve the data from the backend every time we need it, or retrieve it once and store it, so we don't create redundant traffic for the server.

However, the issue mainly arises when deciding if components should use localStorage or not. A good example would be when a user has been removed as a friend by another user. In this scenario, you would have to call the backend, since changes will not be detected if they were in localStorage. For example:

Friends List with localStorage

Friend 1 (Deleted but still shown)

Friend 2

Friends List without localStorage

~~Friend 1~~ (Deleted)

Friend 2

Therefore, to optimise the speed of the web application, the use of localStorage versus retrieval from the backend always needed to be considered. This was not easy to determine from the initial implementation, and bugs were usually only found when experimenting.

5.4: Django's learning curve

Though the implementation of Django and Django's rest framework allows for quick implementation, the previous knowledge required for developers to use the technology was high. This means that developers who are not as well-rehearsed in Django took much longer to begin contributing and implementing features than the average technology. Even with thorough inductions, the learning curve of Django prevented members in the backend team from contributing to their full potential quicker ultimately, slowing down the release of backend features.

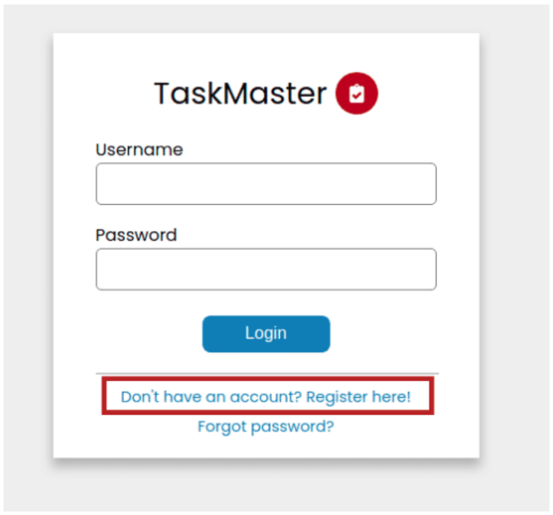
To solve this, more integration times and group programming sessions enabled team members to better grasp the capabilities of Django. Alongside this, a separate project was created that was better in tune with the skillset of members who weren’t well-rehearsed in Django and allowed these members to better utilise their skills in contributing to the Task Master project.

6: User Documentation and Manual

6.1: Authentication

6.1.1: User Registration

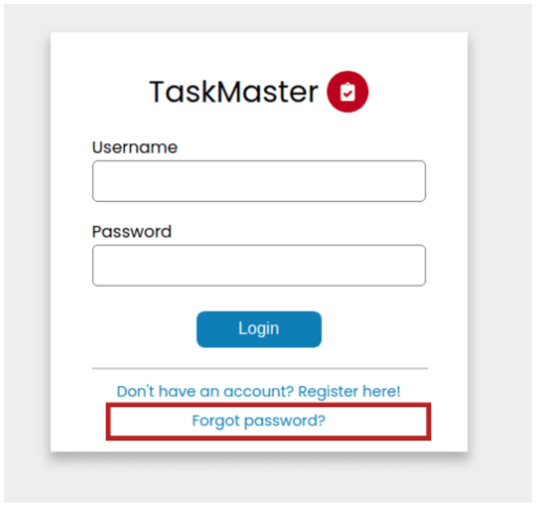
1. On the login page, click the first link below the Login button.



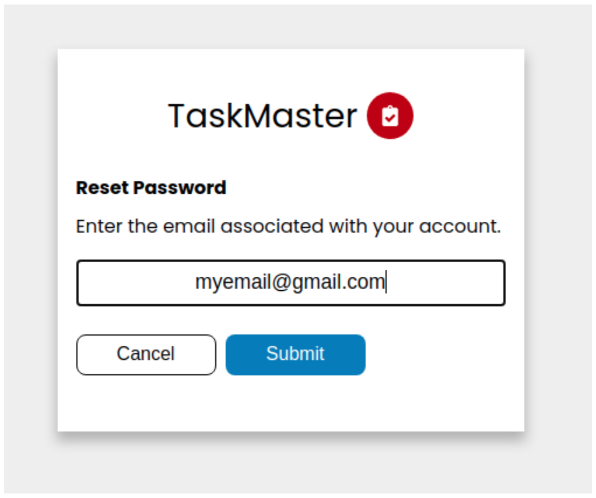
2. You will be redirected to a registration form. Enter the appropriate details to create an account.

6.1.2: Resetting User Password

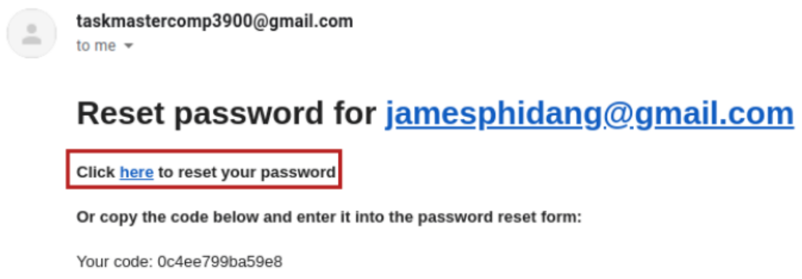
1. On the login page, click the second link below the login button.



2. You will then be asked to enter the email you registered with.



3. If the email is connected to an account, a reset password email will be sent. Check your email inbox and click on the link shown below.



taskmastercomp3900@gmail.com
to me

Reset password for jamesphidang@gmail.com

Click here to reset your password

Or copy the code below and enter it into the password reset form:

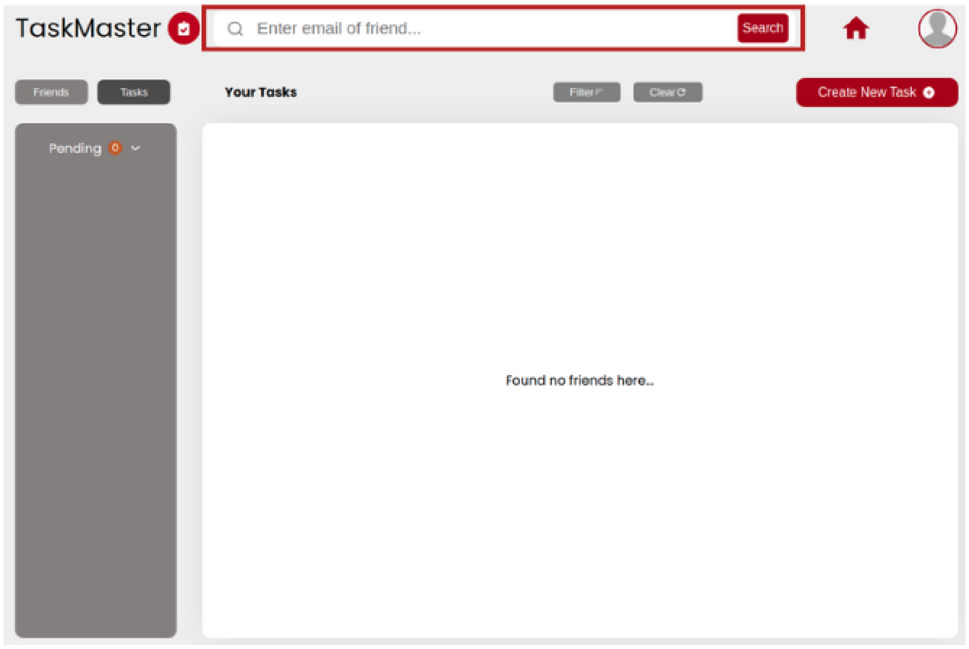
Your code: 0c4ee799ba59e8

4. You will then be redirected to the password reset page, where you can enter your new password.

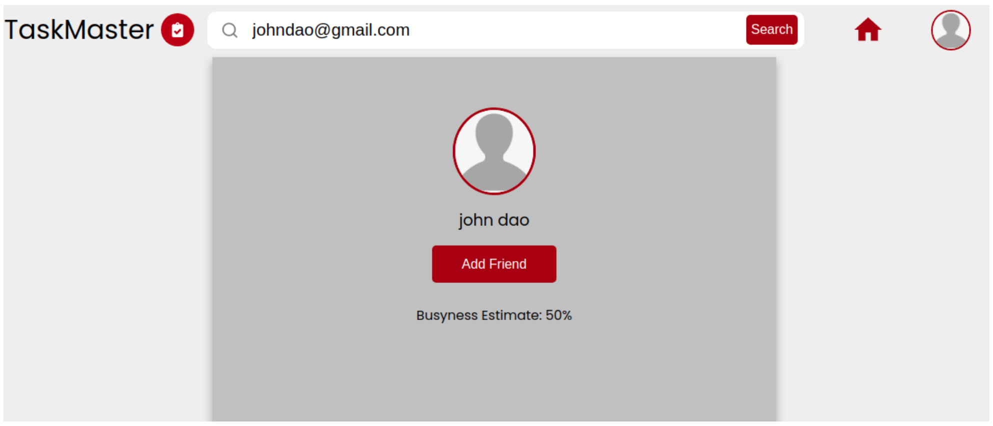
6.2: Connections

6.2.1: Adding a friend

1. On the main page, enter your friend's email to search for their profile

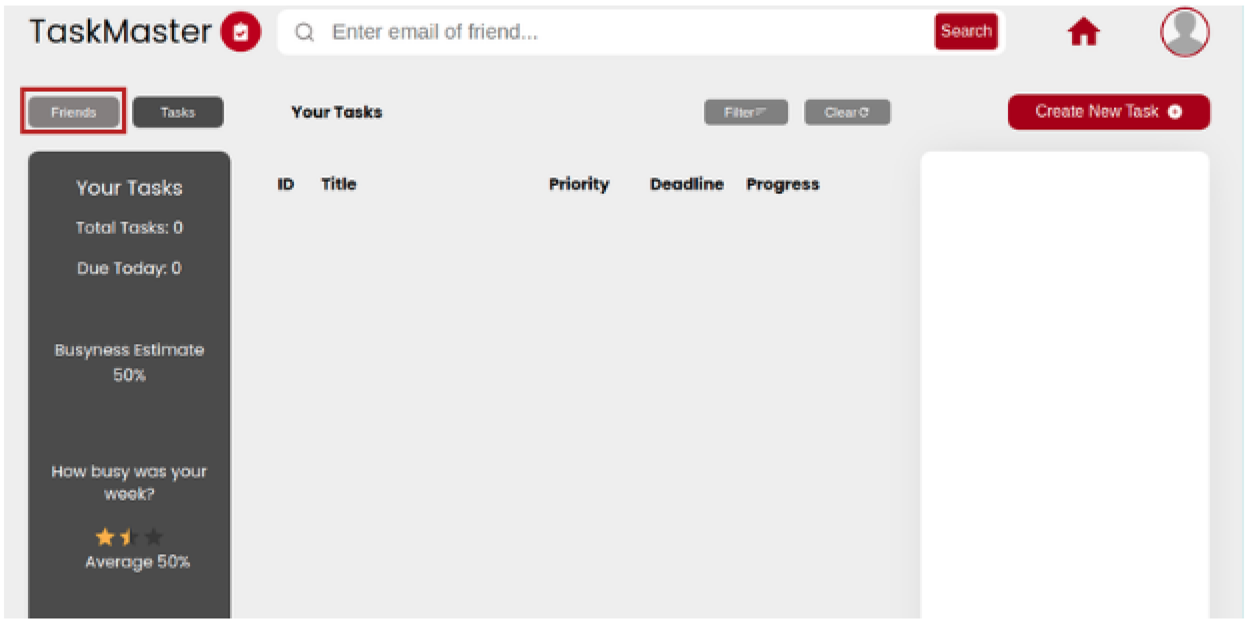


2. You will then be redirected to their profile page, where you'll be able to click the add friend button

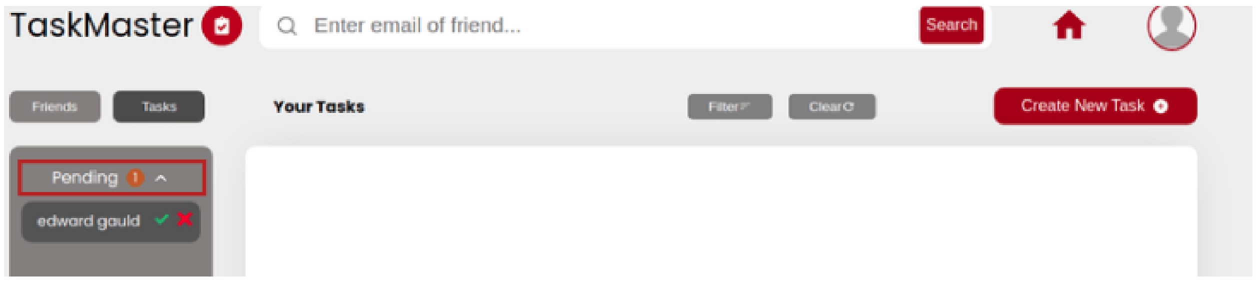


6.2.2: Accepting a friend request

1. On the main page, click the friends button on the left-hand side

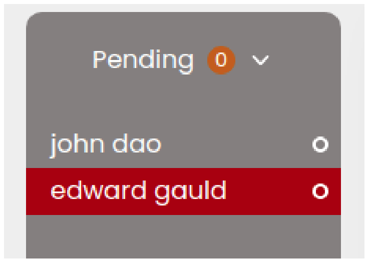


2. Click on the pending requests button to approve or decline a friend request

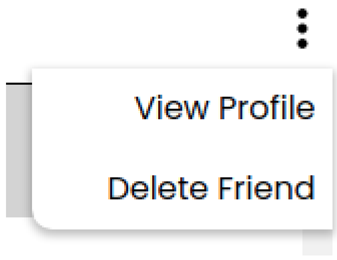


6.2.3: Deleting A Friend

1. Select the friend you want to delete on your friends list

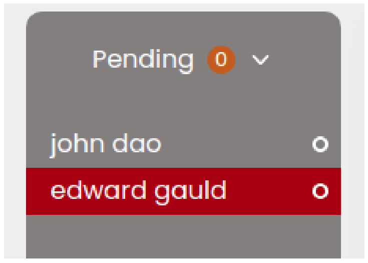


2. On the messaging container, there is a three-dot menu on the top. Click on it to show the delete friend button.

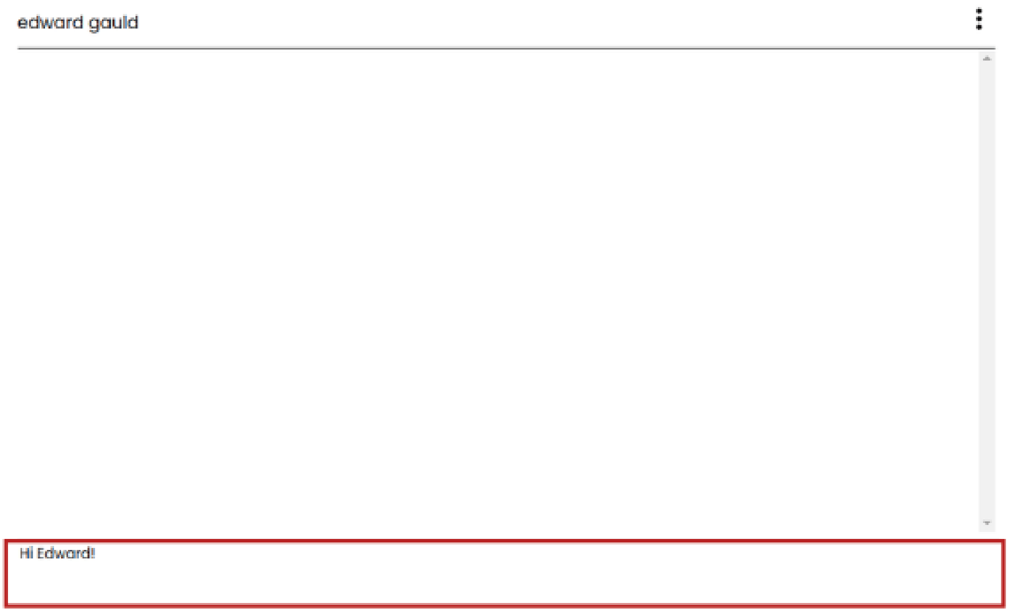


6.3: Messaging

1. Select the friend on the friends list that you want to message



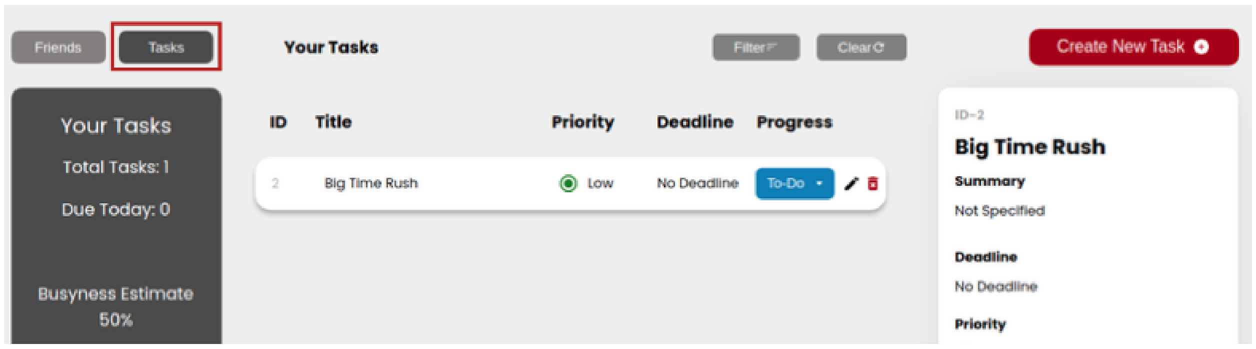
2. Type a message in the message box and press Enter to send



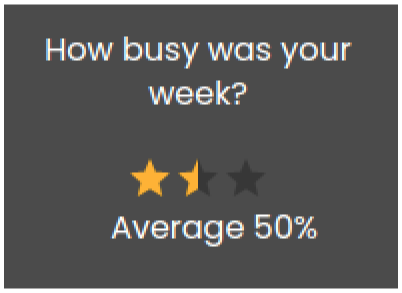
6.4: Busyness

6.4.1: Submitting your busyness

1. To submit your busyness, click on the Tasks button on the main page



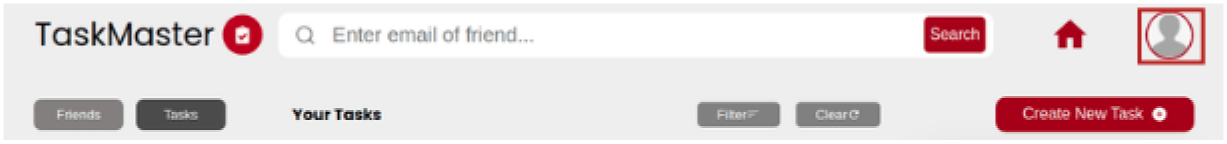
2. On the left-hand side, you will see the busyness component, where you submit your busyness



6.5: Profiles

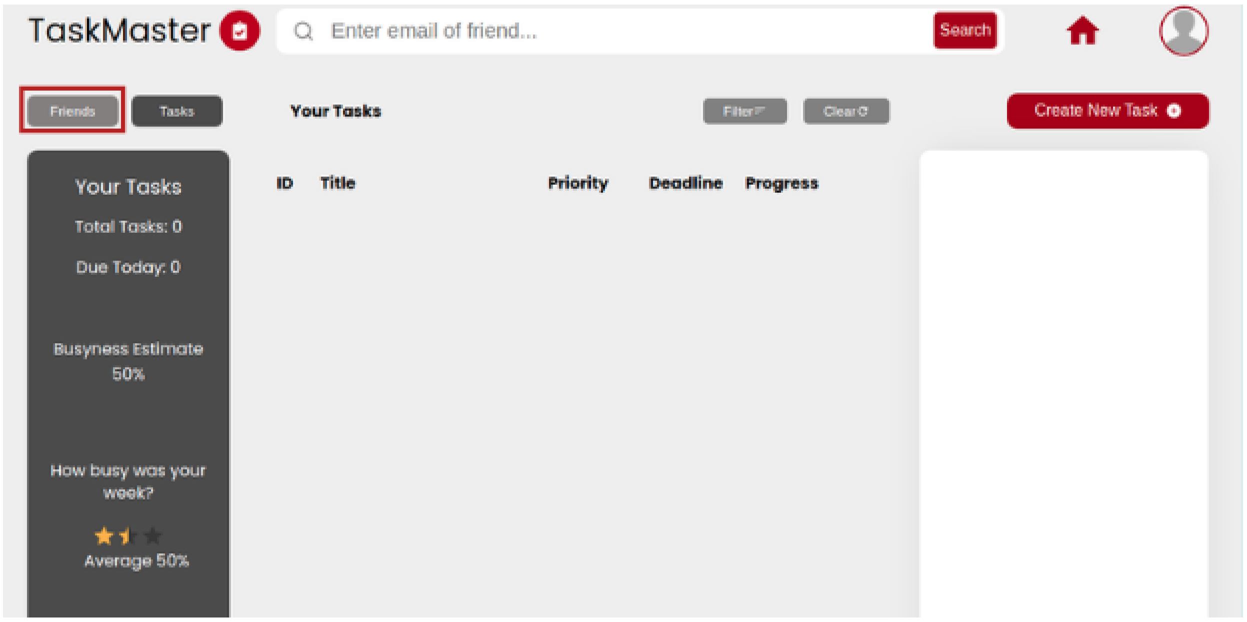
6.5.1: Viewing your own profile

1. To view your profile, click on the profile icon on the header. A view profile button will show in the dropdown menu

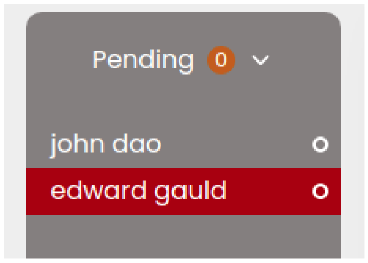


6.5.2: Viewing friend profiles

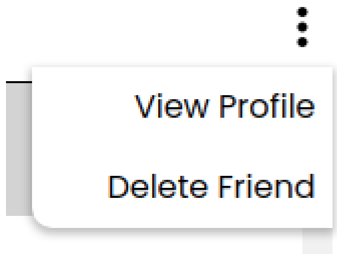
1. To view your friend's profile, click on the friends button on the main page



2. Select the friend on the friends list



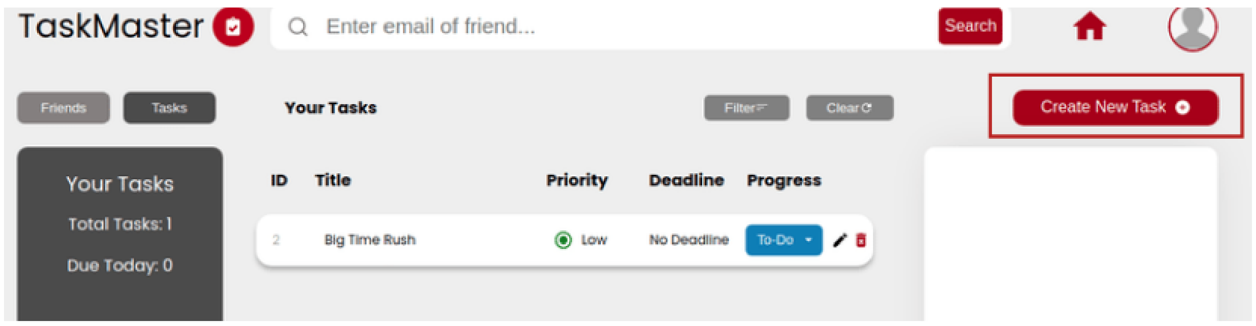
3. On the messaging container, there is a three-dot menu on the top. Click on it to show the view profile button.



6.6: Tasks

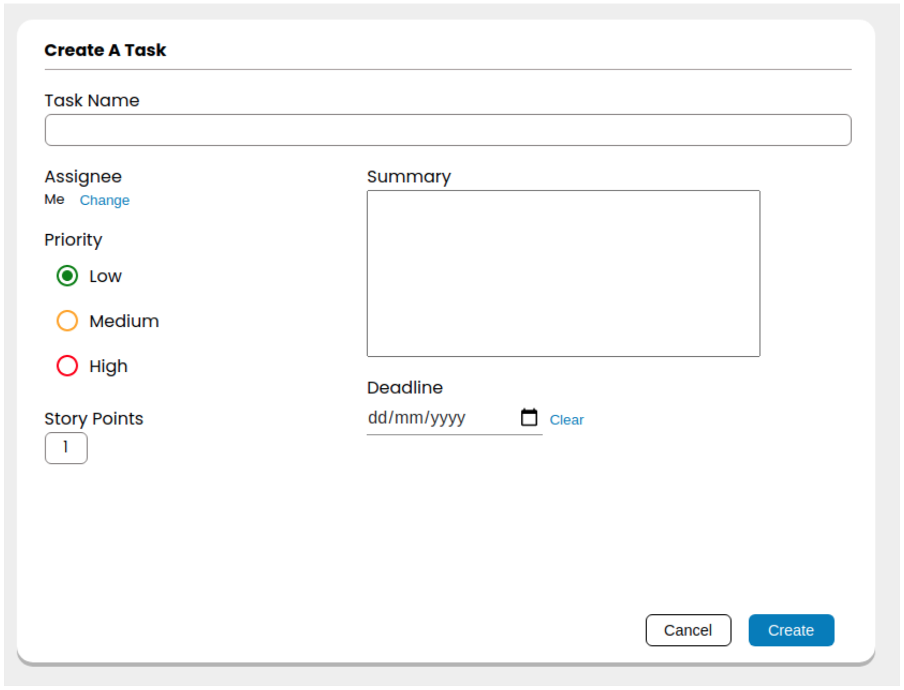
6.6.1: Creating A Task

1. Click on the Create New Task button on the main page|



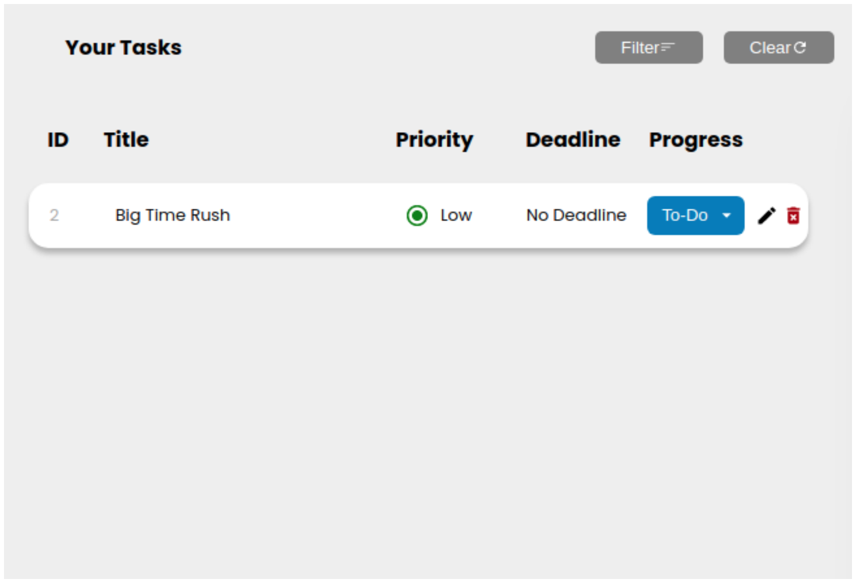
ID	Title	Priority	Deadline	Progress
2	Big Time Rush	Low	No Deadline	To-Do

2. You will then be redirected to the Create Task page where you will be able to assign it to your friends, add a deadline, give it a priority and add story points



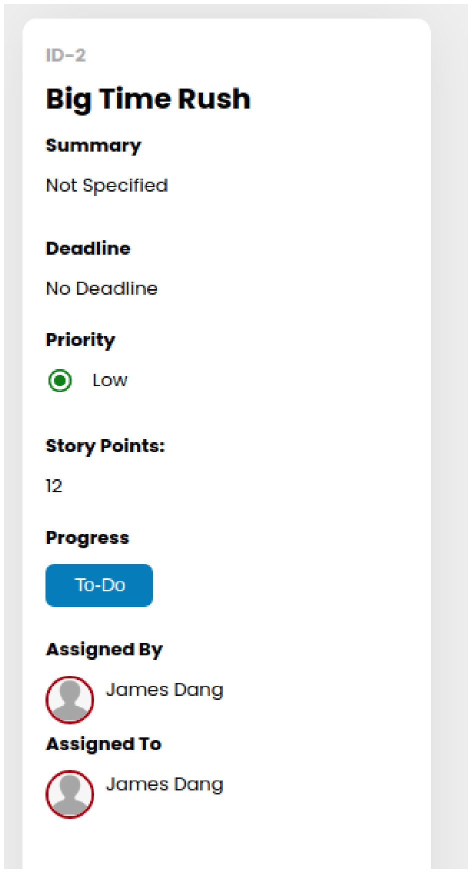
6.6.2: View Task Details

1. On the main page, click the task you would like to view the details of.



ID	Title	Priority	Deadline	Progress
2	Big Time Rush	Low	No Deadline	To-Do

2. On the right of the main page, you will be to view the task details



ID-2

Big Time Rush

Summary

Not Specified

Deadline

No Deadline

Priority

Low

Story Points:

12

Progress

To-Do

Assigned By

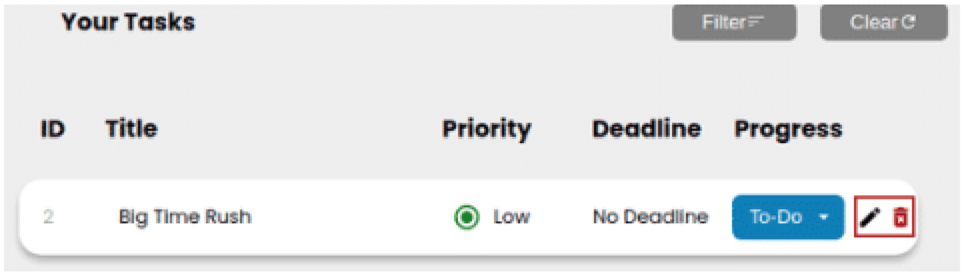
James Dang

Assigned To

James Dang

6.6.3: Editing and Deleting a Task

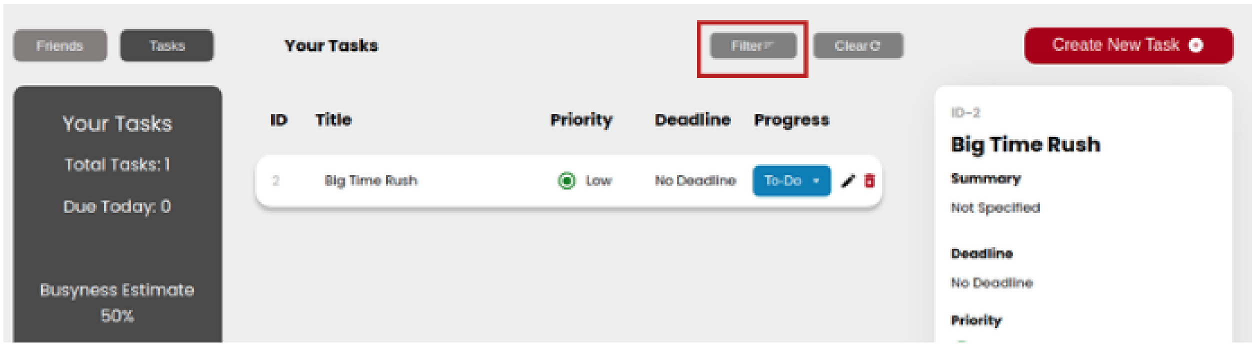
1. To edit and delete a task, click on the icons on the task component



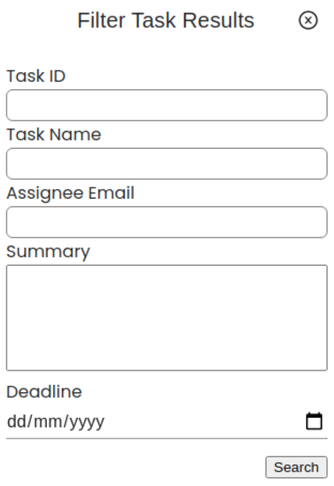
ID	Title	Priority	Deadline	Progress
2	Big Time Rush	Low	No Deadline	To-Do

6.6.4: Filtering Tasks

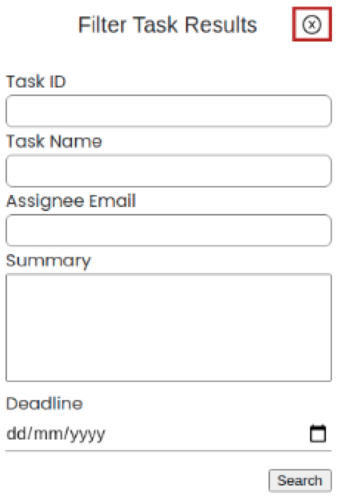
1. To filter the tasks, click on the filter button at the top of the tasks list



2. A filter form will pop up where you can enter the filters you want



3. If you do not want to enter any filters, click the X button on the top right


Filter Task Results

Task ID

Task Name

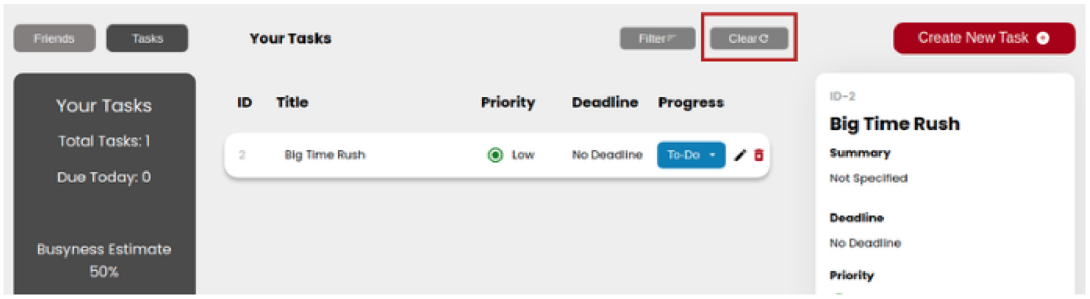
Assignee Email

Summary

Deadline

dd/mm/yyyy

4. To remove any existing filters, click the ‘clear’ button at the top of the tasks



Your Tasks

Your Tasks

Total Tasks: 1

Due Today: 0

Busyness Estimate 50%

ID	Title	Priority	Deadline	Progress
2	Big Time Rush	Low	No Deadline	To-Do

ID-2

Big Time Rush

Summary

Not Specified

Deadline

No Deadline

Priority

7: How to run the project

We chose to run the project on the Lubuntu 20.4.1 LTS virtual machine image stated in the assessment guidelines. We used Virtual Box 6.1.22 to run our virtual machine.

The virtual image can be downloaded here: https://rebrand.ly/o1fy80n(Linux VM Images, 2021)
Once you have set this up on virtualbox, login to the machine and download the project code repo.
In the terminal (QTerminal), navigate the main directory of the repo, and type *ls*.
The output should include the following 5 scripts:
install
runall
runbackend
runfrontend
resetdb

First, run
sudo ./install
This will install all the dependencies required to run the app. Please wait for this to complete.

Once the dependencies are installed, you can run the app.
To run the entire app, run
./runall
This will create 3 new terminals running the frontend and backend servers.
Firefox should open. If a page comes up saying "Warning: Potential Security Risk Ahead", click on the "Advanced" button, then click "Accept the Risk and Continue."
You should then see the login page of our app, and can now use it how you please.
If you find that your configuration of the virtual machine screen size is too small, you will need to widen the screen to use the app effectively.

If you only want to run the frontend, run
./runfrontend
2 terminals are created, running the frontend servers.

If you only want to run the backend, run
./runbackend
A terminal is created running the backend server.

If you want to reset the database, run
./resetdb

References

django-apscheduler. PyPI. (2021). Retrieved 1 August 2021, from https://pypi.org/project/django-apscheduler/.

Download Linux VM Images from SourceForge.net. Rebrand.ly. (2021). Retrieved 2 August 2021, from https://rebrand.ly/o1fy80n.

GitHub - anexia-it/django-rest-passwordreset: An extension of django rest framework, providing a configurable password reset strategy. GitHub. (2021). Retrieved 1 August 2021, from https://github.com/anexia-it/django-rest-passwordreset.

GitHub - revsys/django-friendship: Django app to manage following and bi-directional friendships. GitHub. (2021). Retrieved 1 August 2021, from https://github.com/revsys/django-friendship.

Material-UI: A popular React UI framework. Next.material-ui.com. (2021). Retrieved 1 August 2021, from https://next.material-ui.com/.

PyTorch. Pytorch.org. (2021). Retrieved 1 August 2021, from https://pytorch.org/.

React Icons. React-icons.github.io. (2021). Retrieved 1 August 2021, from https://react-icons.github.io/react-icons/.

Redux Toolkit | Redux Toolkit. Redux-toolkit.js.org. (2021). Retrieved 1 August 2021, from https://redux-toolkit.js.org/.

Socket.IO. Socket.IO. (2021). Retrieved 1 August 2021, from https://socket.io/.

styled-components. styled-components. (2021). Retrieved 1 August 2021, from https://styled-components.com/.