

Documentation

1. Introduction:

This document is to demonstrate my implementation of the “**Greenplan**” assignment test for the student assistant IT job position. The test is to create a web application similar to the company platform that helps clients in monitoring their logistics data such as the list of vehicles and the different trips made.

2. Requirements:

- The application is developed in React.js and CSS for styling. To test the code in the development environment, you need to access the project directory in any terminal and run the following code:

```
npm install
```

This command is going to download automatically all the dependencies needed for the project to run it like **react**, **react-router-dom** (this dependency is responsible for the page routing and redirection) and also **react-map-gl** (map integration with react in the trips section).

Next, to start the application just type the following command:

```
npm start
```

- In case you don't have experience with terminals and how to run a project in development mode, you can deploy the project in any hosting platform (ex: netlify.com). In order to do that, you need to create a new project on the platform and upload the build folder that you can find in the directory to your newly created website. This will serve the website and you can visit it through the link provided by the hosting company. I already deployed did, here is the direct link: <https://greenplan-assignment.netlify.app>
- If you want to serve the project directly from your machine, you will need a web server. Node.js is an option and can be installed from their official website: <https://nodejs.org/en/>.

The next step is to download the project folder from the GitHub repository and especially it should contain the build folder. If you can't find it within the folder, then you need to run the following commands:

```
npm install
```

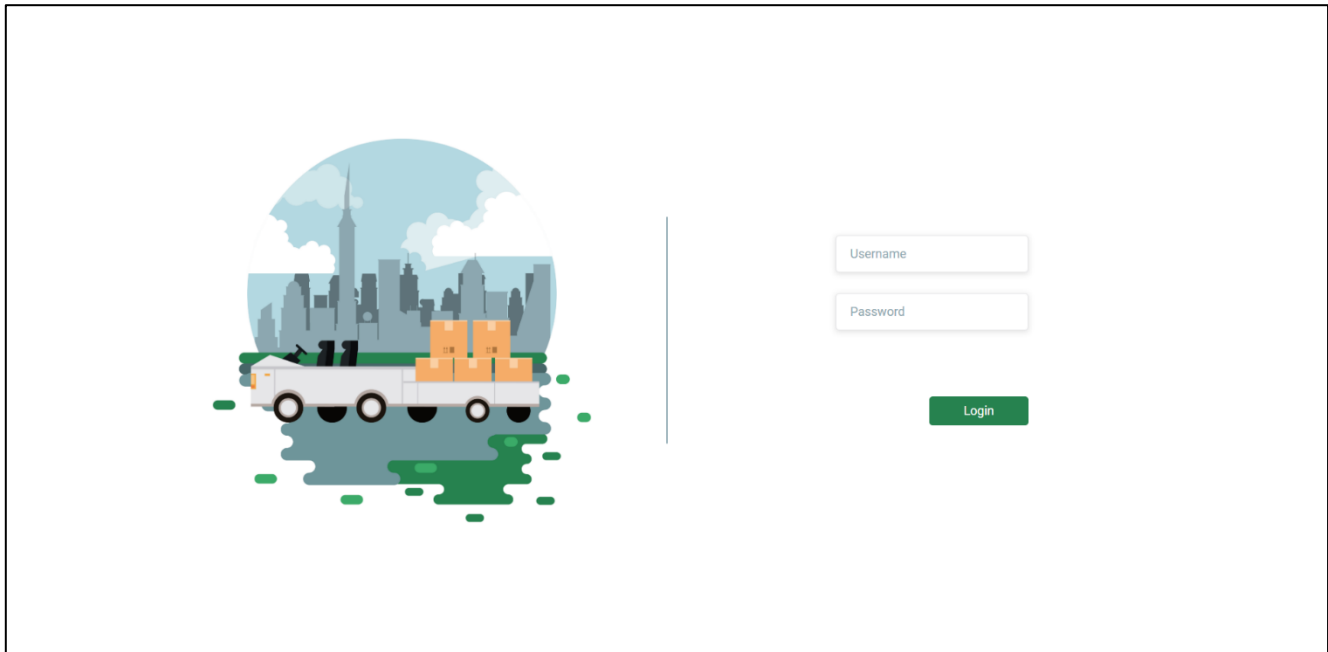
```
npm run build
```

Now the application is ready, the next command will serve the build file and we can access it by visiting the URL visible in the command line:

```
npm serve
```

3. User interface:

The application has basically two pages, the first one is the login page:





To login successfully, just type **"Test"** as username and **"test1234"** for the password.

After entering the correct credentials, the application will redirect the user to the dashboard. This one contains two tabs, the first one displays all the available vehicles with its main information:

Vehicles		Trips		
Name	Length	Width	Height	Cost
Truck	100	50	30	10
Van	70	25	15	8

The "Trips" tab lists all the costumer trips, with some important information like the start point and destination, status and also a map that contains the two locations. The extra information and the map are only displayed when you click on the specific table row, to hide it again just give the row another click.

Vehicles		Trips	
Name	Delivery Count	Date	Status
Trip 1	20	25 September 2022	Done
Trip 1 Start: Bonn Trip 1 End: Cologne Total Trip 1 Time: 3 Hours Total Parcel Delivered: 20 Total Weight of the Parcels: 200 Kgs			
Trip 2	100	26 September 2022	Failed
Trip 2 Start: Frankfurt Trip 2 End: Berlin Total Trip 2 Time: 10 Hours Total Parcel Delivered: 20 Total Weight of the Parcels: 200 Kgs			

4. Implementation:

a. Login page:

The page uses a simple form with two inputs that are marked as required, with an event listener on submission (clicking on the login button).

To mimic a database behavior, I used a simple **json** file to store the user credentials and to check them on login action. Simply, we import the users list from the file, compare if the username entered by the client exists in our list. If it's not then we the application displays an error "user doesn't exist".

If the username already exists, then we move forward and check the password, if it's wrong then we get the following error "Wrong credentials", else the application will direct the client to the dashboard. The error message is displayed and then vanished after some time using **setTimeout**.

b. Dashboard page:

As this page have two tabs, I used **useState** to store a specific value and whenever we click on a tab, the **useState** value will change to the tab number. The content of the page is dynamically changing in relation to the value of the **useState** variable.

In order to render everything dynamically, I used the same concept with the login page here, where I stored the vehicles and trips data in a **json** file and import them in the appropriate file (because there is a component for the vehicles section and another one for the trips section). Once they are imported, we can loop over them because it's an array.

The tried to use a table element to display the data, but couldn't find a way to make the presented idea of displaying more information when clicking on a specific row in trips section. In order to realize this, I just use div elements and tried to make it similar to a normal table but using flexbox. The page structure is divided to multiple components. First, we have the content to show depends on the current variable value. If it's equal to 1 then we should display the **VehiclesTab** components, else it's **TripsTab** component. These both components use another component called Table that is responsible for displaying the data in the table style.

It takes "head" (for the table header row) and "body" (actual data) as parameter while "type" argument is used for some conditional code execution depending on the data type we are working with. The Table component is responsible for organizing all the data in a nice style with taking in count some actions to do depending on the data passed like in the trips tab where every row is clickable and when you click on it, you get to see more information and the trip map.

For the clickable row, I just used a hidden element for every row and styled it with height equals to zero and whenever the row is clicked, we change the hidden part height to the children actual height.

For the map functionality, I used an **npm** package called "**react-map-gl**". To use it, we need to import the Map component, pass some parameters like the access token, the desired map style and also an initial view with some coordinated.

5. Conclusion:

In conclusion, this implementation documentation provides a comprehensive guide for building and deploying a React web application. It covers all the essential elements, from the requirements to the features and implementation. The aim is to make the development process as smooth and efficient as possible, by providing clear and concise information on all aspects of the application. We hope that this documentation serves as a valuable resource for developers and non-developer users.