Genletter

Project Link : https://github.com/Dhruvk7/Genletter

Dhruv Khera
Delhi Technological University
https://www.linkedin.com/in/dhruv-khera-aab240224/

Genletter is a MERN stack website where an authenticated (Google OAuth) user can generate any type of letter whether it can be an offer letter, internship letter etc. They can also generate a single and multi-column ATS friendly resume for them, and can edit those in future.

BACKEND

index.js

- dotenv: This module loads environment variables from a .env file into process.env.
- express: Express.js, or simply Express, is a back-end web application framework for Node.js.
- cors: This is a Node.js package for providing a middleware that can be used to enable CORS with various options.
- passport: Passport.js is a middleware that is extremely flexible and modular, used for authenticating requests.
- cookie-session: It's a middleware that allows you to manage sessions using cookies.
- body-parser: Body-parser is a middleware that helps in parsing incoming request bodies in a middleware before the handlers.
- authRoute, resumeRoute, offerLetterRoute, getRoutes: These are custom route handlers. Their definitions are in their respective files in the routes folder.
- port: This is the port on which the server runs. If the PORT environment variable isn't defined, it defaults to 5000.
- connectDB: This function connects the app to the database.

passport.js

The passport.use method sets up the Google OAuth 2.0 Strategy. It uses the following Google API details:

- clientID: A unique identifier from your Google API console for your app.
- clientSecret: The client secret tied to the clientID, also from the Google API console.
- callbackURL: The route in your application to which users will be redirected after they authenticate with Google.
- scope: The data your application requests access to.

The strategy's verify callback handles what happens when a user is authenticated with Google:

- The user's profile is logged to the console.
- The user's email is stored in the email variable.
- It checks if the user exists in the database by Google id. If not, a new user is created and saved in the database.
- The callback function is then called, passing null and the user's profile.

Serialization and Deserialization

- The passport.serializeUser method defines how the user object should be stored in the session. In this case, the entire user object is stored.
- The passport.deserializeUser method defines how to retrieve the user object from the session. In this case, the entire user object is returned.

User Email Retrieval

- The getUserEmail function is exported from this module. This function returns a Promise that resolves with the authenticated user's email. This can be used elsewhere in your application to access the user's email after they have been authenticated.

auth.js

- /login/success : This route is hit after successful authentication. It checks if the user object is attached to the req (request) object, a step usually done after successful authentication. If the user exists, a success message with the user details is sent. If not, a "Not Authorized" error message is returned.
- /login/failed : This route is used when authentication fails. It simply returns a 401 Unauthorized HTTP status code along with an error message.
- /google and /google/callback : These routes facilitate Google OAuth authentication. The first triggers the authentication process when visited. The second is the callback URL that Google redirects to after the user either consents or declines the authentication request. On successful authentication, the user is redirected to the URL stored in the CLIENT_URL environment variable. On failure, they are redirected to the "/login/failed" route.
- /logout : This route is used to log the user out. It calls the logout method on the req object, effectively removing the login session, then redirects the user to the URL stored in CLIENT_URL.

getRoutes.js

- GET /resume/:id: Fetches a resume with a specific id from the database. It uses Resume.findOne() method to find the document and then constructs an object obj with the found resume data before sending it as a JSON response. If there is no resume found with the given id, it sends a response 'No data found!'.
- GET /offer_letter/:id: Similar to the first route, this one fetches an offer letter with a specific id from the database using OfferLetter.findOne() method. It constructs an object obj with the found offer letter data before sending it as a JSON response. If there is no offer letter found with the given id, it sends a response 'No data found!'.

Each route uses mongoose.Types.ObjectId(id) to convert the id from the request parameters into a MongoDB ObjectId before using it to search in the database. This is necessary because MongoDB uses ObjectIds for its _id field.

offerLetterDataHandler.js

- POST '/' : This route handles HTTP POST requests to the root URL (/). The purpose of this route is to create or update an offer letter based on the request body.

  The function first calls passport.getUserEmail() to get the currently logged-in user's email address. This function is asynchronous, which is why the await keyword is used.

  Then, it extracts information from the request body, including:

  - id: The id of the offer letter. This is used to identify an existing offer letter that should be updated. If there is no id, a new offer letter is created.
  - candidateDetails: Information about the candidate.
  - companyDetails: Information about the company making the offer.
  - jobDetails: Information about the job being offered.
  - deadline: The deadline for the candidate to respond to the offer.
  - personalDetails: Personal details of the user.
  - dateToday: Today's date.
  - date: The date the offer letter was made.

  The function then creates a new offer letter or updates an existing one, depending on whether an id was provided in the request body. The result of this operation is logged to the console.

- GET '/history' : This route handles HTTP GET requests to the /history URL. The purpose of this route is to retrieve all offer letters associated with the currently logged-in user.

  The function calls passport.getUserEmail() to get the currently logged-in user's email address and then uses this email to query the OfferLetter collection in MongoDB. It sends the retrieved data as a JSON response. If there is an error during this process, it is logged to the console.

resumeDataHandler.js is similar to offerLetterDataHandler.js

FRONTEND

(Only discussion of important files are given)

App.js

- Navbar: Contains the navigation bar displayed on all pages except the login page.
- Landing Page/Home: This is the landing page of the application.
- Our Features: A component highlighting the features of the application.
- Forms: Contains form components used in the application, such as Resume and Offer Letter.
- Templates: This directory holds various templates for documents like resumes and offer letters.
- Login: The component responsible for user authentication.
- Profile: The user profile page, displaying the user's email, their resume and offer letter history.

Key Functions

- App: This function initializes states for user, resume and offer letter, and includes the getUser function to fetch user data and history of resumes and offer letters from the server. It also contains the primary routing setup for the application.
- getUser: This asynchronous function fetches the user data from the server and updates the state of the application. It makes separate GET requests to retrieve user login information, resume history, and offer letter history.

Routing

The project uses react-router for routing. Below are the main routes:

- /login: Takes users to the login page.
- /: Directs users to the home page, which also displays the application's features.
- /profile: Displays the user's profile, resume history and offer letter history. If the user is not authenticated, they are redirected to the login page.
- /edit/resume/:id and /edit/offer_letter/:id: Directs authenticated users to edit their respective documents. If the user is not authenticated, they are redirected to the login page.
- /resume/details and /offer_letter/details: Displays the details of the respective documents to authenticated users. Unauthenticated users are redirected to the login page.
- /resume/templates and /offer_letter/templates/1: Displays templates for creating the respective documents to authenticated users. Unauthenticated users are redirected to the login page.

Resume.js

Imports

Resume.js imports several React dependencies, including the useState and useEffect hooks, as well as several components that represent different parts of a resume:

- PersonalDetails, Links, EducationalDetails, Projects, WorkExperience, Achievements, Skills, POR (Positions of Responsibility), and FinalPage.

It also imports useNavigate and useParams from react-router-dom for navigation and retrieving route parameters, respectively. axios is imported for making HTTP requests.

The Resume Component

The main functionality is encapsulated within the Resume function component. This component maintains the resume state and manages the rendering of various sections of the resume.

The resume data structure is held in state with the useState hook. The setRefs function is used to update this state. In addition to personal details, the resume state includes links, education, work experience, projects, achievements, skills, and positions of responsibility.

UseEffect Hook

The useEffect hook is used to fetch the resume data for a given ID, if available, when the component is first rendered. If the resume ID exists, an HTTP GET request is made to the endpoint http://localhost:5000/edit/resume/${id}. If the request is successful, the response data is used to update the refs state.

Handle Change Functions

There are several functions for handling updates to the resume fields:

- handleChange: This is used to update simple fields (those that are not arrays) in the state.
- handleArrayChange: This is used to update fields in the state that are arrays.
- handleLinkChange: This handles changes in link fields.
- handleSkillChange: This handles changes in the skills fields.

Handler Functions

There are also handler functions for adding or removing fields from arrays in the state: fieldAddHandler, fieldArrayAddHandler, fieldRemoveHandler, skillFieldAddHandler, skillFieldRemoveHandler, linkIncreaseHandler, linkDecreaseHandler.

Navigation Functions

The prevStep and nextStep functions are used to navigate between different sections of the resume.

Submit Function

The handleSubmit function is called when the form is submitted. It constructs an object containing the resume data from the state, resets the state, navigates to the resume template page, and sends a POST request to http://localhost:5000/resume with the resume data.

Rendering

The content to be rendered is determined by the index state variable, which represents the current step in the resume building process. The switch statement checks the current value of the index and sets the content variable to the appropriate component. Each component is passed its relevant props, including a function for handling changes to its fields and functions for navigating to the previous and next steps.

handleArrayChange(input, e)

This function is used to update the values of the fields in the objects of the array fields in the refs state variable.

handleLinkChange(input, e)

This function is used to handle the changes in the fields of the objects in the links array, within the array fields of refs state variable.

handleSkillChange(input, e)

This function is used to handle the changes in the fields of the objects in the skills field of refs state variable.

fieldAddHandler(name)

This function is used to add a new object to the array fields of refs state variable.

fieldArrayAddHandler(name)

This function is used to add a new object to the array fields of refs state variable, which themselves are arrays.

fieldRemoveHandler(name)

This function is used to remove the last object of the array fields in the refs state variable.

skillFieldAddHandler(input)

This function is used to add a new object to the skills field of refs state variable.

skillFieldRemoveHandler(input)

This function is used to remove the last object from the skills field of refs state variable.

linkIncreaseHandler(link)

This function is used to add a new object to the links array within an array field in the refs state variable.

linkDecreaseHandler(link)

This function is used to remove the last object from the links array within an array field in the refs state variable.

prevStep()

This function is used to decrease the value of the index state variable.

extStep()

This function is used to increase the value of the index state variable.

formatDate()

This function is used to get the current date in a particular format.

handleSubmit(event)

This function is used to handle the submission of the form, by resetting the refs state variable, navigating to another route and sending the data to a server.

Depending on the value of the index state variable, different components are rendered which provide the UI for the different sections of the form.

The main functionality of the Resume component is to provide the user interface for creating a resume, by filling in a series of forms, and to handle the form submission.

Components:

1. PersonalDetails: This component provides the UI for entering personal details like First name, Last name, Phone, Email and Roll number.
2. Links: This component provides the UI for entering Profile links.
3. EducationalDetails: This component provides the UI for entering Educational details like Degree, Course, University, Year and Grade.
4. WorkExperience: This component provides the UI for entering Work Experience details like Title, Organisation, Duration and Contributions.
5. Projects: This component provides the UI for entering Project details like Name, Description and Completion status.
6. Achievements: This component provides the UI for entering Achievement details.
7. Skills: This component provides the UI for entering Skills details.
8. POR: This component provides the UI for entering Positions of Responsibility details.
9. FinalPage: This component is used to display a final page when all the forms have been filled.

Resume2.js

This is a functional component that renders a styled resume.

useLocation from the react-router-dom library is used to retrieve the location state, which includes all the props for the Resume.

html2canvas and **jspdf** libraries are used to generate a PDF from the Resume that's displayed in the browser.

Different sections of the resume such as Education, Work Experience, Academic Projects, Technical Skills, Positions of Responsibility and Academic Achievements and Awards are created based on the props passed.

The resume can be downloaded as a PDF by clicking on the button with onClick={downloadHandler}.

Functions

- educationHandler: Combines the degree and course into a single string, separated by a comma.
- porHandler: Combines the position and organization into a single string, separated by "at".
- downloadHandler: An async function that generates a canvas from the #resume element on the page, then generates a PDF from that canvas using the jspdf library.

Libraries Used

- html2canvas: For rendering the Resume component as a canvas, which can then be turned into a PDF.
- jspdf: For creating the PDF from the canvas created by html2canvas.
- react-external-link: For rendering external links.

OfferLetter components are similar to resume