

Greenthumb: Work Operations and Scheduling Platform

T3A2 Full Stack Application

By Harry Manton, Connor Ross and Adam Hutt

Purpose

Greenthumb is local landscaping company that is struggling to keep up with its popularity and in-turn are hiring a lot of new staff and have acquired a lot of new business. The office manager is struggling logistically with the amount of jobs due as they are working off mobile-phones and physical conversation to gather and relay information. Our team aims to solve this issue with the Greenthumb online work operations and scheduling platform. It aims to make communication of important job/schedule information more open and accessible to all employees and limit person to person communication. In doing this, the app will also be able to store important job information, employees work hours and other crucial data vital for other parts of the business like accounts.

Target Audience

Our target audience for the application is the Greenthumb Landscaping business and is catered to their requirements. However, tracking, scheduling and employee management tasks are vital for many business of a similar size and could easily cater for other businesses needs also.



Original User Stories - Review

- As a **manager/owner**, I want to be able to create new jobs within the app, including specifying the job title, address, any special requests, and assigning employees to the job.
- As a **manager/owner**, I want to view a list of all jobs, including their status (e.g., pending, in progress, completed), so that I can track the progress of each job and manage the workload efficiently.
- As a **manager/owner**, I want to review completed jobs, including any notes left by employees and customer feedback, to ensure that the work meets quality standards and address any issues that arise.

In regards to the first , user story. We successfully implemented a create job features with the ability to specify job title, and address. We decided against the special requests as we felt the important Job information was enough for the employees needs.

For the second user story, A status on the the job and its relation to the date , is within our app. So any jobs past the current day would be inactive. And all jobs are for the managers are available to see from the calendar.

For the the third user story, we opted against using customer input , due to time constraints, having another authority level to control would have taken more time which we didn't have the project.

- As an **employee**, I want to see a list of upcoming jobs assigned to me, along with relevant details such as the job title, address, and any special instructions, so that I can prepare for my workday.
- As an **employee**, I want to clock in and out for each work order directly within the app, to accurately track my hours worked and ensure that I am paid correctly.
- As an **employee**, I want to leave notes on each job, such as any issues encountered, additional equipment used, or positive feedback from the customer, to provide detailed information to the manager/owner and improve communication within the team.

The Greenthumb work platform allows for tailored access depending on the user and the jobs they are assigned to. Employees are able to log-in and see the type of work (Mowing, Decorating, Cleaning) and the tools they will require for that day of work , without direct communication with the management staff.

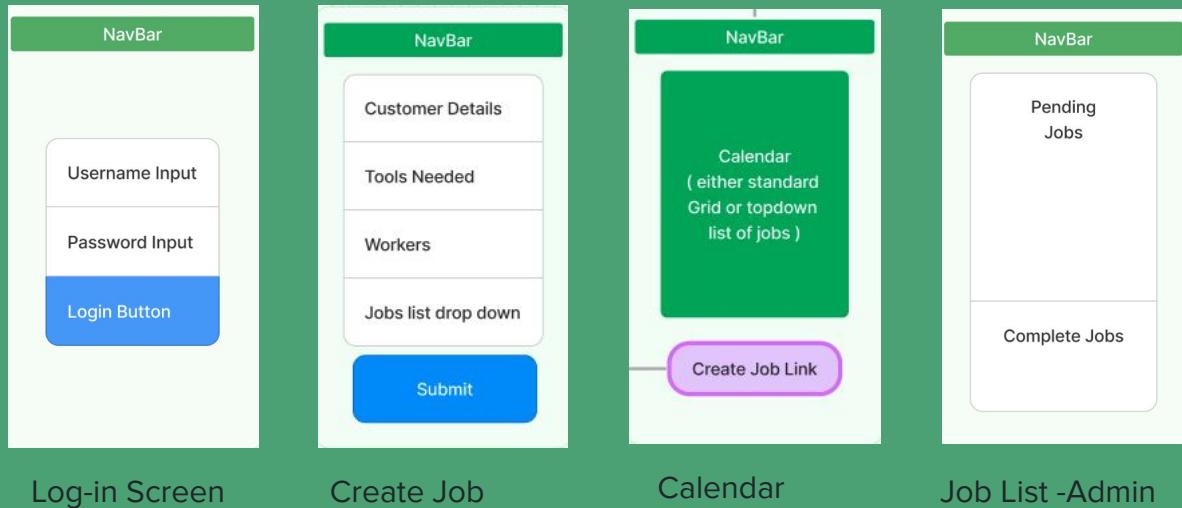
We to have the ability to store and manage employee hours , but it requires some more functionality to be active for employees. Timesheets are available to view as they are hard-coded in the database, and once the clock-in/clock-out mechanism is implemented , They will be viewable for the employee and manager.

We decided to make employee functionality read job only, this helps solve the scheduling problem Greenthumb are experiencing and limit dialogue about job information going back and forth between management and the staff.

- As a **customer**, I want to create an account within the app, so that I can easily submit requests for landscaping services and track the status of my requests.
- As a **customer**, I want to submit a request for a job, including providing details such as the job title, address, any special instructions, and preferred dates/times for the service, to ensure that my needs are clearly communicated to the landscaping company.
- As a **customer**, I want to receive notifications and updates on the status of my job request, including when it has been assigned to an employee and when the work is completed, to stay informed throughout the process.

As I mentioned previously, customer authority and features were out of scope , handling and splitting authorities and features between Managers and Employees proved to to be sufficient for the scope of the project. Ideally ,if continuing on with the project customer authority and a communication system could really streamline and improve operations and booking of jobs.


Part-A Wireframes - Mobile



Here are some examples of our planned Wireframes. We have a a log-in Screen, Create Job , Calendar and a list of Jobs that the Admin authorised accounts.

- The login screen is a basic login screen with inputs for Username and Password and the log in button submits the details.
- Customer Details allows the admin user to Enter Customer Details , Tools Needed, Add Workers and adds Tasks/Jobs for this job.
- The calendar marks jobs on the day , depending on the users involvement in the Job
- The Jobs list shows all the jobs created and sections into active (pending) and complete.

Part-B Production Website - Mobile




Green Thumb Landscaping

Username:

Password:

Login

Log-in Screen



New Job

Customers Name:

Customers Mobile:

Job Address:

Tasks:

Select a task

Tools Required:

General Items (wheelbarrow, shovel, rake, broom, grass bags)


Select Employee:

Select an employee

Mike
Sally
Frank

Add Employees

Create Job



Calendar

February 2024


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

Jobs for Wed Feb 28 2024

Jean Waratah - 5 day job

Click In

Calendar



Customer Name: Jake Rose

Workers on site:

David
Mike

Edit Delete

Customer Name: Harold Daffodil

Workers on site:

David
Frank

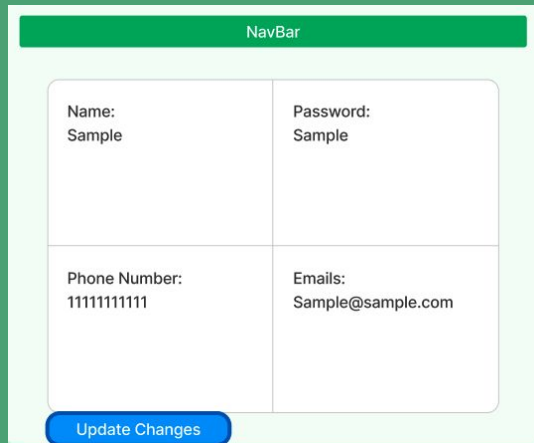
Edit Delete

Job List -Admin

Here we have the same page examples of our planned Wireframes. log-in Screen, Create Job , Calendar and a list of Jobs that the Admin authorised accounts.

- The log-in screen was implemented as per the wireframe description and added the App name and logo.
- Customer Details has been split into 3 sections, Customers Name, Customers Mobile and Job Address (which is a shortened google URL map thats clickable to the user), which will take inputs from the user. Tasks is a hard-coded list of common tasks that the business often does and jobs are categorized into these tasks. Once one of these tasks is selected the required tools will auto generate to the tool required + general everyday items used in all the Landscaping tasks. Also the ability to add and remove users who are not admin into the job and finally submitting the job.
- The calendar marks jobs on the day as planned and allows access to the job via a link generated from selecting on a day marked with a job.
- The Jobs list shows all the jobs created but alternative to the wireframe there is not a reference of completed jobs/active jobs. All Jobs are in the list and the completed jobs are not changed but using logic, any jobs in the past from the current date are not active. From this list they have the added ability to edit/delete the job.

Part-A Wireframes - Tablet

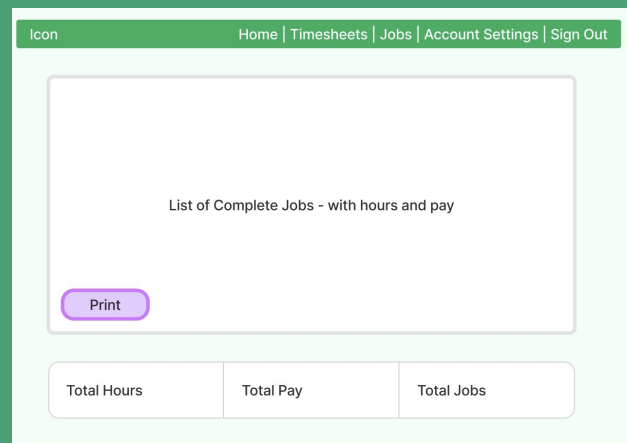


NavBar

Name: Sample	Password: Sample
Phone Number: 1111111111	Emails: Sample@sample.com

Update Changes

Acc Settings



Icon Home | Timesheets | Jobs | Account Settings | Sign Out

List of Complete Jobs - with hours and pay

Print

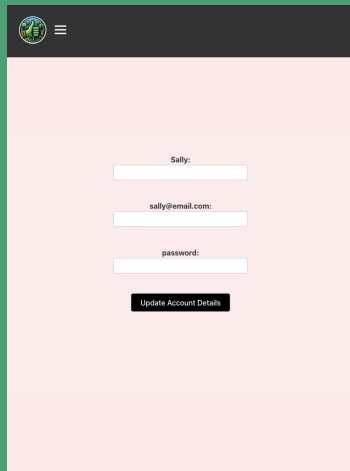
Total Hours	Total Pay	Total Jobs
-------------	-----------	------------

Timesheet

Here are some examples of our planned Wireframes. We have a the Acc setting page , and the timesheet page.

- The Acc settings page allows the user change their name / email / password stored into the database. The current field is marked as a header above the inbox , except for the password as this is sensitive information.
- The timesheet page shows a list of the hours worked, calculated underneath and the ability to print this sheet.

Part-B Production Website - Tablet



The Acc Settings page on a tablet features a dark header with a logo and a hamburger menu. The main content area is light pink and contains three input fields labeled 'Sally:', 'sally@email.com:', and 'password:'. Below these fields is a dark button labeled 'Update Account Details'.

Acc Settings



The Timesheet page on a tablet features a dark header with a logo and a hamburger menu. The main content area is light pink and contains a table with the following data:

Date	Hours	Rate	Earnings
2024-02-16	6	\$25.99	\$155.94
2024-02-24	8	\$29.88	\$239.04

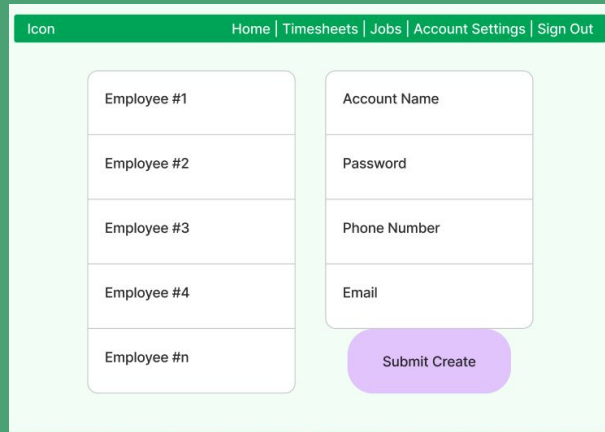
Below the table is a green button labeled 'Print'.

Timesheet

Here are have the same examples from the previous page from the App, the Acc setting page , and the timesheet page.

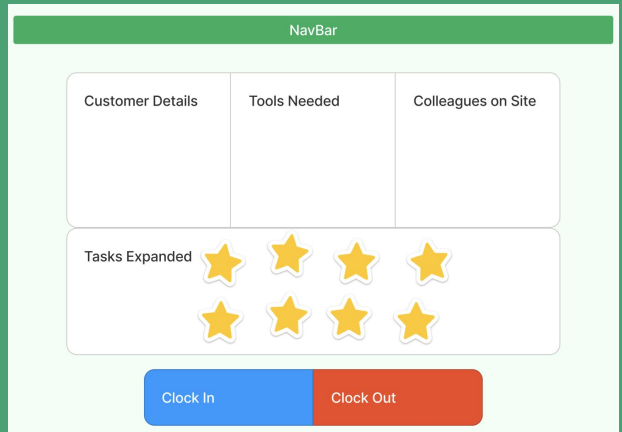
- The Acc settings page allows the user change their name / email / password stored into the database. The current field is marked as a header above the inbox , except for the password as this is sensitive information. It has been displayed slightly different and does not involve the employees mobile number.
- The timesheet page shows a list of the hours worked, but does not calculate/ total the hours , due to time constraints on finishing the project.

Part-A Wireframes - Desktop



This wireframe shows a desktop view for user account creation. It features a green navigation bar at the top with links: 'Icon', 'Home | Timesheets | Jobs | Account Settings | Sign Out'. The main content area is divided into two columns. The left column contains a list of employee entries, each with a label 'Employee #1' through 'Employee #n'. The right column contains a form with fields for 'Account Name', 'Password', 'Phone Number', and 'Email'. Below the form is a purple 'Submit Create' button.

User Account Creation



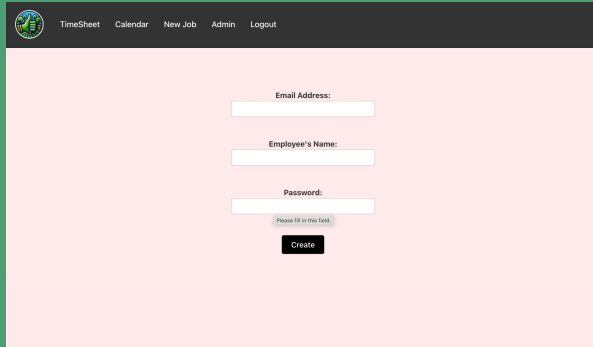
This wireframe shows a desktop view for a single job view. It features a green navigation bar at the top labeled 'NavBar'. The main content area is divided into three columns: 'Customer Details', 'Tools Needed', and 'Colleagues on Site'. Below these columns is a section labeled 'Tasks Expanded' with eight yellow star icons arranged in two rows of four. At the bottom are two buttons: a blue 'Clock In' button and a red 'Clock Out' button.

Single Job View

Here are some examples of our planned Wireframes. We have a the User Account creation , and the Single Job View

- The User Account page on the wireframe displays a list of the Employees and the ability to add and create on the same page.
- The View Job page is reading only, so users can read job all the job information as well as clock-in/clockout

Part-B Production Website - Desktop



TimeSheet Calendar New Job Admin Logout

Email Address:

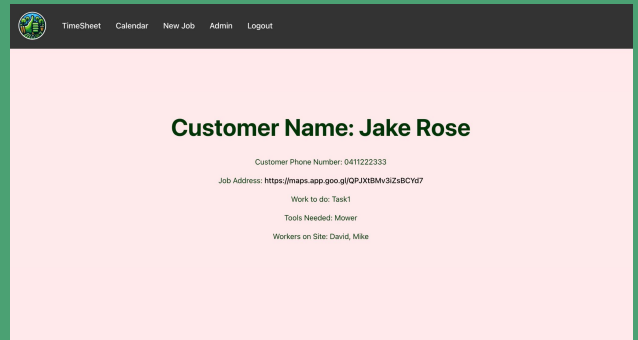
Employee's Name:

Password:

Please fill in this field.

Create

Create User



TimeSheet Calendar New Job Admin Logout

Customer Name: Jake Rose

Customer Phone Number: 041222333

Job Address: <https://maps.app.goo.gl/QPjXlBMv3ZsBCYd7>

Work to do: Task1

Tools Needed: Mower

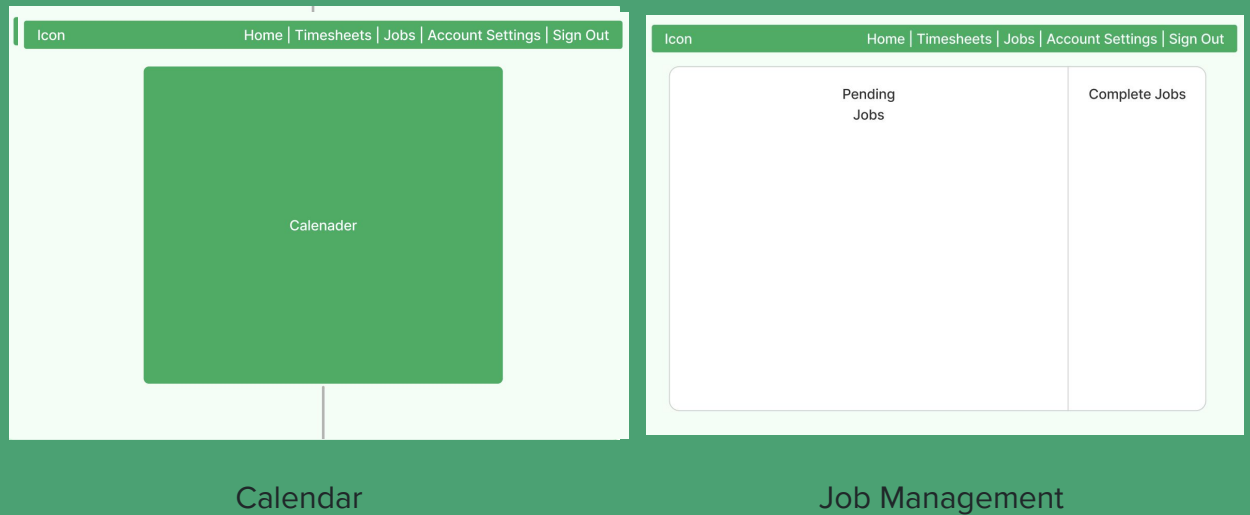
Workers on Site: David, Mike

Single Job Page

Here are some examples of our planned Wireframes. We have a the User Account creation , and the Single Job View

- The User Account in the App does not display the a list of the users information, We thought better to keep this separate and maintain focus on the changing details.
- The View Job page is reading only, and as you can see, the only difference being the the clock-in/clock out functionality has been moved. This functionality has been moved to the calendar as it is easier access for the employee.

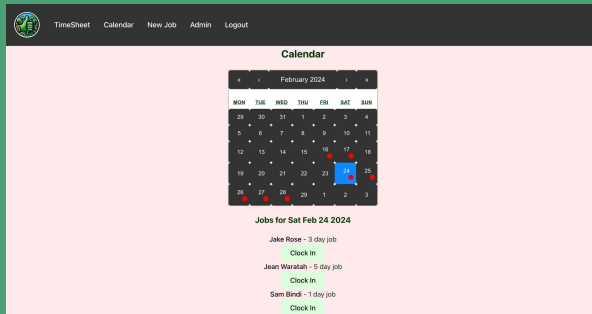
Part-A Wireframes - Desktop



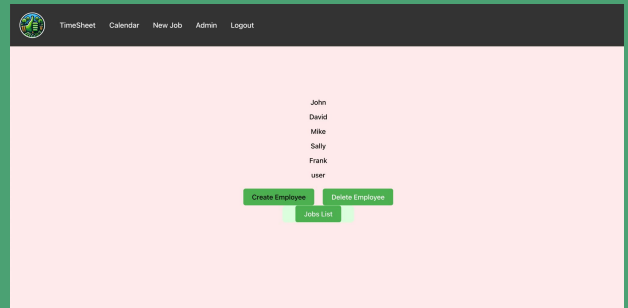
Here are some examples of our planned Wireframes. We have the Calendar and the Login Page viewed again in Desktop Format

- The calendar marks jobs on the day, depending on the users involvement in the Job
- The Jobs list shows all the jobs created and sections into active (pending) and complete.

Part-B Production Website - Desktop



Calendar



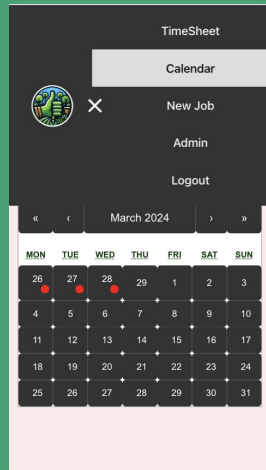
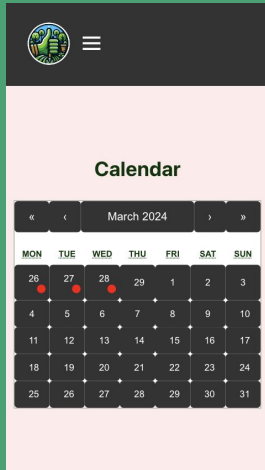
Admin

Here are some examples of our Production App. We have a the Calendar ,and the Login Page viewed again in Desktop Format

- As you can this is the list can hold many jobs per day for the Admins use and still handle clock-in/clock out information.
- The Job Management screen is accessed from the Admin screen where the user can Manage the employee list by creating a new one or deleting from the list. Editing is done by clicking into the users name on the list . From this screen the Job List button access the ability to edit or delete jobs as displayed on previous slides in Mobile.

Code Components & Features

Hamburger Menu - Mobile and Tablet View



```
const Navbar = ({ onLogout }) => {
  const [showMenu, setShowMenu] = useState(false);

  const toggleMenu = () => {
    setShowMenu(!showMenu);
  };

  return (
    <nav className="topnav">
      <img src={Logo} alt="Logo" className="logo" />
      <div className={showMenu ? 'menu-icon' : ''} onClick={toggleMenu}>
        <div className="bar"></div>
        <div className="bar"></div>
        <div className="bar"></div>
      </div>
      <div className={showMenu ? 'links' : ''}>
        <Link to="/TimeSheet">TimeSheet</Link>
        <Link to="/Calendar">Calendar</Link>
        <Link to="/NewJob">New Job</Link>
        <Link to="/Admin">Admin</Link>
        <Link to="/" onClick={onLogout}>Logout</Link>
      </div>
    </nav>
  );
};

export default Navbar;
```

As you can see on the left we have images of our hamburger menu inactive and active in both photos. The menu once active displays all options for our user to access. And clicking on the cross returns it to its regular resting state. The code on the right is the functionality behind these and is used in a react component called Navbar and utilises useState to control when the user clicks to activate or deactivate the user's action in the menu as a state. In addition we use the Link component from the react-router-dom package to establish a link to the other pages and navigate through out.

Calendar Component

```
const tileContent = ({ date }) => {
  // formats calendar date as 'YYYY-MM-DD' to match with DB date format
  const formattedDate = `${date.getFullYear()}-${String(date.getMonth() + 1).padStart(2, '0')}-${String(date.getDate().padStart(2, '0'))}`;

  // Check if any job date matches the formatted date
  const hasJobOnDate = jobs.some(job => job.dates.includes(formattedDate));

  // Return red dot if there's a job on the date
  return hasJobOnDate && <div style={{ backgroundColor: 'red', borderRadius: '50%', height: '0.75rem', width: '0.75rem', marginLeft: '0.75rem' }}></div>;
};

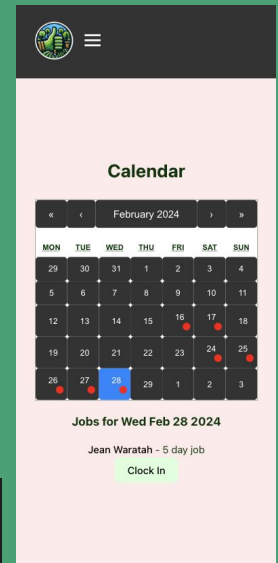
const handleClickDay = (date) => {
  console.log("Selected date:", date);
  setSelectedDate(date);
  // formats again, so can match again for the list of dates
  const formattedDate = `${date.getFullYear()}-${String(date.getMonth() + 1).padStart(2, '0')}-${String(date.getDate().padStart(2, '0'))}`;
  const jobsOnSelectedDate = jobs.filter(job => job.dates.includes(formattedDate));
  setJobsForSelectedDate(jobsOnSelectedDate);
};
```

```
return (
  <div className="calendar-container">
    <h2>Calendar</h2>
    <Calendar
      tileContent={tileContent}
      onClickDays={handleClickDay}
    />
    {selectedDate && (
      <div className="jobs-list">
        <h3>Jobs for {selectedDate.toDateString()}</h3>
        <div>
          {jobsForSelectedDate.map((job, index) => (
            <li key={index}>
              <Link href={`/jobs/${job._id}`}>{job.customerDetails[0]}</Link> - {`${job.dates.length} day job`}
              <button onClick={() => {}}>Clock In</button>
            </li>
          ))}
        </div>
      </div>
    )}
  </div>
);
```

export default MyCalendar;

Middleware on Jobs Route

```
// View all Jobs — Admin/ Owners (list of jobs they are assoc with)
router.get('/', j_auth, async (req, res) => {
  res.send(await JobModel.find().populate('users'))
});
```



For the calendar component we used a package called react-calendar. The calendar has dates that are matched against the dates in the job Database. As dates are entered and stored as a string (date formats can change when transmitting in and out of MongoDB/JSON formats), the program converts the calendar dates to string format before comparing them, then checks if the formatted dates match the dates array in all our jobs data retrieved from the database using array.includes in JS/JXS. If the date matches we use inline styling to generate a small red dot within the day on the calendar.

The handleClickDay() function handles selection of the users clicking a particular day (with or without a job assigned to it) . It then utilises array.filter to remove jobs without the jobs that include the formatted date on the calendar.

The list is then generated dynamically within the return function in react. Using array.map to display all jobs that assigned to the selected day and provides a link to them.

Authority is handled by using middleware on the backend of the get jobs route. The route checks the JWT against the users id that is in the users array of the Jobs data form the database. This is how users only see jobs they are assigned to and the admin can see all.

Timesheet Component

```
import React, { useState, useEffect } from 'react'
import { API_BASE_URL } from './api/endpoints.js'
import { Link } from 'react-router-dom'

const TimeSheet = () => {

  const [employees, setEmployees] = useState([])

  useEffect(() => {

    fetch(`${API_BASE_URL}/users`, {
      method: 'GET',
      headers: {
        Authorization: 'Bearer ${localStorage.getItem('token')}'
      }
    })
      .then(res => res.json())
      .then(data => {setEmployees(data)})

  }, [])

  return (
    <ul className="employee-list" >
      {employees.map((employee, index) => (
        <li className="employee-item" key={index}>
          <Link to={`/timesheet/${employee._id}`} >{employee.name}</Link>
        </li>
      ))}
    </ul>
  )
}

export default TimeSheet
```

```
const UserTimeSheet = () => {

  const [timeSheet, setTimeSheet] = useState([])
  const { id } = useParams()

  useEffect(() => {

    fetch(`${API_BASE_URL}/timesheets/${id}`, {
      method: 'GET',
      headers: {
        Authorization: 'Bearer ${localStorage.getItem('token')}'
      }
    })
      .then(res => res.json())
      .then(data => {setTimeSheet(data)})

  }, [])

  return (
    <div className="time-sheet-container">
      <table id="print-content">
        <thead>
          <tr>
            <th>Date</th>
            <th>Hours</th>
            <th>Rate</th>
            <th>Earnings</th>
          </tr>
        </thead>
        <tbody>
          {timeSheet.map((entry, index) => (
            <tr key={index}>
              <td>{entry.date}</td>
              <td>{entry.hours}</td>
              <td>{entry.rate}</td>
              <td>{entry.earnings.toFixed(2)}</td>
            </tr>
          ))}
        </tbody>
      </table>
      <button onClick={() => window.print()} className="print">Print</button>
    </div>
  )
}

export default UserTimeSheet
```

Date	Hours	Rate	Earnings
2024-02-16	6	\$25.99	\$155.94
2024-02-24	8	\$29.88	\$239.04

Print

The timesheet component works similarly to the calendar component in that it generates based on the logged in user. Currently the timesheet can only display hard-coded date data for each user. Due to time constraints we were unable to match the clock-in/clock-out date and time data to make a POST route to be stored in the "Timesheet" collection in the database, however this will be done post-submission. The Timesheet and UserTimeSheet components handle current functionality.

The Timesheet utilises a fetch request to gather the user information from the database within a useEffect hook to gather database information and is set using the setEmployees function with useState. The employees are mapped as a list for the admin to select the employee

The UserTimeSheet will then generate the selected employed information using the useEffect hook again to gather the timesheet data for the user and display on a neat table to be rendered on the screen.

Testing

JS -Unit Testing

```
import app from '../app.js'
import request from 'supertest'

describe('App Test', () => {

  // Test on the entry route
  test('GET /', async () => {
    const res = await request(app).get('/')
    expect(res.status).toBe(200)
    expect(res.header['content-type']).toContain('json')
    expect(res.body.info).toBe('Hello!')
  })

  // Login API
  describe('Login POST test', () => {

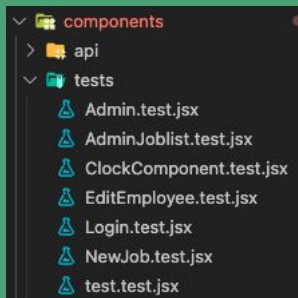
    let user

    beforeEach(async () => {
      user = await request(app).post('/users').send({
        name: 'tester',
        email: 'tester@gmail.com',
        password: 'tester'
      })
    })

    // User created successfully
    test('Returns JSON content', async () => {
      expect(user.status).toBe(201)
      expect(user.header['content-type']).toContain('json')
      expect(user.body.name).toBe('tester')
    })

    // Invalid email
    test('Login with invalid email', async () => {
      const login = await request(app).post('/login').send({
        email: 'failtext',
        password: 'tester'
      })
      expect(login.status).toBe(404)
      expect(login.header['content-type']).toContain('json')
      expect(login.body.message).toBe('User not found')
    })
  })
})
```

JSX -Unit Testing



```
import { describe, test, expect } from 'vitest';
import React from 'react';
import { render, fireEvent, waitFor } from '@testing-library/react';
import NewJob from '../NewJob';
import { JSDOM } from 'jsdom';

// Setup JSDOM
const dom = new JSDOM('<doctype html><html><body></body></html>');
global.document = dom.window.document;
global.window = dom.window;

describe('NewJob Component', () => {
  test('renders NewJob component', async () => {
    const { getByText, getByLabelText, getByPlaceholderText } = render(<NewJob />);

    expect(() => getByText('New Job')).not.toThrow();

    const customersNameInput = getByLabelText('Customers Name:');
    fireEvent.change(customersNameInput, { target: { value: 'John Doe' } });
    expect(customersNameInput.value).toBe('John Doe');

    const customersMobileInput = getByLabelText('Customers Mobile:');
    fireEvent.change(customersMobileInput, { target: { value: '1234567890' } });
    expect(customersMobileInput.value).toBe('1234567890');

    const jobAddressInput = getByLabelText('Job Address:');
    fireEvent.change(jobAddressInput, { target: { value: '123 Main St' } });
    expect(jobAddressInput.value).toBe('123 Main St');

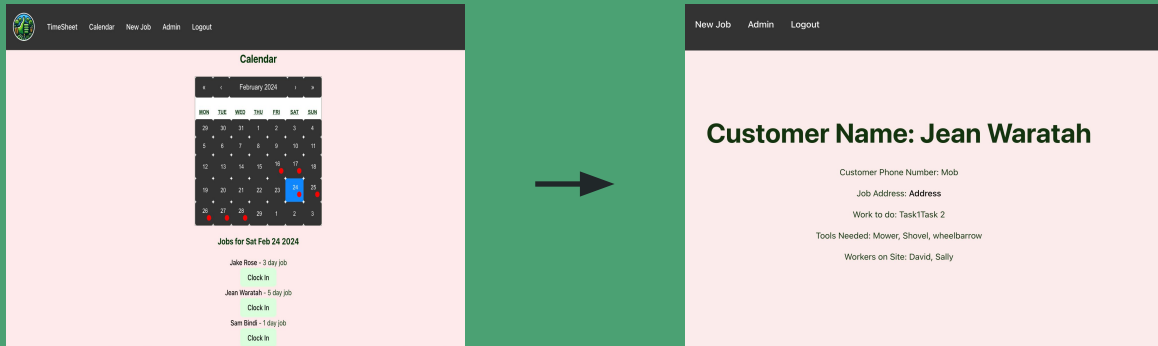
    const tasksSelect = getByLabelText('Tasks:');
    fireEvent.change(tasksSelect, { target: { value: 'Mowing' } });
    expect(tasksSelect.value).toBe('Mowing');
  });
});
```

Our project testing framework focuses on unit testing and integration testing. With Unit testing we utilise various built-in functions to perform these unit tests. Writing these tests in code and ensure we passing the tests they produce is crucial in the testing procedure.

For backend unit testing we used Javascripts built-in capabilities along with a function called 'request' imported from a package called supertest. These tests predominantly focus on testing the connection to the database via the routes and as well as the data in the request and response to this routes. These functions allow as to test the capabilities of our data validation and ensure they are working. As you can see with the photo on the left is an example of the log in test which is implemented with describe, beforeEach, test, toContain, toBe etc

In regards to frontend unit testing , similarly to Backend we write code to facilitate these tests. However quite differently to backend wned testing , we need to set-up these differently. One of the differences is unit testing is done on the individual components on their own. Furthermore, in frontend testing we will require a framework and a testing library to handle the functionality of these tests. The framework used is known as vitest which allows us to access functions like describe , test and expect as you can see the picture on the left. The testing-library named 'testing-library' , is useable with react and uses functions in React code to be testable with our code or mock data. The most common tests we use were render and fireEvent . which test the actual rendering of the component on the page and simulate an event in the DOM for input testing , respectively.

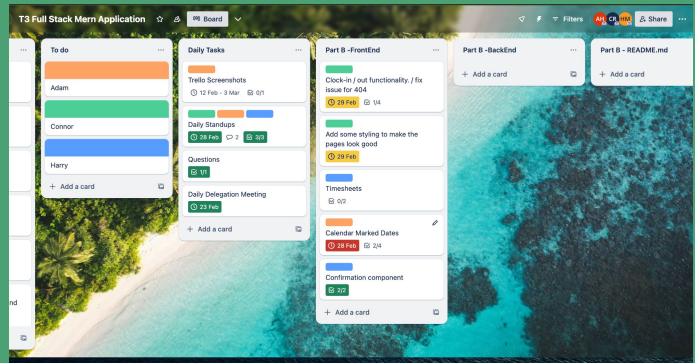
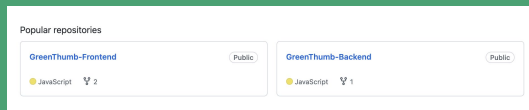
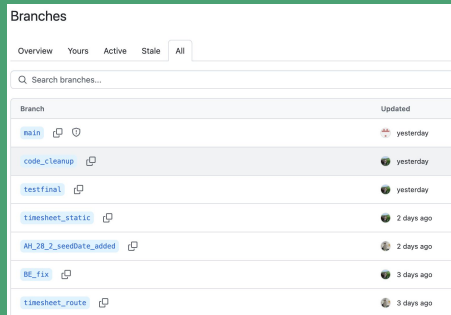
Integration Testing



For integration testing, we performed after this after unit tests were complete. Our integration testing was done manually as we navigate throughout the app. We did this by moving through the application and navigating through components and testing what the following component was and if the correct movement between these was followed. During this stage of testing we were able to catch many errors e.g The dates format when converted was auto formatted to the following day. So Jobs would appear a day ahead on the Calendar.

For the example above, we are jumping from the Jobs List in the calendar component to the ViewSingleJob component. As we recorded the transition happened as expected and displayed the intended data.

Project Management Methodology



Working in part of a team it was important to have a clear project management methodology. Our teams style was unique to our skill-sets and utilised various techniques to move towards the completion of the implemented a daily stand-up conversation where we would discuss what we were working on and potential blockers we were facing and the discuss our progress throughout the project.

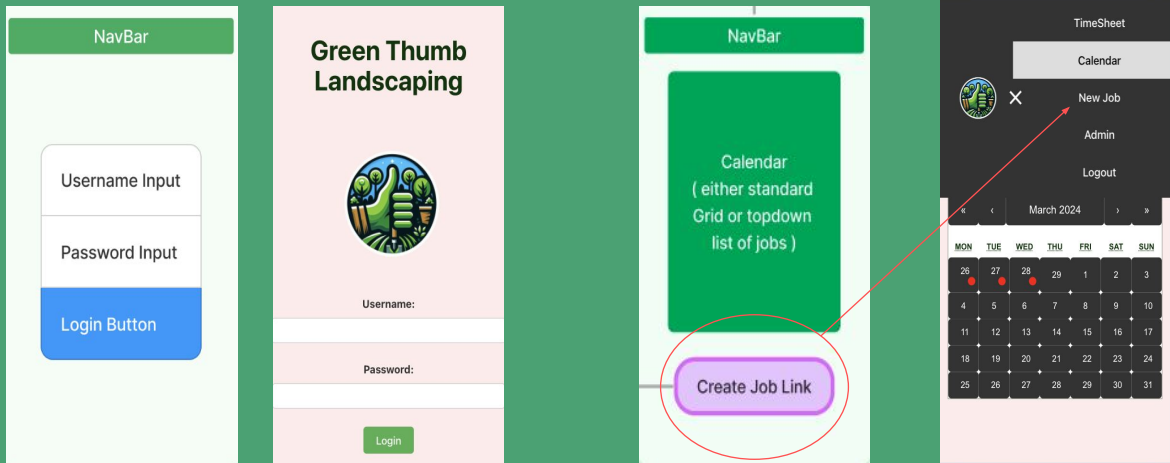
A key factor is our methodology was source control. Our system was to have our main working branch to be the master copy in Github. From here we could connect to this main and fetch and merge once an update had been added to the main. Any work we created we would be made on a separate branch and pushed to the main when ready. Another member would then check and approve/reject the generated Pull request from Github. Conflicts did happen but we easily able to communicate these problems via discord.

Two repos make up our project Greenthumb-frontend and Greenthumb-backend, we transferred these to a new Github account where we could all have access and have the ability to deploy. We also wanted this to act as us handing the project over to the customer.

To track tasks that needed to be followed we used trello.com's kanban boards to add and organize these tasks into a format where we could visually see our progress and see what work other team members had completed or were working on.

Final Layout

Discuss changes from original



For the most part our design and final result was mostly similar. The only parts which were different was that: Our Login screen initially had a NavBar. This was removed as it meant a user could potentially direct themselves to another point on the site without having logged in first. Or they would get sent to a 402 screen because they have not logged in. To fill in some of the screen space we replaced it with the name of the business and the logo. We decided to move the New Job button from the Calendar and put it into the NavBar. These changes made the app feel more user friendly. Not being able to direct themselves to pages that will result in an error from no authentication and being able to find all pages under the hamburger menu.

Our Thoughts

Features/Functions that we did not get to finish as a result of time

1. Clock-in/out is sent to the database and stored.
2. Difference in Layout from admin and standard user.
3. Admin creating timesheets off of the stored clocked hours.
4. Customer variant of app

This assignment proved to be a challenge for all of us. Coming together as a group to collaborate when we have only worked as solo developers was a challenge at first. It took some time for us to get used to working with someone else on the same code. We had some git conflicts along the way which set us back and other coding mishaps which sent us further back. We all worked together to fix each issue but sadly that took time away from us to work on more features. We wanted to discuss some of those features and how we would have implemented them if we had more time.

1. Clock-in and Clock out time being sent to the database. We made separate routes for this initially but due to some errors with the reboot of the backend we didn't have time to finish this functionality. What we would have liked to do is sync up so that on the first button press the user has a timestamp for that job sent to the database. Then the same for the Clock out. All that's needed from that point is to take the difference and convert it from milliseconds to minutes or hours.

2. Another addition we were not able to get functional due to time factors was the layout difference between admins and standard users. Initially we had this set up in the code and it worked for admin users but if the user was not an admin, nothing would load on that page. We took it out to return to it but we ran out of time. If we were able to complete it, the admin would have a different Navbar and main screen to the user and would have more functionality.

3. Admin creating timesheets off of the clocked hours of their employees. This one would help the admin create timesheets quickly. The admin presses the start and

finish date they want to generate the timesheet for, which would usually be week by week and they can see how much they have to pay their employees. From this they would be able to add-on any sick leave/annual leave/holiday leave etc and pay them accordingly. We were essentially aiming to have a full rostering site which included all of their leaves etc.

4. We wanted the customer to be able to create an account and send job requests for the manager to assign employees to, add any required equipment and then schedule it in for the requested time block and have it sent to the database. We originally planned for this to be quite big in nature, and allow the customer to be a part of the local business. However we discussed and agreed that the Manager/Owner/Admin would likely be taking phone calls from customers to log in jobs and it would make more sense for it to be completed by that person. In future, it would be a great feature to add as this could involve an initial screen asking if you are a customer or employee and it takes them to different pages depending on the answer.

Thank You