INDIAN INSTITUTE OF TECHNOLOGY KHARAGPUR

SUMMER INTERNSHIP REPORT

TOPIC - FRONTEND ANGULAR DEVELOPMENT (Reviewprobe.com)

COMPANY - COMPUTER SOFTWARE SOLUTIONS LLC

DURATION : 15th May, 2023 – 15th July, 2023

NAME - SAIKAT MOI

ROLL NO - 20CS10050

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Key Contributions

During my internship, I served as a Frontend Angular Developer, where I actively contributed to the development of dynamic web applications, gaining valuable experience in Angular and front-end technologies.

- **Development of ReviewProbe.com:**
  I played an integral role in creating the dynamic job search and posting website, ReviewProbe.com. My work significantly contributed to enhancing the user experience on the platform.
- **Technologies and Tools:**
  I utilized a variety of technologies and tools, including Angular, TypeScript, HTML, CSS, Bootstrap, and Firebase, to build and improve the website.
- **API Integration:**
  I was responsible for implementing API calls, ensuring seamless data communication between the front-end and back-end systems. This helped to provide users with up-to-date job listings and information.
- **Performance Optimization:**
  I worked on optimizing the website's performance through state management, enhancing load times, and making it more responsive to user interactions.
- **Efficient Data Fetching:**
  I designed and implemented efficient data fetching mechanisms across multiple components, ensuring that the website could handle large volumes of data without compromising user experience.
- **Real-time Chat Functionality:**
  I incorporated Stomp.js to enable real-time chat functionality on ReviewProbe.com. This feature enhanced user engagement and interaction on the platform.

Various Technologies Utilized in this Project

Angular Features Used:

- **Directives:**
  Angular directives were employed to extend HTML with custom behaviors. Structural directives, such as ****ngFor***, dynamically generated HTML elements based on data collections, allowing for the dynamic rendering of job listings and other content. Attribute directives, like ***[ngClass]***, were used to conditionally apply styles to elements, enhancing the user interface.

- **Data Binding:**
  Data binding in Angular was a fundamental part of creating a responsive user interface. One-way data binding through property binding ***([property])*** was used to display data from the component to the view. Two-way data binding ***([(ngModel)])*** enabled real-time synchronization of user inputs with component data.

- **Child Components:**
  The implementation of child components allowed for the modularization of the application. For example, individual components were created for job listings, user profiles, and chat rooms. These components were then integrated into the main application, promoting code reusability and maintainability.

- **Input and Output Decorators:**
  The ***@Input*** decorator was applied to allow parent components to pass data to child components. For example, when navigating to a user's profile, the user's ID was passed to the user profile component. The ***@Output*** decorator facilitated the communication of user interactions from child components to parent components. For instance, when a user initiated a chat, the chat component emitted an event to inform the parent component.

- **Lifecycle Hooks:**
  Angular's lifecycle hooks were crucial for managing component behavior. ***ngOnInit*** was used to perform initialization tasks, such as fetching initial data. ***ngOnChanges*** was utilized to respond to changes in input properties. ***ngOnDestroy*** was employed to release resources or perform cleanup when components were destroyed.

- **Services and Dependency Injection:**
  Angular services were created to encapsulate business logic, manage data retrieval, and provide shared functionality. Dependency injection ensured that components had access to these services. For instance, a JobService handled the retrieval of job data and provided it to various components.

- **Observables:**
  Observables from the ***RxJS*** library were used to handle asynchronous operations effectively. For instance, when fetching job listings or enabling real-time chat functionality, observables provided a powerful mechanism for working with asynchronous data streams.

- **Routing:**
  Angular's routing module was configured to define the navigation flow of the web

application. Route definitions allowed users to access different sections of the website through URL routes. Guards were implemented to control access to certain routes and ensure security and data privacy.

- **Angular Forms:**
  Angular forms played a crucial role in capturing user inputs and enabling interactions. Template-driven forms were employed for simpler forms, such as user registration. More complex forms, such as search filters, utilized reactive forms, offering advanced validation and data manipulation capabilities.

TypeScript Features Used:

- **Strong Typing:**
  TypeScript's static typing enhanced code reliability and maintainability by providing a clear structure for variables, functions, and classes.
- **Interfaces:**
  TypeScript interfaces ensured data consistency by defining data structures such as job listings, user profiles, and chat messages.
- **Classes:**
  TypeScript classes facilitated object-oriented programming, allowing for the creation of components, services, and models, thus improving code organization and reusability.
- **Generics:**
  Generics were used, especially with observables, to create flexible and reusable components that could work with different data types.
- **Enums:**
  Enums represented named constants, improving code readability and maintenance, such as for user roles and job categories.
- **Decorators:**
  TypeScript decorators were applied for different purposes, including @Input and @Output for communication between parent and child components, and @Component and @NgModule for defining component and module metadata.
- **Modules and Dependency Injection:**
  TypeScript modules and dependency injection facilitated project organization by separating concerns into feature modules and services.

CSS Libraries/Frameworks Used:

- **Angular Material:**
  Angular Material UI was used to create a consistent and visually appealing user interface. It provided a wide range of pre-built, Material Design-inspired components, such as buttons, cards, dialogs, and input elements, simplifying the development of a modern and user-friendly application.

- **Bootstrap:**
  Bootstrap provided a foundation for a consistent and responsive design, offering pre-designed components and elements such as navigation bars, buttons, and forms.

- **Custom CSS:**
  Custom CSS was used to tailor the visual design to specific project needs, ensuring a unique and polished appearance.

- **Font Awesome:**
  Font Awesome icons were applied to enhance the user interface, offering a wide variety of icons for buttons, navigation elements, and UI components.

- **CSS Preprocessors:**
  CSS preprocessors like Sass or Less may have been used to enhance code maintainability with features like variables, nesting, and mixins.

- **Responsive Design:**
  Media queries were implemented for responsive design, ensuring the application's adaptability to various screen sizes and devices.

Angular Routing Configuration and Website Flow

Description

I played a key role in configuring the Angular routing for our web application. This included designing the overall flow of the website and implementing the passing of route parameters

Key Strategies and Implementations:

- **Routing Configuration:**
  Angular provides a powerful and flexible routing system that allows for the creation of single-page applications with dynamic content loading. In our application, the routing was configured in the app-routing.module.ts file. This file served as the central hub for defining the application's routes.

- **Designing the Overall Flow:**
  To design the website's overall flow, we carefully considered the various sections and views within the application. This included components for the homepage, job listings, user profiles, and chat functionality. Each section was mapped to a specific route, creating a structured and intuitive user experience. For instance, the homepage was represented by the empty route, job listings had a dedicated route, user profiles had their routes, and the chat feature had its route as well.

- **Passing Route Parameters:**
  One of the crucial requirements was to pass route parameters, such as job IDs or user IDs, to specific components. This allowed for the dynamic loading of data and personalized user experiences. Angular's routing system provides a straightforward way to pass route parameters. We defined dynamic segments in the route URL using placeholders, for example, ***:jobId*** or ***:userId***. In the component, we accessed these parameters through the ***ActivatedRoute*** service, which provided access to the route's parameters via observables. We subscribed to these observables to retrieve and work with the parameter values.

- **Nested Routes:**
  Our website featured nested routes to create a hierarchical structure. For instance, the user profile route was nested under the /users route, allowing for a more organized URL structure. This was achieved by defining child routes within the parent route configuration, resulting in URLs like ***/users/:userId/profile***.

- **Route Guards:**
  To enhance security and control access to certain routes, we implemented route guards. These guards prevented unauthorized access to specific views or functionalities, such as user profiles or chat rooms.

- **Route Navigation:**
  To enable smooth navigation, we incorporated Angular's routerLink directive in our templates. This allowed users to click links and buttons to navigate to different sections of the application.

Implementing API Calls with *Retrofit-Axios-TS*

Description

During my internship as a Frontend Angular Development Intern I had the opportunity to work extensively on implementing API calls using the Retrofit-Axios-TS library. This library served as a crucial tool for creating a client to interact with our backend API. The code I worked on was pivotal in facilitating communication between the frontend and backend, enabling the retrieval and submission of data.

Key Strategies and Implementations:

- **Import Statements:**
  To set the stage for effective API communication, I began by importing the necessary modules and classes, including the components and data models provided by Retrofit-Axios-TS. These imports laid the foundation for building and managing API requests.

- **ApiService Class:**
  The ApiService class emerged as the central component in our API communication infrastructure. This class extended the BaseService class provided by Retrofit-Axios-TS, and it played a pivotal role in managing the connection between the frontend and backend. It was adorned with the ***@BasePath*** decorator, which allowed us to specify the base URL for our API. The base URL was constructed using the environment.devBackendURL and appended with '/rp-backend', forming the foundation for all subsequent API calls.

- **API Endpoint Methods:**
  The heart of the API communication process lay in the methods defined within the ApiService class. Each method was designed to correspond to a specific API endpoint. These methods were further adorned with HTTP method decorators, such as ***@GET, @POST, @PUT, or @DELETE***, depending on the HTTP request type required by the endpoint. The methods were equipped to specify the URL path and parameters necessary to interact with the respective endpoint.

- **Response Types:**
  In order to streamline data handling, each method was furnished with TypeScript's generic types and the Response type provided by Retrofit-Axios-TS. This allowed us to clearly define the expected response type for each API call. For instance, a method signature like ***Promise<Response<UserType>>*** clearly indicated that the method returned a promise of a response, and the response contained data of type UserType.

- **Path and Body Parameters:**
  For a comprehensive API call setup, the methods featured various decorators to manage path parameters, query parameters, and request bodies. Path parameters were handled using the ***@Path*** decorator, query parameters with the ***@Query*** decorator, and request bodies with the ***@Body*** decorator. These decorators played a critical role in constructing the URL and request payload, ensuring that the HTTP request contained all the necessary information for successful interaction with the API.

Optimizing Performance with *NgRx* for State Management

Description

As part of my role as a Frontend Angular Development Intern, one of the key challenges was to optimize the website's performance and enhance the user experience. To address this, I employed the ***NgRx*** library for state management, which played a pivotal role in ensuring efficient data management, minimizing API calls, and sharing fetched data seamlessly across multiple components.

Key Strategies and Implementations:

- **Store for Data Management:**
  I established a centralized store using *NgRx*, which served as the repository for all the application's data. This store allowed for the organized storage and retrieval of various data types, including job listings, user profiles, and chat messages.

- **State Normalization:**
  To avoid redundant data and reduce storage requirements, I implemented a state normalization strategy. Data was stored in a normalized form within the *NgRx* store, allowing for efficient data retrieval and updates.

- **Caching Fetched Data:**
  A core optimization strategy was the caching of fetched data. Once data was retrieved from the API, it was stored in the *NgRx* store. This cached data was then readily available for use across different components without the need for additional API calls.

- **Lazy Loading and Pagination:**
  I integrated lazy loading and pagination techniques to fetch data incrementally as the user scrolled through listings. This approach minimized the initial data load, resulting in quicker page rendering and a smoother user experience.

- **Sharing Data Across Components:**
  By utilizing the *NgRx* store, I made data accessible to multiple components throughout the application. This allowed different parts of the website, such as job listings, user profiles, and chat modules, to efficiently share and interact with the same data source.

- **Data Consistency and Synchronization:**
  NgRx's state management ensured that data remained consistent across the application, reducing the risk of data discrepancies or conflicts. Real-time updates were handled seamlessly through the store, providing users with the latest information.

- **Selective Data Fetching:**
  To minimize unnecessary data retrieval, I implemented selective fetching mechanisms. Only the required data was fetched on demand, reducing the burden on the server and network resources.

Incorporating *Stomp.js* for Real-Time Chat Functionality

Description

To enhance the real-time communication features of our web application, we integrated Stomp.js, a widely used library for WebSocket communication. Stomp.js allows us to establish WebSocket connections, exchange messages, and achieve seamless real-time chat functionality within our Angular application.

Key Strategies and Implementations:

Incorporating Stomp.js involved the development of a dedicated Angular service named SocketService. This service encapsulates the core WebSocket-related functionality, enabling real-time chat features. Here's a high-level overview of the key components and functionalities within the SocketService:

- **WebSocket Client Initialization:**
  We initiate a WebSocket connection by creating an instance of the Stomp.js client within the SocketService. This client is configured with a broker URL, debug settings, and a predefined reconnect delay.
- **User Connection and Disconnection:**
  Users can establish a connection to the WebSocket server by invoking the connect(userId: string) method. Upon connection, users are automatically registered as active on the WebSocket server. The service also provides a disconnect() method to gracefully terminate the connection, which involves unsubscribing from specific topics and deactivating the client.
- **Real-Time Message Handling:**
  Messages received from the WebSocket server are processed and distributed to the application's components through a Subject<chatMessageModel >. This allows the application to react to new incoming messages in real-time. The handleUnicastMessage method is responsible for handling and forwarding received messages.
- **User Registration and Deregistration:**
  The service enables users to register as active or unregister from the WebSocket server. This registration mechanism is vital for managing the presence and activity status of users in real time.
- **Message Sending:**
  Users can send chat messages to other users by calling the sendMessage(message: chatMessageModel) method. This method verifies the connection status before sending messages.

Challenges Faced and Overcoming Them

During my internship, I encountered several challenges in the development of the ReviewProbe.com website. These challenges tested my problem-solving skills and pushed me to learn and adapt quickly. Here are some of the key challenges and how I overcame them:

- **Complex UI Requirements:** The project involved a complex user interface with various dynamic elements, data-driven components, and responsive design requirements. To overcome this challenge, I broke down the UI into smaller, manageable components and leveraged Angular's modular structure. This allowed for more efficient development and easier troubleshooting.
- **Asynchronous Operations:** Handling asynchronous operations, such as fetching data from APIs and managing real-time chat functionality, presented challenges related to data synchronization and user experience. I overcame this challenge by using RxJS observables to manage asynchronous tasks effectively, ensuring smooth data flow and real-time updates.
- **Collaborative Development:** Working within a team environment required effective communication, code version control, and code integration. Regular team meetings and code reviews helped address this challenge by facilitating communication and ensuring that code changes were well-documented and compatible.
- **Performance Optimization:** Ensuring a fast and responsive web application was essential. I optimized performance by implementing techniques like lazy loading of modules, minimizing unnecessary API requests, and optimizing the use of Angular's change detection mechanisms.
- **Learning Curve:** The project introduced me to new technologies and tools, such as Angular Material UI and observables. Overcoming the learning curve was achieved through self-guided learning, mentor support, and hands-on experience. I dedicated time to explore these tools and gradually integrated them into the project.
- **Security and Data Privacy:** As the project involved user data, ensuring security and data privacy was a priority. I implemented secure authentication and authorization mechanisms and validated user inputs to mitigate security risks.

In summary, my internship experience was marked by various challenges, each of which presented an opportunity for growth and learning. Through effective problem-solving, collaboration with the team, and dedicated efforts, I was able to overcome these challenges and gain valuable skills in web development and project management. These experiences have not only enriched my technical capabilities but have also prepared me for future challenges and opportunities in the field of web development.

Conclusion

My internship experience was deeply collaborative. I had the privilege of working closely with a diverse team of professionals, where regular interactions with colleagues, team discussions, and code reviews promoted effective teamwork. This collaborative environment not only improved my technical skills but also enhanced my ability to communicate and collaborate with others in a fast-paced development environment. In conclusion, my internship at **Computer Software Solutions LLC** has been a transformative experience that has equipped me with a diverse skill set and the confidence to tackle complex web development projects. I am grateful for the support and guidance of my mentors and the trust of the organization in entrusting me with critical responsibilities. This internship has been instrumental in preparing me for a successful career as a Frontend Developer. I am excited to apply the knowledge and skills I have gained to future projects and contribute effectively to the ever-evolving field of web development.