Car Rescue App Report

Abrar Galib Zaman (1083789), Mousa Al Agha (1082677), Mohammed Houssain (1084308)
Under the supervision of Dr. Mohammed Ghazal & Engineer Maha Yaghi
Abu Dhabi University, College Of Engineering, Abu Dhabi, UAE

***Abstract*—The car rescue app is a very useful tool in emergency situations. It has many important features like Firebase, Ionic Native, Geolocation, Google Maps API, and Navigation and Indoor Localization. These features work together to provide a smooth experience for users who need emergency roadside assistance or medical aid. The app has a list of emergency services that users can easily contact. With Firebase and Remote Persistence, the app manages data efficiently and updates service availability in real-time. The Geolocation feature tracks users' locations accurately and helps service providers respond quickly. Google Maps API enhances navigation with interactive maps and efficient route planning. Indoor Localization ensures accurate tracking in large indoor areas. These features combined improve response times and user safety. The car rescue app is designed to provide timely assistance, including medical aid, when users need it most.**

I. Introduction

Our Car Rescue App is designed to help drivers in emergency situations. It offers a variety of emergency services that are easily accessible through the app. We use Firebase, Remote Persistence, Ionic Native, Geolocation, Google Maps API, Navigation, and Indoor Localization to create a seamless user experience. This ensures that users can access the services they need quickly and easily. The app connects users with nearby experts and guides them through stressful situations.

It is the ultimate solution for drivers in need of emergency services. We understand how stressful it can be to search for assistance during critical moments, which is why our app is designed to provide a comprehensive range of services at your fingertips. You can access emergency towing, roadside assistance, lockout assistance, fuel delivery, and tire replacement with just a few taps. Our user-friendly interface ensures that you can quickly identify and contact the service you require. The Car Rescue App makes it easy to regain control of your situation promptly.

Our app relies on Firebase and Remote Persistence features to create a seamless user experience. These back-end systems keep user data synchronized across all devices, allowing for easy access from anywhere. The real-time database and remote persistence capabilities of Firebase ensure that data transfers are smooth and uninterrupted, even during critical moments. Users can review their service history thanks to these features, gaining valuable insights that can inform better decision-making in the future. With these tools, users can rest assured that their information is always safe, secure, and accessible.

Incorporating the functionalities of Ionic Native and Geolocation into our Car Rescue App has significantly improved the precision and effectiveness of emergency service dispatches. With this app, users can share their exact location with service providers, leading to a prompt response. The cutting-edge geolocation capabilities enable real-time monitoring, enabling professionals to reach users expeditiously without any delays. This streamlined approach substantially reduces waiting times, thereby increasing user contentment and facilitating expedited and safe road trips.

This application tackles the daunting task of navigating through unknown territories in emergency situations by seamlessly integrating Google Maps API, Navigation, and Indoor Localization functionalities. When users require aid, our app delivers step-by-step guidance, enabling first responders to promptly locate them. Additionally, with the aid of indoor localization technology, experts can effortlessly pinpoint users' whereabouts in complex parking structures or expansive facilities. This unparalleled accuracy translates into quicker response times and mitigates any unnecessary delays in resolving vehicular mishaps.

In short, our revolutionary Car Rescue App boasts an array of functionalities supported by advanced technologies such as Firebase and Remote Persistence, Ionic Native and Geolocation, Google Maps API, Navigation, and Indoor Localization. The app features a handpicked list of emergency services that connect users to nearby professionals with just a tap. With seamless navigation integrated into the app's interface, drivers can easily overcome car troubles without breaking a sweat. Our app guarantees peace of mind on the road by ensuring swift and reliable assistance at all times. Join us as we transform the way drivers handle car emergencies with unparalleled convenience and safety measures in place.

II. Explanation Of Registration Page Code

A. Registration Page HTML Code

The `ion-content` component represents the main content area of the page. By using the attribute `[fullscreen]="true"`, it occupies the entire screen and has the `body` class for additional styling. Inside the `ion-content`, there is a `div` element that contains an `img` tag. The `img` tag displays an image specified by a base64-encoded string. Following the `div`, there is a `table` element with the class `table`. It consists of three rows of table data represented by `tr` tags. Each row has two

cells represented by `td` tags—one for a label and one for an input field. The input fields are linked to properties of the `anItem` object using `[(ngModel)]` for two-way data binding. After the table, there is a line break represented by the `br` tag. A `div` with the class `center` contains an `ion-button` component. The button has two event listeners: one for the `storeItemRemote()` function and another for the `navigateTohome()` function. The button's text is "submit", and its style is set to "clear". A `footer` element includes a line break and a `p` tag with the class `center`. The paragraph displays a copyright notice. Overall, this code represents a form-like layout with an image, a table with input fields, a submit button, and a footer. It utilizes Ionic components and directives to create a mobile-friendly user interface.

B. Registration Page TS Code

This code utilizes the "Item" class to store information related to a registration form. The "RegPage" component is responsible for handling the submission of the registration form and imports necessary modules. It defines several variables, including "anItemRemote," "anItem," "lat," and "lng." When the view enters, the "ionViewDidEnter()" function sets the background color of the content element. The "navigateTohome()" function uses the NavController to navigate to the home page. To store the registration data remotely, the "storeItemRemote()" function creates a new "Item" object and interacts with Firebase Realtime Database. Additionally, the "observeLocation()" function utilizes the Capacitor Geolocation plugin to track the device's geolocation. It assigns the latitude and longitude coordinates to "lat" and "lng" while also calling the "storeItemRemote()" function to store the registration data remotely. The "ngOnInit()" function initializes the component and initiates the observation of the device's location. Overall, this code captures the device's geolocation and saves the registration data in Firebase Realtime Database.

C. Registration Page SCSS Code

This code consists of CSS styles targeting various elements and classes. The `#container` selector applies styles to a container element. It centers the content vertically using absolute positioning and translation. The `#container strong` selector sets the font size and line height for any `strong` element within the container. The `#container p` selector sets the font size, line height, color, and margin for any `p` element within the container. The `#container a` selector removes the default text decoration for any `a` element within the container. The `--ion-content` selector sets the background color of the `ion-content` element to red. The `.toolbar` selector sets the background color of elements with the class `toolbar` to red. The `--ion-toolbar` selector sets the background color of `ion-toolbar` elements to red. The `ion-content` selector sets the background color of `ion-content` elements to a specific shade of red using a custom CSS variable. The `.table` selector centers the table by setting the left and right margins and adding some right padding. The `.input` selector styles input elements with a white background, thin gray border, rounded border corners, and specific height and width. The `.text` selector styles elements with the class `text` by setting the font color to white, specifying a font family, and making the text bold. The `.btn` selector styles elements with the class `btn` by setting the text color to black, background color to white, and adding rounded corners to the border. The `.center` selector centers elements by using flexbox properties to set the flex direction to column and align items to the center. In summary, these CSS styles define the appearance of various elements within a container, including text, links, tables, inputs, buttons, and toolbar components. The styles set colors, font properties, spacing, and alignment to achieve the desired visual layout.

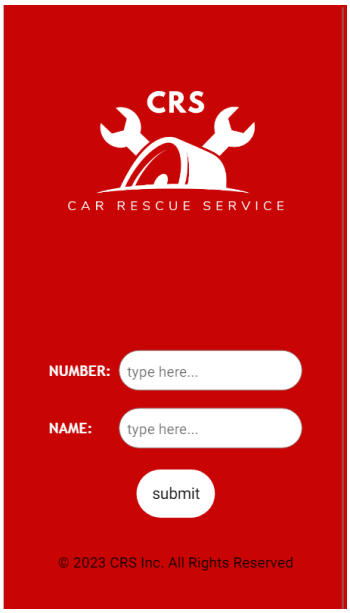

Fig. 1. Registration Page

III. Explanation Of Nearby User Code

A. Search For Nearby Users For Help HTML Code

This code represents the layout structure of the nearby users page. It consists of an `ion-header` with a translucent background and an `ion-toolbar` with custom styling. Inside the header, there is a table containing a back button implemented as an `ion-button` with a specific icon. The title of the app and "CRS" is displayed using a custom CSS class. The main content of the app is enclosed within an `ion-content` element. It includes a `div` element with an ID of "map" likely indicating the integration of a map component. Below the map, there is a line break represented by `<br>` code. Lastly, a copyright notice is displayed using a CSS class to center-align the text. Overall, this code showcases the use of Ionic components, custom styling, and likely integration with a map feature to create a visually appealing and functional app interface.

B. Search For Nearby Users For Help TS Code

This code represents an Angular module implementation for a page called ”CallNearPage” in an Ionic app. The module imports several Angular and Ionic modules, such as `CommonModule`, `FormsModule`, and `IonicModule`, which provide essential functionalities for building Angular and Ionic applications. Additionally, it imports a routing module specific to the `CallNearPage` called `CallNearPageRoutingModule`. Within the module, the `CallNearPage` component is declared, indicating that this module is responsible for managing the dependencies and configuration related to this specific page. Overall, this code demonstrates the modular approach in Angular development, where different functionalities and components are organized into modules, promoting code reuse, separation of concerns, and maintainability. The combination of Angular and Ionic modules helps to create a robust and interactive user interface for the ”CallNearPage” within the larger Ionic app ecosystem.

C. Search For Nearby Users For Help SCSS Code

In this code, the `.hh` class is defined with a custom property `--background` set to `#c90404`. The `!important` keyword is used to ensure that this style takes precedence over other styles applied to the element. This class is likely applied to an `ion-toolbar` element, resulting in a red background color.

The `#map` selector targets an element with the ID ”map” and sets its width and height to 100%, allowing the element to fill its parent container.

The `.center` class utilizes flexbox properties to horizontally and vertically center its contents. By setting the `display` property to `flex` and the `flex-direction` property to `column`, the content is aligned along the vertical axis.

Overall, this CSS code demonstrates the customization of styles for specific elements in an Ionic app. The use of custom properties, selectors, and flexbox properties enhances the visual appeal and layout of the app’s components and ensures consistent design across different devices.

IV. Explanation Of Choose Emergency Code

A. Choose Emergency HTML Code

In this code it showcases a user interface (UI) component consisting of an Ionic header, toolbar, content area, and a button for the choose emergency page. The <ion-header>component creates a header section at the top of the screen with a semi-transparent background. Within the header, the <ion-toolbar>component with the class ”hh” contains an <ion-title>component, serving as the title of the header. The title is set to ”CRS” and styled with white text color. Inside the <ion-title>, there is a <table>element with a single row and three cells. The first cell contains an <ion-button>component with the ”caret-back-outline” icon and a click event listener linked to the ”navigateToHome()” function. The second cell contains a element with the class ”crs” and the text ”CRS”. The third cell is empty. Following the header, the <ion-content>component is present with the attribute [fullscreen]=”true”, indicating that it should occupy the entire screen. Inside the content area, there is a element with the class ”card” that includes an <img>tag with the class ”img1” and a source attribute pointing to a Base64-encoded PNG image. Below the image, there is another element with the class ”card-text” containing an <h2>heading and a paragraph of text. After the card section, there is a element with the class ”center” that houses an <ion-button>component with the class ”rr”. Clicking the button triggers the ”navigateToNearbymechanic()” function. The button displays the text ”Nearby car mechanics”. Lastly, there is a <p>element with the class ”center” at the end of the code, which displays a copyright notice. In summary, this code represents an Ionic UI component with a header, toolbar, content area, and button. It utilizes various Ionic components, classes, and event listeners to create a visually appealing and interactive mobile app interface.

Fig. 2. Registration Page

B. Choose Emergency TS Code

This code begins with importing necessary modules and dependencies from the Angular and Ionic frameworks. These imports include modules such as `NgModule` and `CommonModule` from the `angular/core` and `angular/common` packages, respectively. It also imports the `FormsModule` from the `angular/forms` package and `IonicModule` from the `ionic/angular` package. Additionally, a custom routing module called `ChooseEmerPageRoutingModule` is imported. Within the `NgModule` decorator, the `imports` array specifies the modules that will be used within this module. In this case, it includes `CommonModule`, `FormsModule`, `IonicModule`, and `ChooseEmerPageRoutingModule`. The

declarations array lists the components, directives, and pipes that belong to this module. Here, it only declares the ChooseEmerPage component. Finally, the ChooseEmerPageModule class is exported, making it accessible for other modules or components to import and use. In summary, this code demonstrates the import and configuration of modules in an Angular and Ionic application. It showcases the organization of dependencies and the declaration of specific components within a module for effective modular development.

C. Choose Emergency SCSS Code

The code defines several CSS styles to enhance the visual appearance of various elements on a web page or application. The .hh class sets the background color to a specific shade of red. The .card class positions a container element relatively, while the .card img class adjusts the maximum width of images to 100% while maintaining their proportional height. The .card-text class positions the text at the top of the card element, sets its color to white, and adds padding. The .img1 class adds a border radius to the bottom left and bottom right corners of an image, and the .img2 class applies a border radius to images. The .rr class defines styles for a button, including text color, border, and border radius. The .rr:hover class specifies styles for the button when hovered over, changing the text color to white and setting the background color to the same shade of red. The .center class is used to center elements by utilizing flex display, aligning items vertically in a column direction, and horizontally centering them. These CSS styles contribute to the overall visual presentation and aesthetic appeal of the elements within the web page or application.

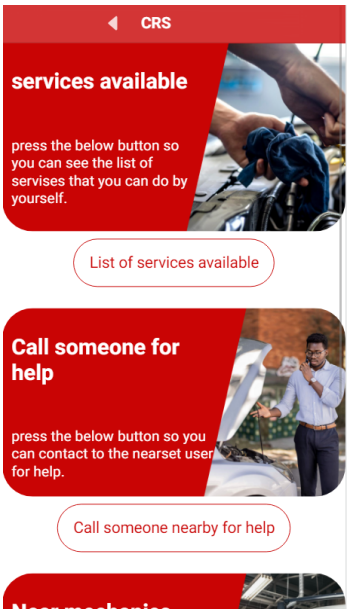

Fig. 3. Registration Page

V. Explanation Of Home Page Code

A. Home Page HTML Code

This code represents the home page for a user interface (UI) with a header, content area, and a copyright notice. The < ion-header > component creates a header section at the top of the screen. The translucent attribute is set to true, indicating that the header will have a semi-transparent background. The class attribute is set to ”body”. Inside the header, there is an < ion-toolbar > component with the class ”hh”. It contains an < ion-title > component with white text color. The title text is ”CRS”. The < ion-content > component represents the content area of the UI. The fullscreen attribute is set to true, indicating that the content area should occupy the entire screen. Within the content area, there is a ¡div¿ element that contains an < img > tag. The image source is a Base64-encoded JPEG image. Following the image, there is an < ionic-grid > component that is not properly closed in the provided code. After the grid, there is a line break < br > tag. At the end of the code, there is a < p > element with the class ”center” that displays a copyright notice.

B. Home Page TS Code

The import statements at the top include necessary modules and dependencies such as Component and NavController. The Component decorator defines the metadata for the component, including the selector, template URL, and style URL(s). The HomepgPage class implements the OnInit interface, indicating that it has a lifecycle hook method ngOnInit(). The constructor initializes the NavController as a private property of the component. The navigateToChooseEmg() method is defined within the class. It uses the NavController to navigate to the ”choose-emer” page when called. The ngOnInit() method is an Angular lifecycle hook that is executed when the component is initialized. In this code, it is empty and does not contain any logic. In summary, this code sets up the ”HomepgPage” component in an Ionic application. It imports necessary modules and dependencies, defines methods for navigating to other pages, and implements the OnInit interface.

C. Home Page SCSS Code

The .hh class sets the background color of elements to a specific shade of red using a custom CSS variable. The .rr class defines styles for a button. It sets the text color to a specific shade of red, adds a thin solid border, and applies border radius to the button. The .rr:hover class specifies styles for the button when hovered over. It changes the text color to white and sets the background color to the same shade of red. The .crs class sets the display of elements to grid and centers the items horizontally using the justify-items property. The .center class is used for centering elements. It sets the display to flex, aligns items vertically in a column direction, and horizontally centers them. The hr selector sets the background color of horizontal rules to gray. The .textt class defines styles for text elements. It aligns the text to the center, sets the color to white, makes it bold, and adds padding at the top. The .roww class applies a background color and

border radius to the top-right and bottom-right corners of a row element. The `.roww2` class applies a background color and border radius to the top-left and bottom-left corners of a row element. In summary, this code provides visual styles for different elements in a web page or application, including background colors, button styles, text alignment, grid layout, centering of elements, and border radius for rows.

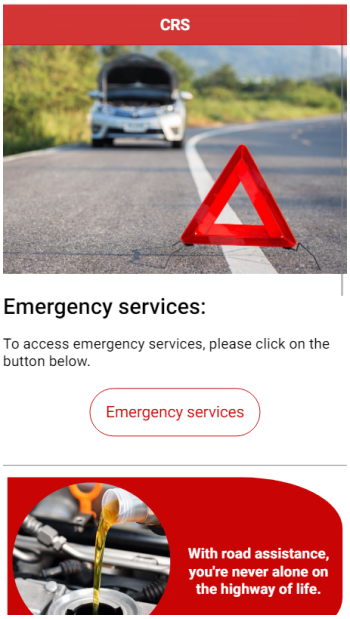

Fig. 4. Registration Page

VI. Explanation Of Services Available Code

A. Services Available HTML Code

In this code, it defines the layout and functionality of a page that provides solutions for common car issues. The code starts with an <ion-header >element, which contains a translucent toolbar with a title. The toolbar includes a back button for navigation. The main content is wrapped in an <ion-content >element, which has the attribute [fullscreen]=”true” and adds padding. The content begins with a heading and an unordered list describing the issue of running out of fuel while driving. A button labeled ”Route to nearest ADNOC station” is provided to search for the nearest ADNOC (a fuel station brand) stations. The content then continues with sections for other common car issues, such as overheating, flat tire, cruise control failure, key stuck in the ignition, and being stuck in scenarios like mud or sand. Each issue has a heading, a brief description, and a button labeled ”Show Steps”. Clicking this button toggles the display of a list of steps to resolve the specific issue. The steps are displayed using an <ion-header >element with multiple <ion-item >elements. Finally, there is a copyright notice at the bottom of the page. Overall, the code defines a user interface for a mobile application that provides solutions for various car issues and allows users to navigate through the different sections and view the steps for each problem.

B. Services Available TS Code

In this code, the component is defined with the selector ’app-list-serv’, indicating that it can be used in an HTML template with that tag. The component imports necessary dependencies, including `Component` and `OnInit` from `angular/core` and `NavController` from `ionic/angular`. The component class `ListServPage` implements the `OnInit` interface, which requires implementing the `ngOnInit()` method. The class has a property `showDropdown` initialized as null or a number. It is used to keep track of the currently displayed dropdown section. The constructor injects the `NavController` instance for navigation purposes. The `nearstAdnoc()` method is called when the user clicks the ”Route to nearest ADNOC station” button. It uses the `NavController` to navigate to the `/near-adnoc` page. The `back()` method is called when the user clicks the back button. It uses the `NavController` to navigate to the `/choose-emer` page. The `ngOnInit()` method is empty in this code, as it is not being used to perform any specific initialization tasks. The `toggleDropdown(index: number)` method is used to toggle the display of dropdown sections. It takes an index as a parameter and sets the `showDropdown` property to either the passed index or null, depending on its current value. Overall, this code sets up the functionality for navigation and toggling dropdown sections within the `ListServPage` component of an Ionic application.

C. Services Available SCSS Code

This code defines some CSS styles for a specific class and HTML elements. `.hh` is a class selector used to target elements with the class `hh`. It sets the `--background` CSS variable to the color `#c90404` for those elements. The `!important` rule ensures that this style takes precedence over other styles. `.rr` is a class selector used to target elements with the class `rr`. It sets the text color to `#c90404`, adds a thin solid border, and applies a border radius of 25pt (points) to those elements. `.rr:hover` is a selector that targets elements with the class `rr` when the user hovers over them. It changes the text color to white and sets the background color to `#c90404`, creating a hover effect. `hr` is an element selector that targets all horizontal rule elements. It sets the background color to gray. `.center` is a class selector used to target elements with the class `center`. It applies flexbox styles to those elements, making them display as a column and aligning their items in the center vertically.

VII. Explanation Of Nearby Mechanics Code

A. Nearby Mechanics HTML Code

This code represents the structure and content of the nearby mechanics page. The page begins with a header section created using the `ion-header` component. The header is made translucent with the `[translucent]="true"` attribute. Within the header, there is a toolbar (`ion-toolbar`) with the class ”hh”. Inside the toolbar, there is a title (`ion-title`) with white text color. The title contains a table element

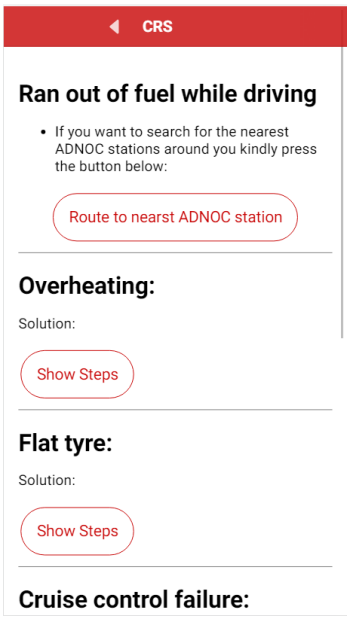

Fig. 5. Registration Page

(table) with a single row (tr) and three table cells (td). The first cell contains a button (ion-button) with a clear fill style and a back arrow icon (ion-icon). The second cell contains a span element (span) with the class "crs" displaying the text "CRS". The main content area of the page is represented by the ion-content component. The content area covers the entire screen ([fullscreen]="true"). Inside the content area, there is a div element (div) with the class "map" and an ID of "map". This div is used for displaying a map and can be referenced in the component code using the #map attribute. After the div, there are line breaks and a button (ion-button) with the class "rr" and a clear fill style. This button has a click event handler bound to the "RouteToNextNearest()" function and displays the text "ride to nearest". Further line breaks and a paragraph element (p) with the class "center" display a copyright message. In summary, this code defines the layout of an Ionic application template with a header, a content area containing a map and a button, and a copyright message at the bottom. It also includes event bindings for button clicks.

B. Nearby Mechanics TS Code

This code represents the implementation of a page called "NearbyMechanPage" in an Ionic application. The page is designed to display a map and show nearby car service locations. The necessary dependencies, including the Capacitor geolocation plugin, Ionic's NavController, and the Google Maps API, are imported.

The component declares various variables to store information such as latitude, longitude, the map itself, map element reference, service details, nearest mechanics, display settings, and nearest count. In the constructor, these variables are initialized with default values.

The back() function enables navigation to the previous page. The getPosition() function utilizes the Geolocation plugin to retrieve the current device coordinates. The loadMap() function is responsible for loading the map. It first calls getPosition() to obtain the current coordinates and then sets up the map options. A new instance of the Google Map is created, and a marker is added to represent the current location. Additionally, the Google Places service is initialized to facilitate the search for nearby car service locations.

The ngOnInit() function is called when the component is initialized and triggers the loadMap() function to load the map. The addMarker() function adds a marker to the map at the specified latitude and longitude coordinates, utilizing a custom image. It also sets up an info window to display the place name when the marker is clicked.

The GooglePlacesNearbySearch() function performs a nearby search for car service locations using the Google Places service. It sends a request with the current coordinates and a keyword, and when the results are returned, it adds markers for each found place and stores them in the nearestmechan array.

The RouteToNextNearest() function calculates the route from the current location to the next nearest car service location in the nearestmechan array. It utilizes the DirectionsService to obtain the route and displays it on the map using the DirectionsRenderer.

The drawRoute() function takes the start and end coordinates and employs the DirectionsService to calculate the route between them. The resulting route is then displayed on the map using the DirectionsRenderer.

In summary, this code sets up a page with a map where the user's current location is retrieved. It performs a nearby search for car service locations and allows the user to view routes to the nearest car service locations.

C. Nearby Mechanics SCSS Code

We utilized the provided code to design the Nearby Mechanics page. The styling includes various selectors to define the appearance of different elements on the page.

The #container selector is used to style a container element, applying centered text formatting. It employs absolute positioning and transforms to ensure the container is vertically centered in the middle of the page. Within the container, the #container strong selector is used to style strong elements, setting a font size of 20 pixels and a line height of 26 pixels.

Paragraph elements inside the container are styled using the #container p selector, with a font size of 16 pixels and a line height of 22 pixels. The text color is set to a shade of gray (#8c8c8c), and there is no margin.

Anchor elements inside the container are styled using the #container a selector, removing any text decoration such as underlines.

The #map selector is used to style a map element, specifying a width of 100% and a height of 80% of its container.

The .hh selector defines a custom class, setting the background color to a shade of red (#c90404) and marking it as important to override any conflicting styles.

The .rr selector styles elements with the class ”rr”. They are given a red color (#c90404), a thin solid border, and a border radius of 25 pixels. On hover, the color changes to white, and the background color becomes the same shade of red.

The .mapp selector styles elements with the class ”mapp”, assigning a thin solid border color of #c90404.

Additionally, the .rr selector absolutely positions elements with the class ”rr” at the top 85% and left 30% of their containing element.

The .center selector styles elements with the class ”center” by setting their display property to flex, aligning items both vertically and horizontally.

Overall, this CSS code provides styling rules for positioning, font sizes, colors, and borders to create a visually appealing and centered layout for the web page.

Fig. 6. Registration Page

VIII. App Module TS Code Explanation

The code imports necessary modules and classes from Angular and Ionic packages. It imports the AppComponent and AppRoutingModule from the application files. The Firebase SDK functions getAnalytics and initializeApp are imported from the ”firebase/analytics” and ”firebase/app” packages, respectively. These functions will be used to set up Firebase analytics and initialize the Firebase app. The Firebase configuration object firebaseConfig contains the necessary credentials and settings for connecting to the Firebase backend. It includes properties such as apiKey, authDomain, databaseURL, projectId, storageBucket, messagingSenderId, and appId. The Firebase app is initialized by calling initializeApp(firebaseConfig), passing the firebaseConfig object as a parameter. This initializes the Firebase app with the provided configuration. The getAnalytics(app) function is called to initialize Firebase analytics for the app.

The AppModule class is decorated with NgModule to define the Angular module. Within the NgModule decorator, the declarations property specifies the components declared within the module (in this case, only the AppComponent). The imports property lists the imported modules (BrowserModule, IonicModule, and AppRoutingModule). The providers property specifies the providers used in the module. Here, it includes the RouteReuseStrategy with the IonicRouteStrategy class as its implementation. The bootstrap property indicates the component to be bootstrapped when the module is loaded, which is the AppComponent in this case.

In summary, the code sets up the Angular module for the application, imports necessary modules, initializes the Firebase app with the provided configuration, and defines the root component (AppComponent) and routing configuration for the application.

IX. Conclusion

The Car Rescue App is a remarkable achievement in utilizing technology to enhance emergency response and provide timely aid to individuals. It integrates various features, such as Firebase integration, Ionic Native functionalities, Geolocation services, the Google Maps API, and Indoor Localization. All of these features work together seamlessly to create an efficient user experience.

With the Car Rescue App, users have immediate access to a comprehensive list of emergency services for roadside or medical emergencies. The integration of Firebase and Remote Persistence ensures effective data management which allows for real-time updates on service availability. This enhances the overall reliability of the app.

Geolocation functionality accurately tracks users’ locations which enables service providers to respond promptly and effectively. By incorporating the Google Maps API, navigation becomes more efficient with interactive maps and route planning that further expedites emergency response times.

Furthermore, Indoor Localization functionality ensures precise tracking even in large indoor areas which enhances the app’s effectiveness in diverse environments.

Overall, prioritizing user safety was at the forefront when designing the Car Rescue App. It provides essential medical assistance during critical situations through harnessing technology and capitalizing on mobile application connectivity. The potential impact during emergencies is significant as individuals can rest assured knowing they will receive necessary help when it is most needed.