

PDF CONVERTER

Setup Process

Installation of Dependencies

- First, I installed Node.js on my computer by downloading it from the Node.js website.
- Then, I navigated to the root directory of the project in my terminal and ran the command `npm install`. This downloaded and installed all the required packages, including React, Axios, Cors, Express and Puppeteer.
- Imported the required libraries in both frontend and backend folders
- Frontend requires installation of React-app and Axios
- Backend requires installation of Puppeteer, Express, Axios and Cors

Starting the Development Server

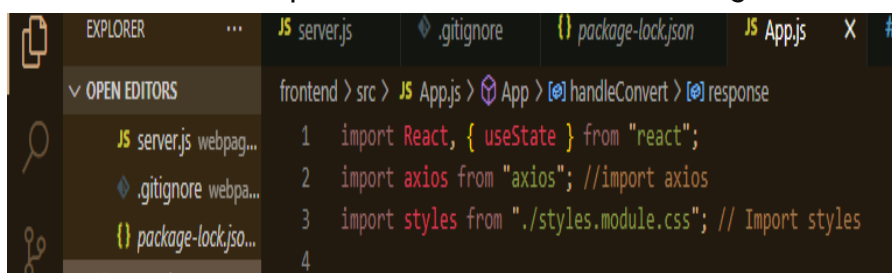
- Once the dependencies were installed, I created frontend and backend files
- Started working on the code for PDF converter
- I used Puppeteer, a NodeJs API in the backend, to convert the webpage to PDF.
- I used Express, Axios and Cors to connect the frontend and backend and run them smoothly.
- I started the development server by running `npm start` in the terminal.
- This launched the application in my default web browser, and I could see the PDF Converter application up and running.

Frontend Configuration

- In the frontend code (typically in `App.js`), I imported Axios to make HTTP POST requests from the client-side application.
- Axios was used to send the URL of the web page to be converted to the backend server, along with handling the response data received from the server.
-

Backend Configuration

- I ensured that I had a backend server running on my local machine to handle the conversion process. Alternatively, I updated the backend URL in the code to match the server address
- Additionally, I configured Cross-Origin Resource Sharing (CORS) on the backend to allow requests from the frontend running on a different origin.



The screenshot shows a code editor with a sidebar on the left containing 'EXPLORER' and 'OPEN EDITORS'. The 'EXPLORER' sidebar shows a file tree with 'server.js', '.gitignore', 'package-lock.json', and 'App.js'. The 'OPEN EDITORS' sidebar shows the same files. The main editor area displays the content of 'App.js' with the following code:

```
1 import React, { useState } from "react";
2 import axios from "axios"; //import axios
3 import styles from "./styles.module.css"; // Import styles
4
```

```
JS server.js X .gitignore {} package-lock.json JS App.js # styles.mod
webpage-to-pdf > JS server.js > ...
1  const express = require("express");
2  const puppeteer = require("puppeteer");
3  const cors = require("cors");
4
5  const app = express();
6  const port = 3001;
7
8  app.use(express.json());
9  app.use(
10     cors({
11       origin: "http://localhost:3000",
12     })
13   );
14
```

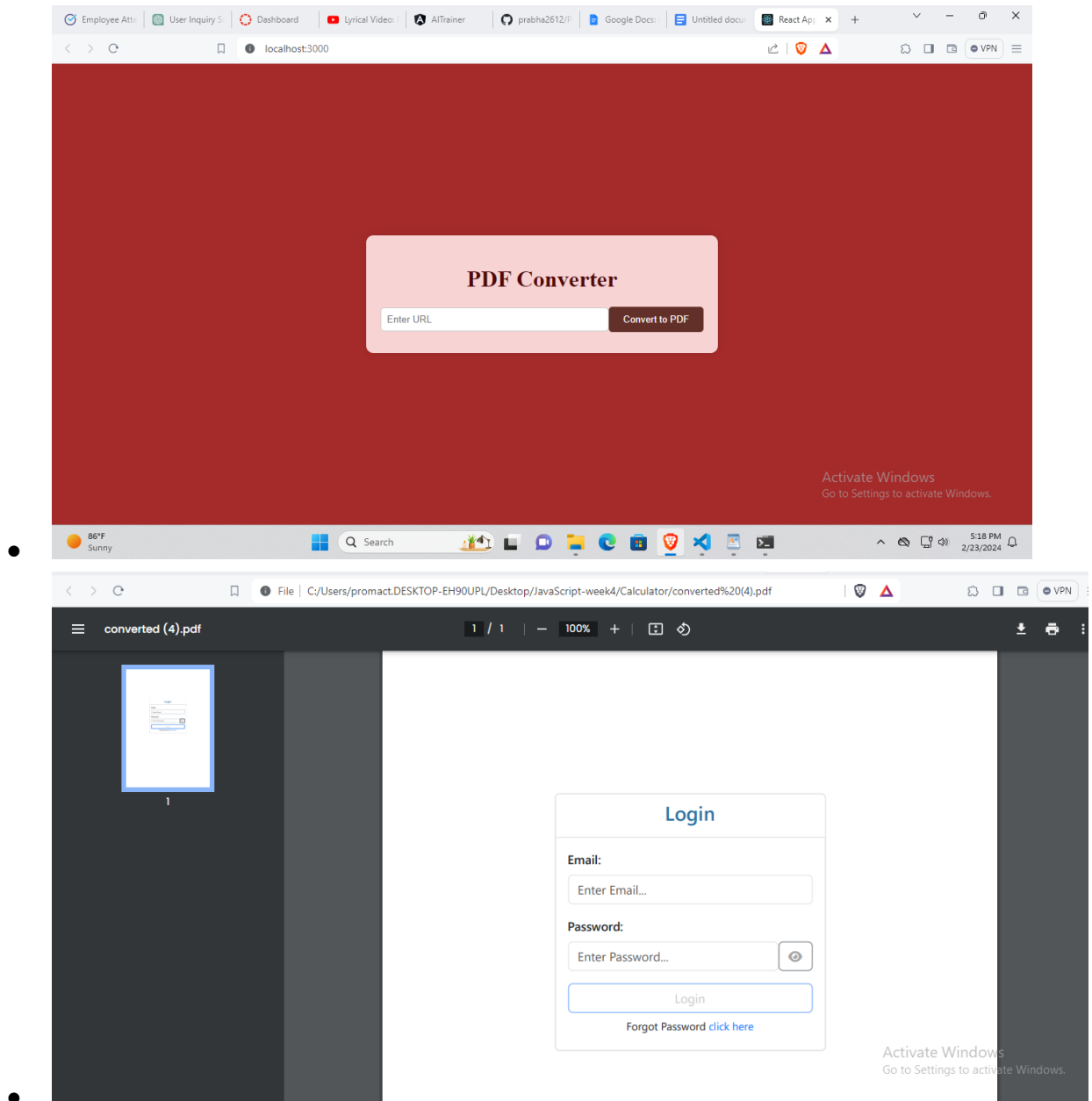
Cross-Origin Resource Sharing (CORS)

- I configured CORS middleware in the Express backend to allow requests from the frontend application running on a different origin (`http://localhost:3000`).
- This ensured seamless communication between the frontend and backend servers without encountering CORS policy errors.
- Code snippet:

```
const cors = require("cors");
app.use(
  cors({
    origin: "http://localhost:3000",
  })
);
```

Testing

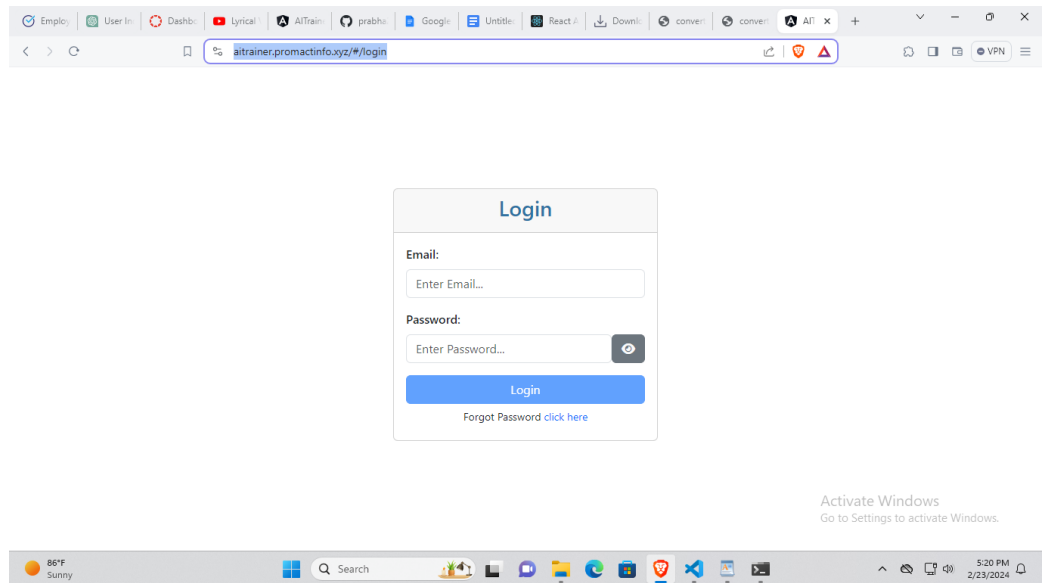
- With the application running, I tested its functionality by opening my web browser and navigating to the URL where the application was running (`http://localhost:3000`).
- Following the steps outlined in the user guide below, I successfully imported a page's content as a PDF.



User Guide

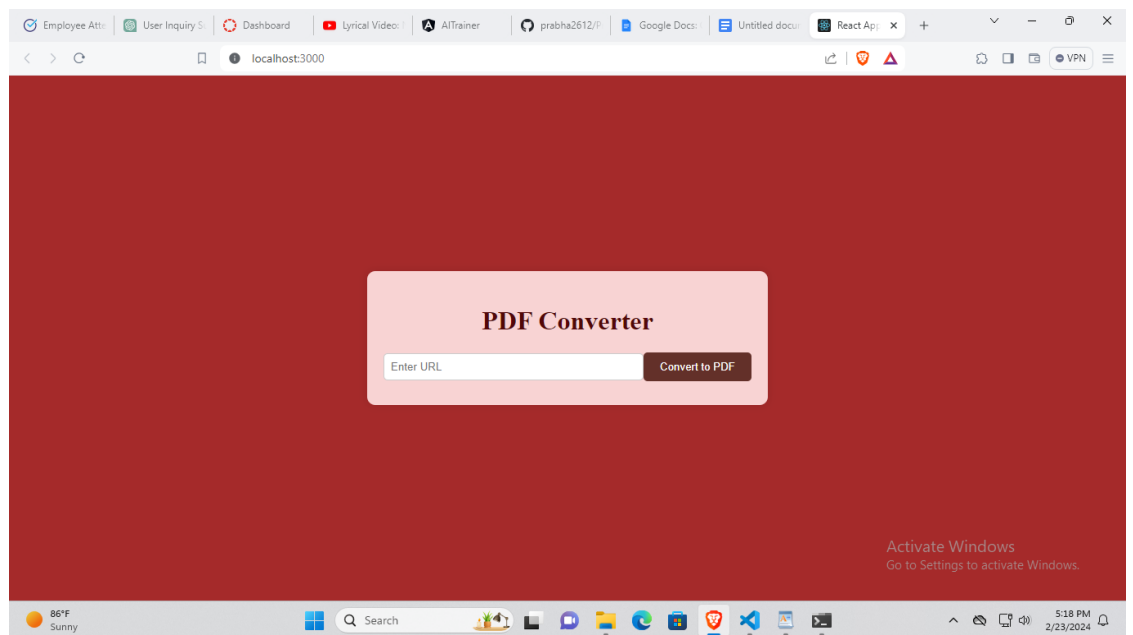
Entering the URL

- The user starts by navigating to the desired web page that they want to convert into a PDF document.
- Once on the webpage, they have to copy the URL from the address bar of their web browser.



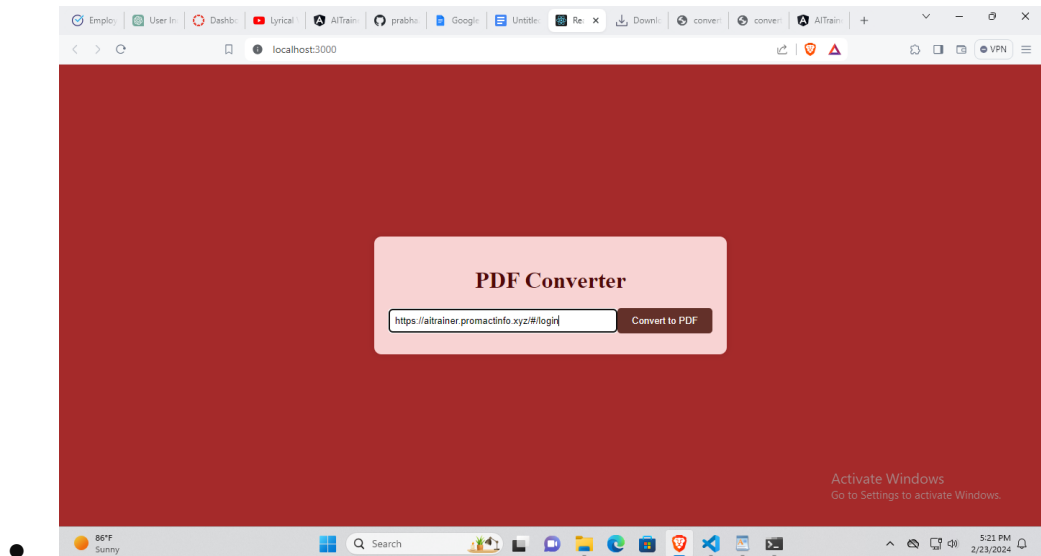
Opening the Application

- The user opens their web browser and goes to the URL where the PDF Converter application is running (<http://localhost:3000>).



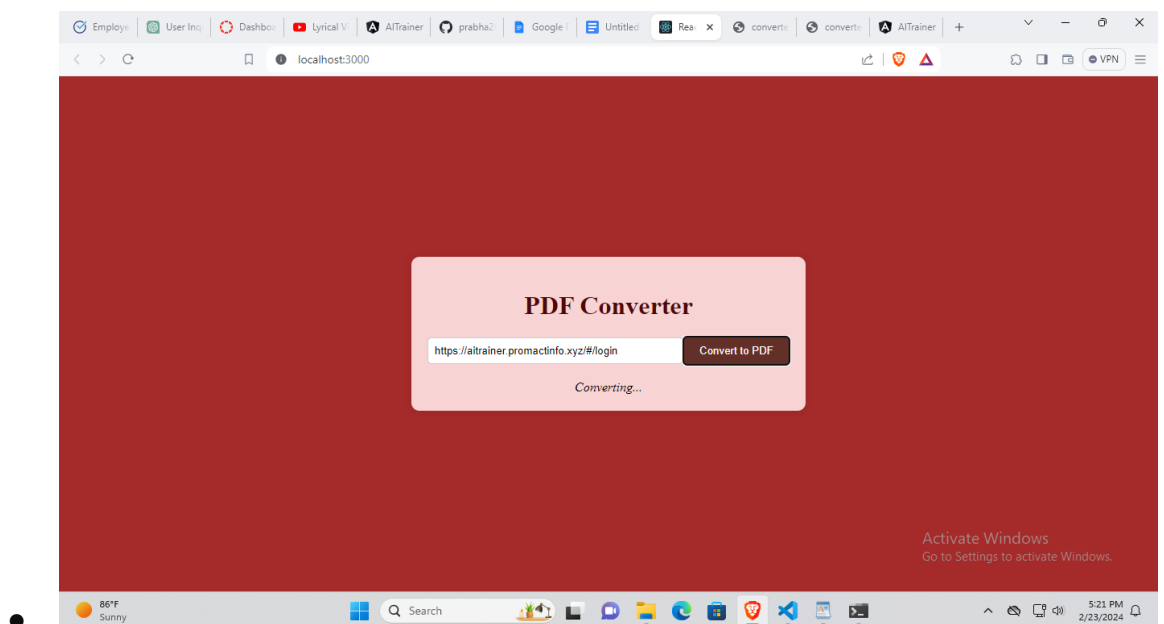
Inputting the URL

- In the PDF Converter application, the user finds an input field labelled "Enter URL."
- They paste the copied URL into this input field



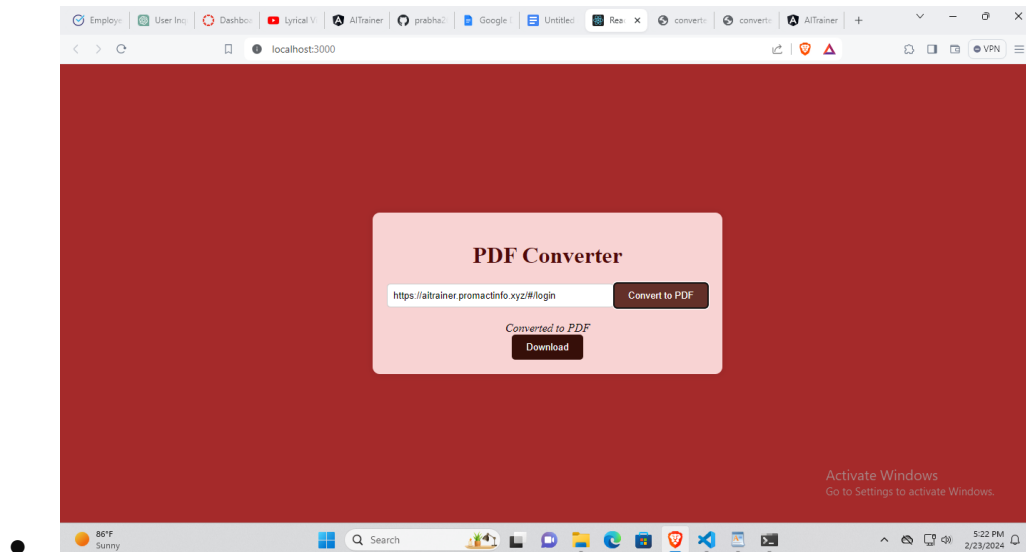
Converting to PDF

- After entering the URL, the user clicks the "Convert to PDF" button.
- The application initiates the conversion process, and the user can see a status message indicating the progress, such as "Converting..." or "Conversion failed."



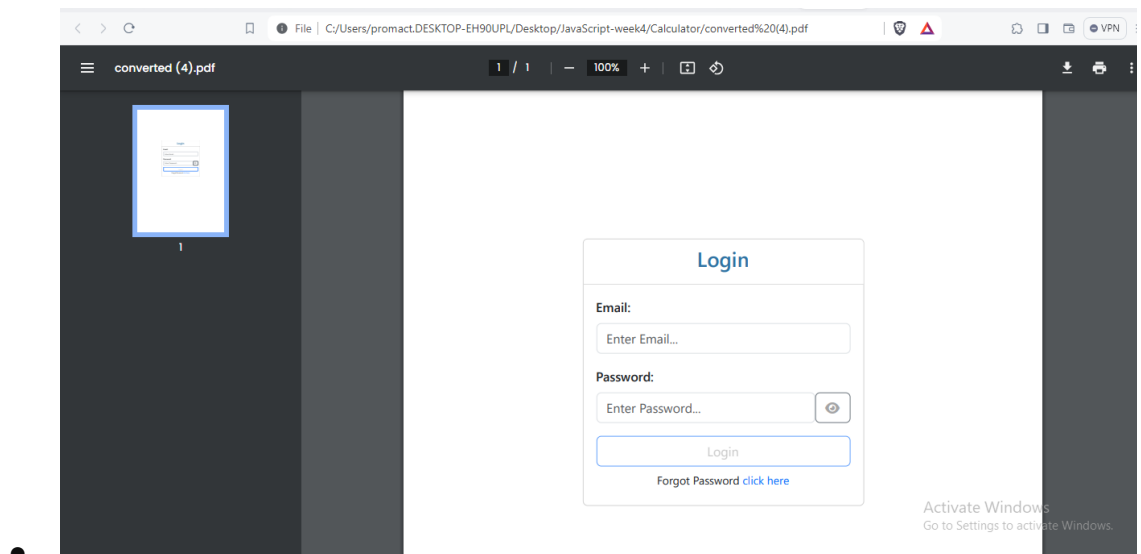
Downloading the PDF

- Once the conversion is complete, a "Download" button appears.
- Clicking this button initiates the download of the converted PDF file to the user's local device.



Viewing the PDF

- The user locates the downloaded PDF file on their computer and opens it using a PDF viewer application.
- The PDF file contains the content of the webpage the user converted, including text, images, and layout, all accurately preserved in PDF format.



That's it! The user has successfully used the PDF Converter feature to import a webpage's content as a PDF document. They can now enjoy the convenience of saving information offline or sharing it with others in PDF format.