climate
County Durham

Technical Report

Group 4 - CERP Target Tracking

Elvis Kam (cqst66)
Millie Crawford (dsvg51)
Jasmeet Kaur Jasvinder Singh (fvtx23)
Hongyu Bu (hpgm94)
Rachel Dela Ysla (lwcl21)
Tirenioluwa Oladimeji (sxgm48)

13/03/25

Table of Contents

Technical Report

GitHub Repository: https://github.com/COMP2281/software-engineering-group24-25-04/tree/main/code/target-tracker_v1

Section 1: Introduction

1.1: Summary of the Project

Durham County Council (DCC) aims to achieve net-zero carbon emissions by 2030, requiring an effective method to track the progress of its climate initiatives. Their current Excel-based tracking system is inefficient due to difficulties in data navigation, frequent manual reminders, limited security, and poor visual representation. To resolve these issues, we need to develop a scalable, user-friendly platform to replace Excel and enhance productivity. The main goals of this project include:

- Improve usability and design a more intuitive user interface so that users at different levels (such as project managers, partners and related personnel) can easily enter, update and query progress information.
- Integrate automatic reminders to ensure that users update task progress on time and reduce manual intervention by managers.
- Introduce role-based permission control to ensure that different users can only access and manage target data related to them to prevent misoperation or unauthorised changes.
- Support real-time progress bar, task filtering and multiple data visualization methods to help managers understand the progress of targets more intuitively.
- Ensure that the system can run smoothly on different devices (PC, tablet, mobile phone) to meet modern office needs.

In conclusion, we hope to reduce manual management costs while improving data accuracy and transparency, so that DCC and its stakeholders can manage carbon emission targets more efficiently and promote sustainable development.

1.2: System Set Up

To access the system locally, clone the GitHub repository [https://github.com/COMP2281/software-engineering-group24-25-04/tree/main/code/target-tracker_v1] in an interactive development environment such as Visual Studio Code (with git built-in) and install Node.js from its official website. The code is split into two folders: frontend and backend. To get them to work together, you'll need to type '`npm install axios`' and '`npm install react-bootstrap bootstrap validator`' in the terminal while in the frontend directory and then ‘npm install express cors body-parser nodemailer’ in the backend directory. To start the website, split the terminal into two parts, navigate to the frontend and backend in the other, and type ‘npm start’ in both.

1.3: Status of Behavioural Requirements

BR Code	Requirement Description	Modification Status	Status	Justification
BR 1.1	Logging in without an account	Unchanged	Met	An error message is shown when a user tries to log in without creating an account.
BR 1.2	Creating an account	Unchanged	Met	Users can click on the Sign-Up icon on the Login page and will be redirected to a page where they can create an account. Users must abide by strict password requirements for security purposes.
BR 1.3	Logging in with correct credentials	Unchanged	Met	Users will be redirected to the staff dashboard and managers are redirected to the admin dashboard upon logging in with their registered credentials.
BR 1.4	Logging in with incorrect credentials and attempt is less than three	Unchanged	Met	An error message pops up each time incorrect credentials are used to login. Users and managers are not redirected to their respective dashboards.
BR 1.5	Logging in with incorrect credentials on third attempt	Unchanged	Not Met	Currently, an error message is displayed each time incorrect credentials are entered. However, a limit on failed login attempts has not yet been implemented, and users are not prompted via email to reset their password. A login attempts limit, and automated password reset prompts will be implemented before the product is submitted to strengthen account protection.
BR 2.1	Permission to view other teams’ tasks	Modified	Met	Initially, users were only allowed to view and edit their own assigned targets, requiring permission to view other targets. However, in the implementation, all users can view all targets, but they can only edit the targets assigned to them. This change was made to foster transparency, collaboration, and a better understanding of organizational goals.
BR 2.2	Manager granting permissions to users	Unchanged	Met	Managers can assign different targets to users and these targets will show up on the My Targets section of the respective user’s dashboard. Only authorized users are permitted to make edits, while targets not assigned to a user are available in read-only mode. This maintains clear responsibility and prevents unauthorized modifications.
BR 2.3	Manager transferring tasks from one user to another	Unchanged	Met	On the manager’s dashboard, the *Assign Target* feature enables managers to assign targets to users, while the *Remove Target* feature allows them to reassign or remove targets from existing users. This functionality facilitates seamless task transfers between users, ensuring efficient workload management and adaptability to changing responsibilities.
BR 3.1	Adaptive user layout on different devices	Unchanged	Met	The dashboard has been tested across multiple devices, confirming that the layout dynamically adjusts to different screen sizes. The responsive design ensures a seamless user experience, maintaining usability and functionality across desktops, tablets, and mobile devices.

BR 3.2	Customisable visual settings	Unchanged	Not Met	The user and manager dashboards currently follow a single colour scheme to maintain design consistency. Colour customization and font size adjustment features have not yet been implemented, as priority has been given to the *Must-Have* requirements. However, these features will be implemented before the product is submitted.
BR 4.1	Filter tasks based on Day/Week/ Month/Year	Unchanged	Not Met	Only filtering based on target title has been implemented at present. Users and managers can select respective targets and only the chosen targets will be shown on their dashboards. Filtering based on the period will be implemented before the product is submitted.
BR 4.2	Filter tasks based on custom date range	Unchanged	Not Met	Only filtering based on target title has been implemented at present. Users and managers can select respective targets and only the chosen targets will be shown on their dashboards. Filtering based on custom date range will be implemented before the product is submitted.
BR 4.3	Message when no tasks are due within selected period	Unchanged	Not Met	Currently, no message is displayed because filtering based on the selected period has not yet been implemented. However, once the *Day/Week/Month/Year* filtering feature is added, users will also receive a message indicating when no tasks match their selected criteria. At present, when using the target title filter, if a target is not selected, it simply does not appear on the dashboard without any notification.
BR 4.4	Clear filter	Unchanged	Met	When a specific target title is not selected on the target filter bar, the dashboard shows all the targets available.
BR 5.1	Add a goal	Unchanged	Met	Both users and managers can add a new target by clicking on the *Add a New Target* box. This action redirects them to an empty form where they can enter the target details and save it. Once saved, the new target becomes visible on both the user and manager dashboards, ensuring seamless goal tracking and management.
BR 5.2	Remove a goal	Unchanged	Met	Users and managers can click the remove goal icon on their dashboard to delete a target once the users have completed it or if the target is no longer needed. Once the target is removed, it will no longer show up on the dashboard.
BR 5.3	Edit a goal	Unchanged	Met	Both users and managers can click on the edit goal icon and make changes to their existing targets. The edited targets are then updated to replace the old ones.
BR 5.4	Transfer a goal	Unchanged	Met	The assign target and remove target functions available on the manager's dashboard allows seamless transfer of a target from one user to another. The manager can first remove the target from the original user and then reassign it to a new user.
BR 6.1	Staff viewing their own progress	Unchanged	Met	Users can see their progress based on the level of the progress bar of a target. Each time a user updates the value of the target set on the target description page; the progress bar will reflect the progress of each target on the dashboard.

BR 6.2	Manager viewing staff's progress	Unchanged	Met	Progress bars have been implemented to track user's progress for each target. These progress bars are visible on the manager's dashboard, allowing managers to monitor staff performance.
BR 7.1	Automated reminders on project progress	Unchanged	Not Met	Currently, users receive automated reminders for tasks based on their due dates. These emails can also serve as prompts for users to update their project progress. However, to enhance clarity and efficiency, separate emails for task deadlines and progress updates will be implemented. This improvement is planned for completion before the product submission.
BR 7.2	Automated reminders for overdue tasks	Unchanged	Met	The system automatically generates and sends email reminders for due and overdue targets by querying target completion dates, mapping targets to users, and dispatching notifications via NodeMailer and the Mailersend SMTP API. This ensures timely task completion and enhances user accountability.
BR 7.3	Change in automated reminders when project is transferred	Unchanged	Not Met	The system does not yet send automated email notifications to users when a project is transferred. This feature will be implemented to ensure users are informed of task reassignments. At present, users are only receiving email reminders for due and overdue targets.
BR 8.1	Viewing target details	Unchanged	Met	When a target is clicked on the dashboard, users and managers are redirected to a page that has a detailed description of the specific target.
BR 8.2	Clicking on an icon to carry out a function	Unchanged	Met	Every icon on both the manager and user dashboard works well and carries out their respective functions.

Section 2: Technical Development

2.1: Source Materials

The basis of our system was illustrated by the project brief provided to us by our client. We were given an expected output that should be aimed for. It outlined that the system needs to be usable by a variety of users, usable on a variety of computers, and should give a good, visible tracking of progress towards targets. They also highlighted that users shouldn't be able to delete other users' data. This is how we decided user roles should be implemented. Monthly reminders were also a suggested feature.

We had a meeting with our client so that we could understand what the company meant by 'target tracking'. They provided us with an Excel spreadsheet, containing all their goals and the data associated with those goals. Upon inspection, the spreadsheet was convoluted. We asked our client to confirm what each column was tracking, and which columns were necessary or not.

Following our market research we compared multiple target tracking applications, including Notion and Jira. We identified key features that could be useful to implement into our system but also noted that these current systems available wouldn't be suited to our client. It was suggested in the brief that the solution could be web-based. This influenced our decision for our solution to be web-based and not app-based.

After our research we spoke to the client to confirm that what we were planning was in their scope and met their needs.

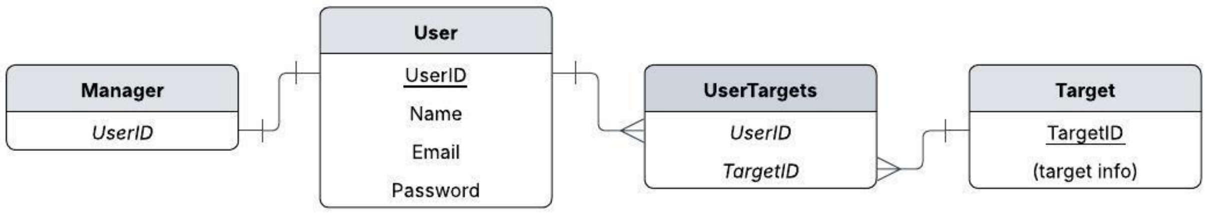
We started our project by viewing a YouTube tutorial explaining how to create a login page with React.js. This provided us with a template for our project to expand from.

2.2: Technical Description

Technologies and Platforms Used

Component	Description	Technology
Front-end	UI component libraries	React
Front-end	HTTP client for the browser and Node.js to send HTTP requests from client-side JavaScript and Node.js applications	axios
Front-end	Data validation and sanitization Node.js library	validator
Front-end	Front-end framework for faster and easier web development.	bootstrap
Back-end	Node.js-based web application framework that simplifies server-side application development	Express
Back-end	Configuring and managing cross-domain requests for web applications	CORS
Back-end	Middleware for Node. js that parses incoming request bodies and makes them available as objects in the req. body property	Body-parser
Back-end	Send emails from application by providing a high-level API for interacting with SMTP servers	NodeMailer

For this prototype version we have used JSON files to store the users and target data. This can be switched to databases, such as MongoDB, on implementation if desired. See below the structure of the database:



The Software Development Process Followed by the Team

We adopted the Agile Scrum methodology for the development process. This approach was chosen due to its flexibility, iterative development cycles, and focus on continuous client feedback. The methodology allowed us to manage tasks efficiently, adjust to changing requirements, and ensure that the final product met the client's expectations.

The key aspects of the development process were:

1. Sprint-based development:
   - We divided the project into multiple sprints, each focusing on a specific set of tasks.
   - Early sprints involved requirement gathering and initial documentation, system design, and learning relevant technologies.
   - Later sprints focused on further documentation, coding, testing, and refining application features.
2. Task assignment and collaboration:
   - Tasks were distributed amongst ourselves, based on individual workload and expertise.
   - Weekly meetings helped track progress, address roadblocks, and adjust priorities.

3. Periodic client interaction:
   - We maintained periodic meetings with our client to gather feedback and ask requirements-related questions.
   - This ensured the solution aligned with the client's needs and expectations.
4. Technology stack selection:
   - Frontend: To develop the frontend, React.js was chosen for its efficiency and dynamic UI capabilities.
   - Backend: To develop the backend, Node.js with Express was chosen for its ability to handle routing and API requests effectively, and for its ease of setup and maintenance.
   - Database: Instead of a traditional database, JSON files were used to store data, providing a lightweight, file-based approach that simplified data access and reduced setup complexity.
5. Version control and risk management:
   - GitHub was used for code-wise collaboration, tracking changes, and preventing code loss, while Google Docs facilitated collaboration on documentation.
   - Risks, such as framework updates and security concerns, were proactively mitigated.
6. Final testing and deployment:
   - The last sprint involved testing all features, fixing bugs, and ensuring smooth operation and adherence to client requirements before deployment.

By following the Agile scrum methodology, we maximised adaptability and developed a solution that effectively replaced the inefficient Excel-based tracking system.

The Role of Behaviour-Driven Development in Implementation

Behaviour-driven development played a crucial role in the implementation phase by ensuring that development was guided by clear, user-centric requirements.

It was applied in our project through:

1. User stories and scenarios:
   - Features were defined using user stories and Gherkin pseudocode (for expressing the user story as a feature with scenarios) written in a structured format.
2. Feature prioritisation:
   - We classified features using the MoSCoW system, ensuring that critical functionalities like login, permission levels and progress tracking were developed promptly.
3. Test-driven development integration:
   - The user stories and scenarios developed were converted into test cases to validate expected system behaviour, enabling early capturing of errors and aligning development to user expectations.
4. Enhanced collaboration:
   - Behavioural-driven development facilitated clear communication between group members, enabling effective contributions and overall efficiency.

By leveraging behaviour-driven development, the implementation phase was structured around real-world user interactions, leading to a more intuitive and functional system.

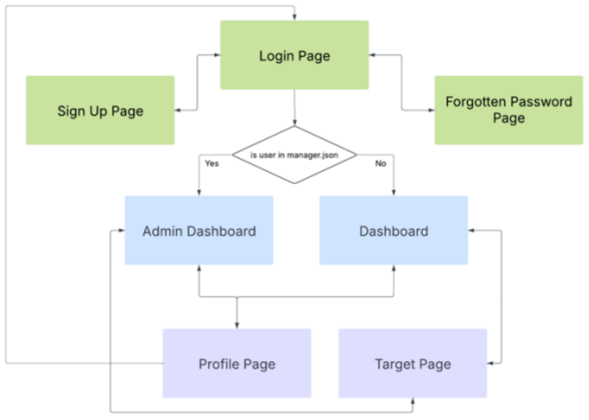
Design Patterns

In the frontend we used React, which allowed us to use the container presentational pattern. The containers handle the logic and manage the state of the application. They fetch data and handle user interactions. The container component doesn’t handle the UI, it focuses on the data. The presentational component is the part that focuses on the UI. They solely focus on receiving the data and rendering it, not where it comes from. We implemented this design pattern in Targets.jsx, having components; TargetField, TargetDropdown and

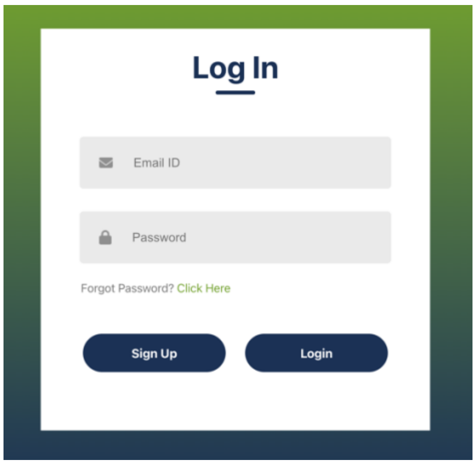
TargetDate. The use of this encourages the separation of concerns. The presentational components are also then easier to reuse, as it displays all data in the same way.

In the backend we used Node.js, which allowed us to use the controller-service-repository pattern. The controller layer handles the HTTP requests and processes them, returning their responses. This can be seen in our router.js file. The service layer has the business logic, processing the data received and performing the operations. This can be seen in our functions in router.js. The repository layer handles the data access by communicating with the JSON files, saving, updating or deleting data. This again aids separation of concerns, ensuring each layer has its own responsibilities.

Technical Implementation



The flowchart depicts the flow of our website. The user will enter at the 'Login Page' and can access the 'Sign Up Page' and 'Forgotten Password Page'. The decision node then dictates which dashboard the user will be directed to, based on the manager.json file. Each dashboard can access the 'Target Page' and the 'Profile Page'. The 'Profile Page' is the only page that has access back to the 'Login Page' once past that decision node. This flowchart justifies the flow of the explanations below.



Login Page

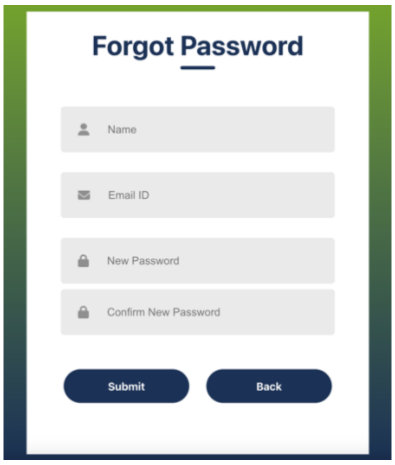
Provided the user has an account, enter the email and password into the respective boxes, then click the 'Login' button. The handleLoginClick function is called, which will send a post request to the backend. The frontend will receive which dashboard to send the user to, staff or manager.

The backend will check to see if the inputted post data matches an entry in the user.json file. If there is a match, the user's role is checked in the manager.json file. If the user id is in that file, then the user is a manager. This is returned to the frontend to direct the user to the correct dashboard.

```
Endpoint:  POST /login
Request: {"email": "example@example.com", "password": "Password123!"}
Response on Success:
{User found:
{"id":"1","name":"example","email":"example@example.com","password":"Password123!","role":"user"},
Manager Data: {'3': true, '5': true, '6': true}, User ID: 1 | Is Manager: false}
Response on Failure: {Login failed: Invalid credentials.}
```

Challenges

The challenge with the login system was that it was the first backend code to be written which deemed to be overwhelming at the start. The server had to be configured as well as the code written to read the database and authenticate users. To overcome this, the task was broken down into smaller more manageable steps.



Forgotten Password Page

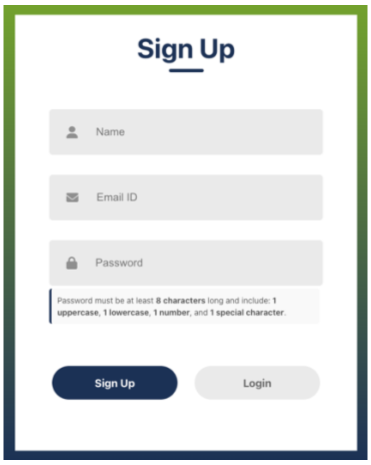
On the Login page there is a 'Forgotten Password' link that allows the user to reset their password. If this link is clicked the action is set to "Forgot Password". The 'Forgotten Password' page is loaded, allowing the user to enter their name, email and their new password twice. When the user clicks the 'Submit' button the handleResetClick function is called. This will send a post request to the backend.

The backend will check the user.json file to see if the name and email match. If the user and email inputted exists, the old password is overwritten with the new password. If successful, the action is set to "Login", and the user is directed back to the Login page.

Endpoint: POST /reset
Request: {"name": "newUser","email": "newUser@example.com","password": "NewUser1!"}
Response on Success: {New password updated}
Response on Failure: {Error updating password: {error}}

Challenges

We initially faced navigation issues – the Forgotten Password page did not direct the user to the Login page. To address this, we ensured that the forgotten password workflow included clear success messages and redirected users to the login page after a successful reset.



Password must be at least **8 characters** long and include: **1 uppercase**, **1 lowercase**, **1 number**, and **1 special character**.

Sign Up Page

If the user clicks the 'Sign Up' button. This will implement useState from React, to set the action to "Sign Up" and will call the handleSignupClick function. This will send a post request to the backend. The frontend will then load to Sign Up page. The user then enters their name, email and password, and clicks the 'Sign Up' button. Form validation is implemented to ensure that all fields are filled, if there are missing fields then an alert is shown.

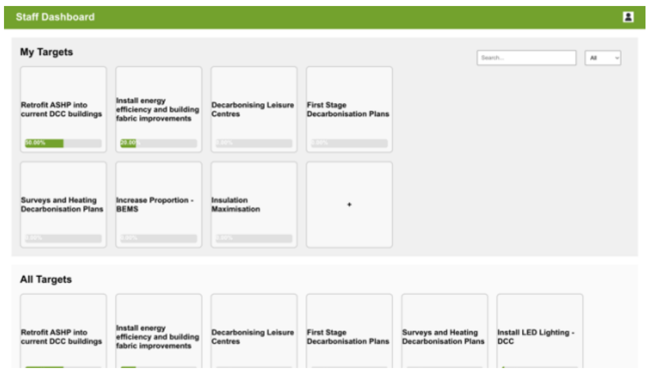
The backend will check the user.json file to ensure the email hasn't already been registered. If it hasn't, the users account is then added to the user.json file. The user is then directed to the Login page to login.

Endpoint: POST /signup
Request: {"name": "newUser","email": "newUser@example.com","password": "NewUser1!"}
Response on Success: {New user registered: {id: '9', name: 'newUser', email: 'newUser@example.com',password: 'NewUser1!',role: 'user'}}
Response on Failure: {Error saving user: {error}}

Challenges

Validating email formats and enforcing strong passwords deemed a challenge for the Sign-Up page. We overcame this by utilizing validator, a robust input validation library for checking email format and password criteria.

Staff Dashboard


Staff Dashboard
My Targets
Search...
All
Retrofit ASHP into current DCC buildings
Install energy efficiency and building fabric improvements
Decarbonising Leisure Centres
First Stage Decarbonisation Plans
Surveys and Heating Decarbonisation Plans
Increase Proportion - BEMS
Insulation Maximisation
+
All Targets
Retrofit ASHP into current DCC buildings
Install energy efficiency and building fabric improvements
Decarbonising Leisure Centres
First Stage Decarbonisation Plans
Surveys and Heating Decarbonisation Plans
Install LED Lighting - DCC

When a user is directed to the Staff Dashboard, the useEffect function is triggered. This fetches the targets, by sending multiple axios get requests. The first one gets the user id by sending the userEmail and the second gets the user specific targets. The user's targets are then set to myTargets. Each target card is then populated with the information and displayed on the frontend. All targets are displayed under the heading ‘All Targets’. There are also filters and a search box present, what the user types/selects is converted to lower case and compared to the values stored in the targets.title. A progress bar is displayed that shows the progress of each target, based on the ‘Target Set’ field.

The backend retrieves all the targets from targets.json and retrieves the userID using the userEmail from user.json. It then uses that to collect the user’s targets, based on what is stored in usertargets.json. This information is then sent back to the frontend.

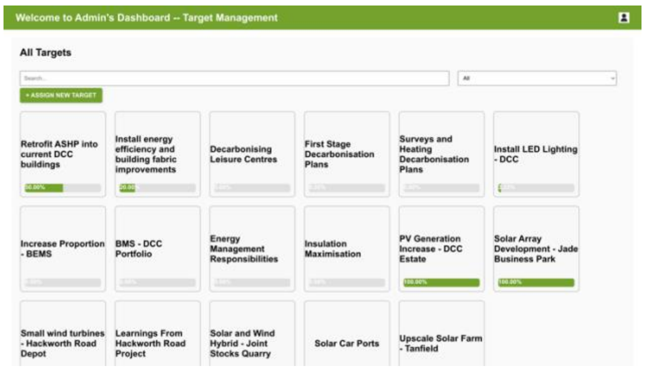
Endpoint: GET /user/:email
Request: {“email”: “email1@example.com”}
Response on Success: {{“id”:”1”}}
Response on Failure: {error: User Not Found}

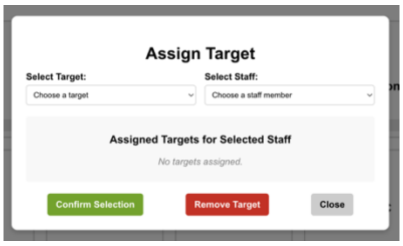
Endpoint: POST /target/:id
Request: {“target-id”:”1”}
Response on Success: {{"target-id":1,"fields":[{"id":"target-cerp3_action_type",…}}
Response on Failure: {message: target-id is required}

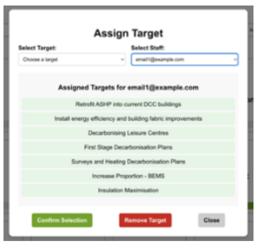
Challenges

During the development of the staff dashboard, several challenges were faced including initial difficulty in adding and editing new targets, creation of duplicate targets when an already existing target was edited, rendering target titles into the target cards, inability to render user data and logging the user out after refreshing the page.

Manager Dashboard


Welcome to Admin's Dashboard – Target Management
All Targets
Search...
All
+ ASSIGN NEW TARGET
Retrofit ASHP into current DCC buildings
Install energy efficiency and building fabric improvements
Decarbonising Leisure Centres
First Stage Decarbonisation Plans
Surveys and Heating Decarbonisation Plans
Install LED Lighting - DCC
Increase Proportion - BEMS
BMS - DCC Portfolio
Energy Management Responsibilities
Insulation Maximisation
PV Generation Increase - DCC Estate
Solar Array Development - Jade Business Park
Small wind turbines - Hackworth Road Depot
Learnings From Hackworth Road Project
Solar and Wind Hybrid - Joint Stocks Quarry
Solar Car Ports
Upscale Solar Farm - Tanfield




Assign Target
Select Target:
Choose a target
Select Staff:
email1@example.com
Assigned Targets for email1@example.com
Retrofit ASHP into current DCC buildings
Install energy efficiency and building fabric improvements
Decarbonising Leisure Centres
First Stage Decarbonisation Plans
Surveys and Heating Decarbonisation Plans
Increase Proportion - BEMS
Insulation Maximisation
Confirm Selection
Remove Target
Close

When a manager is directed to the Manager Dashboard, the useEffect function is triggered. This fetches the targets, by sending an axios GET request. This gets all the targets from the targets.json file. Each target card is then populated with the information and displayed on the frontend in individual target cards under the heading 'All Targets'. Again, there are filters on this page. However, the filters are only for 'All Targets' as there is no 'My Targets' section. When the manager clicks the ‘Assign Target’ button, a popup model is presented. The manager can then select a staff members email from the dropdown menu, this will trigger fetchAssignedTargets, where the staff members targets will show. When the manager selects a target from the dropdown list, and clicks the ‘Confirm Selection’, the front end will send a post request with the selectedTarget, and userEmail. This will then call fetchAssignedTargets. The same happens for when the user clicks the ‘Remove Target’.

The backend then retrieves the targets from targets.json and returns this to the frontend.
For ‘Assign Target’ the backend retrieves the staff members id using their email that is passed through by checking the user.json file. Once the userID is found, the backend will check the user’s assigned targets in the usertargets.json file, then retrieving the specific targets from the targets.json file, before sending it to the frontend.

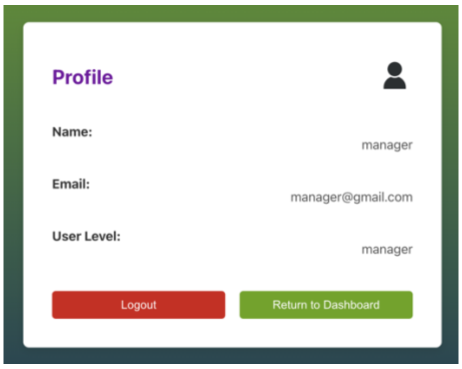
Endpoint: POST /targets
Request: No additional request fields
Response on Success: Same as POST /target but displays all targets, not just one.
Response on Failure: {Error fetching targets: {error}}

Challenges

During development we had many integration issues, due to multiple technologies clashing. Different frameworks used conflicted with each other. This led us to changing certain implementations so that it would function seamlessly.

Profile Page

When the user clicks the profile icon on the Dashboard page, an onClick is triggered, which calls the handleIconClick. This calls the goToProfile function, passing in the user’s email. The frontend sends an axios POST request to send the postdata (email) to the backend. On success, the user's information (name, email, role (user or manager)) is displayed on the frontend profile page. There is a 'Logout' button that when clicked calls the handleLoginClick function, logging the user out and returning them to the Login page. There is also a 'Return to Dashboard' button, that calls the handleDashboardClick function, taking the user back to their dashboard.


Profile
Name: manager
Email: manager@gmail.com
User Level: manager

The backend checks the user.json file to retrieve the user’s information, based on the email given. The name and role (user or manager) are retrieved and all 3 are sent back to the frontend.

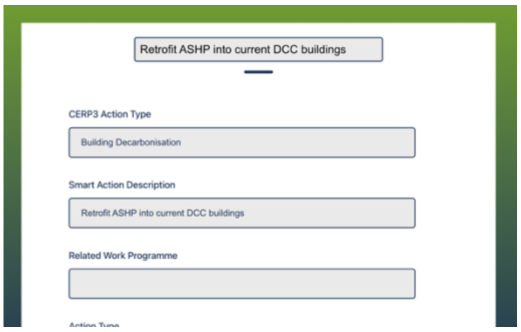
Endpoint: POST /userData
Request: {"email": “example@example.com”}
Response on Success: {"name": "Example","email": “example@example.com”,"role": "user"}
Response on Failure: {Error retrieving user data: {error}}

Challenges
During development, when we first had this loading, we noticed an error that would mean that when the user first logged in, and went straight to the profile page, it worked. But if the user had logged in, clicked on a target, then went back to the dashboard and clicked the profile page, the data wouldn't load. This was fixed as soon as it was noticed and now loads the correct user information whenever the page is visited.

Target Page

When a user clicks on a target box, they are directed from the Dashboard to a Target page. The frontend uses the React's useEffect function which sets the form data for the target that is passed through (the target that has been clicked). On the Target page, the user is loaded into view mode, where each field has its own heading and display box. The front end calls each component, TargetField, TargetDropdown, and TargetData. These are separate as they relate to different frontend components (input, dropdown, date). The id is set based on the ids stored in the targets.json file. On view mode the display boxes are paragraph tags. When the user clicks the 'Edit' button the action is changed to 'Edit'. If isEditing is True, then the fields are changed from a <p> to either an input tag, dropdown, or date. These can then be edited by the user. A 'Save' button then displays at the bottom of the page. When the user changes an input field, and clicks the 'Save' button, the frontend sends an axios post request to the backend with the target data.
When the user clicks the 'Return to Dashboard' button, the handleDashboardClick function is called, taking the user back to their dashboard (staff or manager).

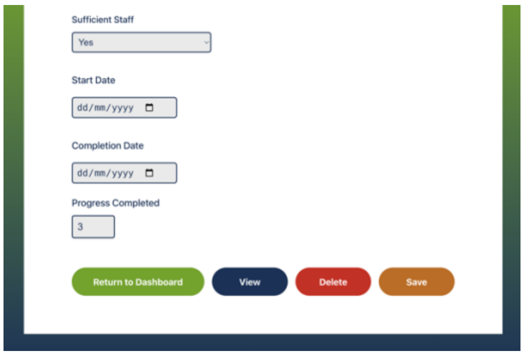
When the 'Save' button is clicked, the backend will receive the data from the frontend and write it to the targets.json file. Then the changes are saved into the targets.json file, and these are reflected in the frontend.


Retrofit ASHP into current DCC buildings

CERP3 Action Type
Building Decarbonisation

Smart Action Description
Retrofit ASHP into current DCC buildings

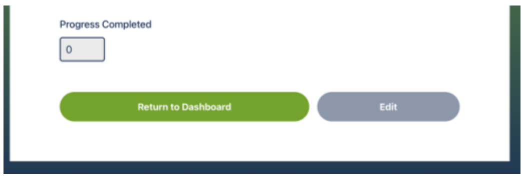
Related Work Programme


Sufficient Staff
Yes

Start Date
dd/mm/yyyy

Completion Date
dd/mm/yyyy

Progress Completed
3


Progress Completed
0

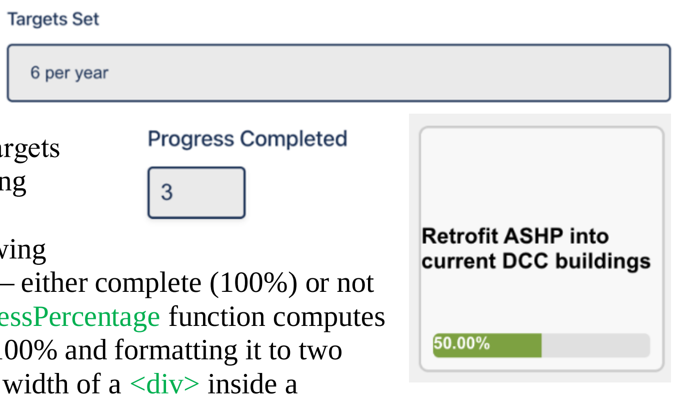
Endpoint: GET /usertargets
Request: No additional request fields
Response on Success: {"1":[1,2,3,4,5,7,10],"2":[6,7,8,9,10],…}
Response on Failure: {message: Failed to load user targets}

Endpoint: POST /usertargets/:userEmail
Request: {"userEmail":" email1@example.com"}
Response on Success: {["Retrofit ASHP into current DCC buildings", …]}
Response on Failure: {message: Failed to load user targets}

Challenges

Trying to get the targets to load onto the frontend was quite complicated. The main cause of issue was the post requests going to the wrong location. Once this was fixed, it was more straightforward to find the solution.
When the user edits a target, we had an issue where the edited target would save a new target to the targets.json file with the same target-id, instead of overwriting the original target. To solve this, a function that checks if the target already exists was created, then replaces the existing target data with the new target data.
We also faced the issue of how to manage user permissions for this section. We settled on disabling the 'Edit' button if the user had clicked on a target that wasn't assigned to them based on the usertargets.json file.

Progress Tracking and Updating


Targets Set

6 per year

Progress Completed

3

Retrofit ASHP into current DCC buildings

50.00%

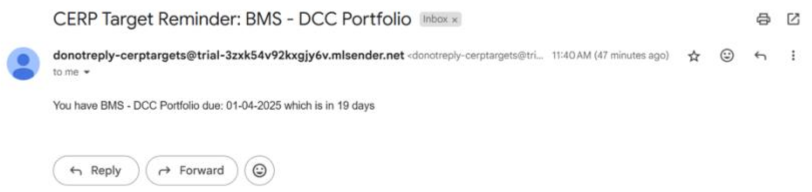
The progress bar logic dynamically calculates and visually represents target completion while enforcing valid user input. It extracts the total number of tasks from the “Targets Set” field using the extractTotal function, ensuring consistent calculations across components. If no numeric value is found, the total is set to 1, allowing the progress to be treated in a Boolean manner — either complete (100%) or not (0%). In the dashboard component, the getProgressPercentage function computes progress as (progress * 100)/total, capping it at 100% and formatting it to two decimal places. This value is then applied as the width of a <div> inside a progress bar, providing real-time visual feedback. In the target component, when a target is loaded, the maxProgress state is set based on the extracted total, and the “Progress Completed” field allows user modifications while ensuring values stay within 0 and maxProgress. The handleProgressChange function also prevents invalid inputs by clamping values within the allowed range. Progress values changed under the “All Targets” section by the administrative staff is directly reflected on the other staff’s pages as well.

This integration ensures a seamless experience where progress updates are both visually intuitive and data-consistent, supporting both precise and binary tracking of completion.

Challenges

During development, the main challenge faced was ensuring that progress percentage calculations were accurate and consistent, and determining how to treat targets that did not have a numerical value set. This was solved by using string matching to derive a total and then using the user input under the “Progress Completed” field to determine how much of the total has been completed.

Target Email Reminders


CERP Target Reminder: BMS - DCC Portfolio

donotreply-cerptargets@trial-3zxk54v92kxgjy6v.mlsender.net <donotreply-cerptargets@tri... 11:40 AM (47 minutes ago)

to me

You have BMS - DCC Portfolio due: 01-04-2025 which is in 19 days

When the system is operational, it queries the targets.json file to identify all targets that are due within the next month, 3 months, 6 months, or overdue. This is done by reading the target competition date, calculating the number of days that is from the current date, and saving the necessary information temporarily.

After identifying the due or overdue targets, the system cross-references this data with user-target relationships stored in userTargets.json. The query ensures that each target is matched with the associated user(s).

For each user, the system obtains their email from the users.json file and generates a separate email for each relevant target. These emails are tailored to notify users about specific due or overdue targets. These emails include key information such as the target’s title, due date and how many days it is due in.

The email dispatch is facilitated through NodeMailer, a Node.js library, in conjunction with the MailerSend SMTP API. NodeMailer is configured with MailerSend’s SMTP server, port and authentication credentials, and leverages MailerSend’s service for reliable delivery.

Challenges

Facing budget constraints, many secure email delivery services were out of reach. Prioritizing security, we avoided resorting to free yet unreliable options, which extended the time required to find an appropriate solution. Ultimately, we secured a free trial with MailerSend—a reliable and well-suited choice for our system.

2.3: System Usability and User Experience

From our discussions with our client, we gathered that the solution should be simple and intuitive to use, due to the varying technical abilities of the users. We set out a clear design that would allow users to easily navigate our site.

The Login page is straightforward, where the user enters their email and password, and clicks the 'Log In' button. If the user doesn't have an account, then the user needs to click the 'Sign Up' button. This then allows them to enter their name, email and password. If the user has forgotten their password, then there is a link that allows them to reset it. All the text boxes have placeholders telling the user what they need to input. The email input box has an email icon, and the password input box has a password icon. This is all for ease of understanding for the user.

When the user logs in they are directed to their dashboard. Their targets are displayed under 'My Targets' and all targets are displayed under 'All Targets'. This separation is for better usability. There are filters that the user can apply so that they can view specific targets. When the user clicks on a target, they are directed to said Target page, where they can view all the target information. There is an 'Edit' button at the bottom of the page which the user can click if it is one of their targets. There is then a 'Save' button which the user can use if they have edited their target. The ‘Delete’ button is in red so that the user can differentiate and signifies a warning colour.

The Target page and Profile page both contain a 'Return to Dashboard' button. This is for ease of understanding, as opposed to having a 'Back' button. The profile page contains the ‘Logout’ button, that again is in red.

Our colour schemes were selected based on the client’s colour scheme, green and navy. The specific colour values were obtained from their logo. This is to ensure consistency between their systems.

Section 3: Use Instructions

3.1: Software Installation Guide: System Requirements and Installation Configuration Steps

3.1.1 Hardware Requirements

Component	Minimum Requirement	Recommended Configuration
CPU	Dual-core 2.0GHz	Quad-core 3.0GHz+
Memory	4GB	8GB
Storage	10GB HDD	20GB SSD

3.1.2 Operating System Requirements

Operating Systems	Minimum Requirement	Recommended Configuration
Windows	Windows 10 or later	Windows 10 Pro or later
Linux	Ubuntu 20.04 LTS or equivalent distribution	Latest Long-Term Support (LTS) distribution (e.g., Ubuntu 22.04 LTS)
macOS	macOS 10.15 (Catalina) or later	macOS 11 (Big Sur) or later

3.1.3 Software Installation and Configuration Guide

The software is finally presented in the form of a website. Compared to using an app presenting the software as a website, it offers several advantages, such as cross-platform compatibility, no need for users to download and install applications (which reduces storage space usage), and easier updates and maintenance. It is recommended to use a web browser for accessing the website, with Google Chrome and Firefox being the preferred options. Below are the installation steps for these browsers.

Chrome

Download: Visit the Chrome official website to download the installation package suitable for your operating system.

Installation:

- Windows Users: Run the downloaded .exe file and follow the prompts to complete the installation
- macOS Users: Open the .dmg file and drag Chrome to the Applications folder
- Linux Users: Use the .deb or .rpm package for installation

Configuration:

- Go to **Settings** → **Privacy and Security** to adjust cookie, cache and script execution permissions.
- Install **Developer Tools** for debugging. Use shortcut keys F12 or Ctrl + Shift + I to open.

Firefox

Download: Visit the Firefox official website to download the installation package suitable for your operating system.

Installation:

- Windows Users: Run the downloaded .exe file and follow the prompts to complete the installation
- macOS Users: Open the .dmg file and drag Chrome to the Applications folder
- Linux Users: Use the .deb or .rpm package for installation

Configuration:

- Go to **Settings → Privacy and Security** to adjust cookie, cache and script execution permissions.
- Install **Developer Tools** for debugging. Use shortcut keys F12 or Ctrl + Shift + I to open.

3.1.4 Code Acquisition and Execution

If customers need to manage the software internally on their servers, the following steps provide local execution instructions: This software adopts a front-end and back-end separation architecture, allowing both teams to develop in parallel, reducing mutual dependencies. The frontend handles page rendering, while the back end focuses on data logic, lowering server load. The frontend and backend can be upgraded independently without affecting each other.

Node.js

Download: Use the official installer to install directly. Visit the Node.js official website to download version v22.14.

Verify Installation:

```
node -v   # Ensure Node.js version is v22.14
npm -v    # Ensure npm (node package manager) is correctly installed
```

Obtain and Run the Code via Git

Clone the Repository:

```
git clone https://github.com/COMP2281/software-engineering-group24-25-04.git
```

Install Frontend Dependencies:

```
cd “code/target-tracker_v1/frontend”
npm install react-bootstrap bootstrap axios validator
```

Install Backend Dependencies:

```
cd “code/target-tracker_v1/backend”
npm install express cors body-parser nodemailer
```

3.2: Deployment

Deploy Node.js and Express-based web applications directly in your local environment. The deployment solution is compatible with Windows, Mac, and Linux. In addition, the solution supports devices within the same Local Area Network (LAN) to access the application via IP address, such as multiple computers connected to the same Wi-Fi hotspot.

3.2.1 Prerequisites

Ensure that hardware, operating system, and web browser requirements are as stated in Section 3.1. Clone GitHub repository and install required dependencies.

3.2.2 Deploying the Web Application

Run terminal in frontend folder and execute:

```
npm run start
```

Run terminal in backend folder and execute:

```
npm start
```

In default browser, http://localhost:3000 will display the web-application.

3.3: Startup and User Account Management

3.3.1 Creating User Accounts or Logging into the System

The system provides two access methods:

- Creating a New Account: Users who register through the front-end interface are assigned a low-permission role by default. If department managers or other users require a high-permission account, they should contact the system administrator, who can add high-permission accounts through the backend.
- Logging into an Existing Account: Users can log into the system using their registered username and password.

3.3.2 User Roles and Permissions

Root Account (Highest Permission)

- Logs in directly using the system's preset username and password.
- Has access to the Administrator Page to manage all user accounts.
- Can assign administrator permissions to standard user accounts from the User Overview Page.

Administrator Account

- Requires users to first register as a standard account, then be upgraded by the Root account.
- Once authorized, administrators can access the exclusive Administrator Panel, with the same permissions as the Root account.
- Administrator Task Panel:
    - Can add new targets.
    - Can modify existing target content.
    - Can grant target modification permissions to standard users.

Standard User

- Registers an account directly on the Registration Page.
- Once created, users can log into the Standard User Task Interface to perform operations.
- Permissions:
    - Can view all targets but can only modify their own target (My Target section).

3.3.2 Initial Setup Guide

Creating a Root Account:

- Log in using the default credentials or reset the Root account during installation.
- When adding the first high-permission super administrator account to the database, it is usually necessary to operate directly on the database (if used).

Registering an Administrator Account:

- Visit the Registration Page to create a standard account.
- Upgrade the account to super administrator by granting permissions via the Root account.

Registering a Standard User:

- Visit the login page, fill in the required information, and complete the account creation.
- Once registered, users can log in and access the Standard User Task Interface with appropriate permissions.

3.4: Common Errors and Log Queries

3.4.1 Common Error Messages and Corresponding Solutions

Error code	Description	Solution
HTTP 500	Internal server error	Refresh the page or clear your browser’s cache.
HTTP 404	Page/target not found	Check URL for typo/incorrect domain name
HTTP 401	Unauthorised	Provide valid login credentials
HTTP 408	HTTP request timeout	Check the network connection and service status between the client and the server.
HTTP 503	Service unavailable	No solution, wait until site is back up and running
HTTP 400	Bad request	Check URL for typo/incorrect domain name

3.4.2 How to find logs or diagnostic information

Error logs and diagnostic information within the website are stored within the browser console. To access the console, you would need to enable Developer Tools (enabled by default on Chrome and Firefox, but not Safari).

Chrome/Firefox:

- Right-click anywhere on the page and click Inspect on the menu that comes up.

Safari:

- From the menu bar, choose Safari.
- Select Settings.
- Go to the Advanced page.
- Check the Show features for web developer’s checkbox.
- After doing this, right-click anywhere on the page, and click Inspect on the menu that comes up.

Section 4: System Maintenance

4.1: How the System Can Be Maintained

To sustain the capability of the system to provide a reliable and high-performance service, regular system maintenance is needed. Regular database maintenance, code maintenance and dependency management are essential for system maintenance.

Database Maintenance:

Regular Database Backup:

Implement automated backup schedules to prevent data loss and ensure quick recovery in case of Failures.

Data Integrity Checks:

Perform routine validation of stored data to detect and correct inconsistencies.

Code Maintenance:

Bug Fixes and Patches

Continuously monitor for potential issues and apply patches promptly to ensure security and maintain optimal functionality.

Code Enhancements:

Regularly reviewing code and adding comments improves readability and maintainability, making it easier to implement new features in the future.

Dependency Management:

Package Updates:

Regularly update libraries and frameworks since many vulnerabilities arise due to outdated software

Compatibility Checks:

When updating libraries and frameworks, compatibility checks are crucial to prevent conflicts that may break the application or introduce security risks.

4.2: How Can Possible Future Development Be Implemented

In this section, we focus on how the system's future development can be implemented to enhance usability, efficiency, and accessibility.

Cloud-Based Infrastructure

Cloud platforms provide load balancing and failover mechanisms which can ensure system stability even in case of hardware failures. Cloud platforms can provide multiple servers and distribute incoming traffic across multiple servers to prevent single server overloading. Also cloud platforms will split large databases into smaller parts, allowing faster processing and parallel execution of queries.

Mobile App

Integrating a mobile app into the system can significantly improve user accessibility since users can update progress and check reports anytime, anywhere. Also, users can input data even without internet access, which syncs automatically when reconnected.

Inclusivity and Accessibility

Currently our system has limited inclusivity and accessibility features, which may create barriers for some users. Supporting multiple languages should improve the usability for diverse users. Also adding features such as text-to-speech, high-contrast modes, and keyboard navigation can make the system more inclusive.

Performance Optimization

Implementing lazy loading in React can improve the speed and performance of the website since data is only rendered when visited. Lazy loading helps to load the web page quickly and presents the limited content to the user that is needed.

User Information Security

Currently, all information is stored in plaintext within the system, which is not secure since an attacker can view and use all stored passwords directly. Since hashing is a one-way transformation method, using hashing tools like bcrypt to hash passwords before storing them makes the passwords unreadable and irreversible. Additionally, for other sensitive data like emails and targets, encryption is a good option to protect this information.

Customization and Personalization

In future, users should be able to customize dashboard, layout and the font style based on their need. Also, notification should be customizable, allowing users to choose their preferred communication channel like in app or email and frequencies.

4.3: Potential Ethical and Societal Impacts

Finally, we will discuss the potential ethical and societal impacts of our system for both its current and future status.

Ethical Impacts

Data Security:

Our system stores personal data, as well as data about targets. Ethical concerns arise regarding how data is stored, shared, and protected. Ensuring sensitive data is not misused or compromised is crucial.

Environmental:

Our system currently stores database information in local JSON files. Scaling the system up requires use of larger and more resources. Environmental concerns arise with how much energy these resources would take.

Transparency and Accountability:

Our system must be transparent and allow users and relevant stakeholders to understand how progress is measured. Ensuring transparency mitigates the mistrust or manipulation of data, undermining efforts to address climate change effectively.

Societal Impacts

Public Awareness and Engagement:

Our system has the potential to raise awareness to the public and encourage climate action. Seeing visual progress on the CERP targets may motivate individuals and stakeholders to continue to, or take more, action.

Policy Development:

Our system could influence policy decisions by providing data that informs the Council’s actions. It could help create more effective policies for reducing emissions but also holds the council accountable for meeting climate targets, leading to more proactive climate policies.

Technological Dependence and Digital Divide:

Our system requires technology to access, and whilst tracking online makes productivity more efficient, we must consider communities that lack technology. Those who don't have the means to engage with such systems may be left out of important climate conversations, potentially exacerbating the digital divide.

Staff Productivity and Morale Enhancement:

A well-structured system ensures that staff understand their responsibilities while minimizing redundant work. Clear progress toward climate goals also can boost morale, making employees feel that their works are meaningful and impactful. This will help Durham County Council achieve its net-zero targets more efficiently.