Insurance Management System

1. Introduction

The purpose of this document is to provide a detailed Low-Level Design (LLD) for the Insurance Management System. The system facilitates streamlined insurance operations, including customer management, policy handling, and claim processing. It employs a REST API-based backend architecture and uses either Angular or React for the frontend. This design is compatible with both Java (Spring Boot) and .NET (ASP.NET Core) frameworks.

2. Module Overview

Customer Management: Manages customer data, including personal information, contact details, and policy associations.

Policy Management: Handles creation, updating, and archiving of insurance policies.

Claim Processing: Facilitates filing, reviewing, and approval/rejection of claims.

Agent Management: Manages insurance agents, their profiles, and assigned policies.

Notification and Alerts: Manages communication with customers and agents via notifications and alerts (e.g., policy renewals, claim status updates).

3. Architecture Overview

3.1 Architectural Style

Frontend: Angular or React

Backend: REST API-based architecture

Database: Relational Database (MySQL/SQL Server)

3.2 Component Interaction

Frontend communicates with the REST API backend via HTTP/HTTPS.

Backend interacts with the database layer for CRUD operations.

Frontend handles user interactions and renders dynamic views.

4. Module-Wise Design

4.1 Customer Management Module

4.1.1 Features

Add, update, and delete customer profiles

Search and view customer details

4.1.2 Data Flow

User requests to create/update/view customer details via the frontend.

The frontend sends the request to the REST API.

Backend processes the request and performs necessary database operations.

Response is sent back to the frontend for display.

4.1.3 Entities

CustomerID

Name

Email

Phone

Address

4.2 Policy Management Module

4.2.1 Features

Create, update, and manage insurance policies

Link policies to customers and agents

4.2.2 Data Flow

Admin/Agent initiates policy-related operations via the frontend.

Request is processed by the backend and mapped to database actions.

Results are displayed on the frontend.

4.2.3 Entities

PolicyID

Name

PremiumAmount

CoverageDetails

ValidityPeriod

4.3 Claim Processing Module

4.3.1 Features

File a claim

Review and process claims

Track claim status

4.3.2 Data Flow

Customers/Agents file claims via the frontend.

Backend validates claim requests and updates the database.

Claim status updates are retrieved by users through the frontend.

4.3.3 Entities

ClaimID

PolicyID

CustomerID

ClaimAmount

Status (Filed, Under Review, Approved, Rejected)

4.4 Agent Management Module

4.4.1 Features

Manage agent profiles

Assign policies to agents

4.4.2 Data Flow

Admin performs agent-related operations via the frontend.

Requests are routed to the backend for database interactions.

Responses are rendered back to the admin interface.

4.4.3 Entities

AgentID

Name

ContactInfo

AssignedPolicies

4.5 Notification and Alerts Module

4.5.1 Features

Notify users of policy updates

Send alerts for upcoming renewals and claim decisions

4.5.2 Data Flow

Backend generates notifications based on triggers (e.g., policy expiration).

Notifications are sent via email/SMS or displayed in the frontend interface.

5. Deployment Strategy

5.1 Local Deployment

The system will be deployed in a local environment. Each developer will set up the application on their local machine for testing and development purposes.

5.1.1 Frontend Deployment

Use local servers like ng serve for Angular or local development servers for React.

5.1.2 Backend Deployment

Deploy the REST API on local servers using Spring Boot/ASP.NET Core.

5.1.3 Database

Set up a local instance of the database for development and testing.

6. Database Design

6.1 Tables and Relationships

6.1.1 Customer

Primary Key: CustomerID

Foreign Keys: None

6.1.2 Policy

Primary Key: PolicyID

Foreign Key: CustomerID

6.1.3 Claim

Primary Key: ClaimID

Foreign Keys: PolicyID, CustomerID

6.1.4 Agent

Primary Key: AgentID

Foreign Key: None

6.1.5 Notification

Primary Key: NotificationID

Foreign Key: CustomerID

7. User Interface Design

7.1 Wireframes

Customer Dashboard

Policy Management Interface

Claim Filing and Tracking Page

Agent Management Console

Notifications and Alerts Panel

8. Non-Functional Requirements

8.1 Performance

The system must handle up to 100 concurrent users in the local environment.

8.2 Scalability

Designed to be easily scalable if moved to a production-grade environment in the future.

8.3 Security

Data must be secured using appropriate access controls and encryption mechanisms.

8.4 Usability

The user interface must be intuitive and responsive.

9. Assumptions and Constraints

9.1 Assumptions

The system will run only in a local environment for development and testing.

9.2 Constraints

No cloud deployment is planned at this stage.

Developers will use standard IDEs for coding and debugging.

The project is done in Clean architecture and we followed the code first approach.

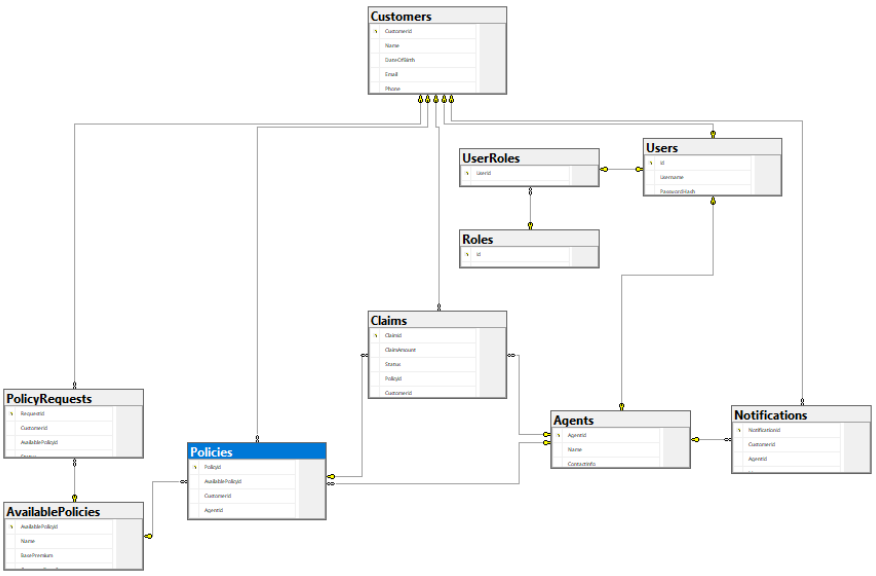
The following are the outputs and the related code for its implementation.

The below is the Structure of our Project for the UI part:

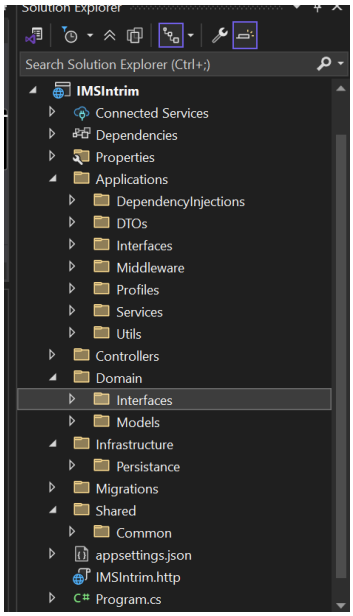
Project Directory Structure

```
src/
├── index.html
├── main.ts
├── styles.css
├── tests.ts
├── app/
│   ├── app.component.html
│   ├── app.component.css
│   ├── app.component.spec.ts
│   ├── app.component.ts
│   ├── app.config.ts
│   └── app.routes.ts
├── core/
    └── auth/
│      ├── auth.service.ts
│      ├── auth.guards.ts
│   └── intercepter/
│      └──services/
│   └── features/
│           ├── admin/
│           │   ├── admin-dashboard/
│           │   └── admin-profile/
│           ├── agent-management/
│           ├── customer-management/
│           ├── policy-management/
│           ├── agent/
│           ├── auth/
│           ├── login/
│           ├── registry/
│           ├── customer/
│           ├── home/
│           ├── products/
│       └── layouts/
│           ├── admin-layout/
│           ├── agent-layout/
│           └── customer-layout/
│        └── shared/
│            ├── components/
│            ├── confirm-dialog/
│           ├── reject-dialog/
│          ├── api-response.interface.ts
│         ├── pagedResult.dtos.ts
└── environment/
    └── environment.development.ts
```

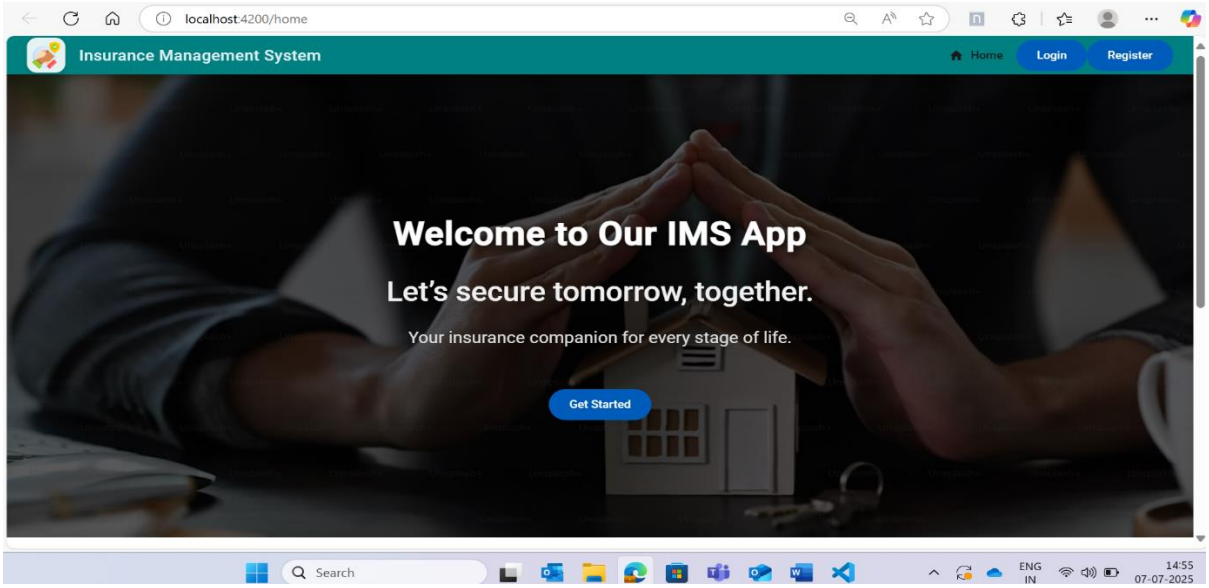
The below is the structure of our SSMS(Backend):



The below is the structure of Web API:



Home Page: This is our landing page



The Home component comprising the three files are as follows:

Home-dashboard.ts:

```
import { Component } from '@angular/core';
import { CommonModule } from '@angular/common'; // Needed for ngFor, ngIf, etc.
import { RouterLink } from '@angular/router';
import { Router } from '@angular/router';// For routerLink if you use Angular routing
import { MatToolbarModule } from '@angular/material/toolbar';
import { MatButtonModule } from '@angular/material/button';
import { MatIconModule } from '@angular/material/icon';
import { MatMenuModule } from '@angular/material/menu';
import { MatCardModule } from '@angular/material/card';
import { MatSidenavModule } from '@angular/material/sidenav';
import { MatListModule } from '@angular/material/list';


@Component({
  selector: 'app-home',
  templateUrl: './home-dashboard.component.html',
```

```
  styleUrls: ['./home-dashboard.component.css'],
  standalone: true, // Mark component as standalone if you are using Angular 15+
  imports: [
    CommonModule,
    RouterLink,
    MatToolbarModule,
    MatButtonModule,
    MatIconModule,
    MatMenuModule,
    MatCardModule,
    MatSidenavModule,
    MatListModule
  ]
})
export class HomeComponent {
  constructor(private router: Router) {}
   navigateTo(route: string) {
    this.router.navigate([route]);
  }
}
```

Home-dashboard.html:

```
<mat-toolbar color="primary" class="navbar-container">
    <img src="https://cdn-icons-png.freepik.com/256/4371/4371433.png" alt="Company Logo" class="company-logo" />
    <span class="navbar-title">Insurance Management System</span>
    <span class="spacer"></span>
    <nav class="nav-links">
      <a mat-button routerLink="/home"><mat-icon>home</mat-icon> Home</a>
```

```
    <button mat-flat-button color="accent" routerLink="/login">Login</button>
    <button mat-flat-button color="accent" routerLink="/register">Register</button>
  </nav>
 </mat-toolbar>

 <header class="hero-section">
  <div class="hero-overlay">
   <div class="hero-content">
    <h1 class="welcome-text">Welcome to Our IMS App</h1>
    <h2 class="subheading">Let’s secure tomorrow, together.</h2>
    <p class="tagline">Your insurance companion for every stage of life.</p>
    <button mat-flat-button color="accent" class="hero-cta" routerLink="/login">
     Get Started
    </button>
   </div>
  </div>
 </header>


 <section class="highlight-section">
  <h2 class="section-title">Our Key Insurance Highlights</h2>
  <div class="card-grid">
   <!-- Highlight Card 1 -->
   <div class="flip-card">
    <div class="flip-inner">
     <div class="flip-front">
      <img src="https://cdn-icons-png.freepik.com/256/4825/4825431.png"
alt="Health Icon" class="card-icon" />
```

```
        <h3>Health Insurance</h3>
      </div>
      <div class="flip-back">
        <p>Comprehensive medical coverage for your family’s well-being.</p>
        <button mat-button color="primary" routerLink="/login">Explore</button>
      </div>
    </div>
  </div>

  <!-- Highlight Card 2 -->
  <div class="flip-card">
    <div class="flip-inner">
      <div class="flip-front">
        <img src="https://cdn-icons-png.freepik.com/256/7557/7557481.png" alt="Life
Icon" class="card-icon" />
        <h3>Life Insurance</h3>
      </div>
      <div class="flip-back">
        <p>Protect your loved ones’ future with personalized life policies.</p>
        <button mat-button color="primary" routerLink="/login">Explore</button>
      </div>
    </div>
  </div>

  <!-- Highlight Card 3 -->
  <div class="flip-card">
    <div class="flip-inner">
      <div class="flip-front">
```

```
        <img src="https://cdn-icons-png.freepik.com/256/11442/11442848.png"
alt="Vehicle Icon" class="card-icon" />
        <h3>Vehicle Insurance</h3>
       </div>
       <div class="flip-back">
        <p>Secure your ride with smart protection and quick claims.</p>
        <button mat-button color="primary" routerLink="/login">Explore</button>
       </div>
      </div>
    </div>

    <!-- Highlight Card 4 -->
    <div class="flip-card">
     <div class="flip-inner">
      <div class="flip-front">
        <img src="https://cdn-icons-png.freepik.com/256/9202/9202595.png"
alt="Property Icon" class="card-icon" />
        <h3>Property Insurance</h3>
       </div>
       <div class="flip-back">
        <p>Protect your valuable assets against unexpected damage or loss.</p>
        <button mat-button color="primary" routerLink="/login">Explore</button>
       </div>
      </div>
    </div>
   </div>
  </section>

  <footer class="footer">
```

```
    <div class="footer-content">
      <p>&copy; 2025 Insurance Management App. All rights reserved.</p>
      <div class="social-icons">
        <a href="#"><i class="fab fa-facebook-f"></i></a>
        <a href="#"><i class="fab fa-twitter"></i></a>
        <a href="#"><i class="fab fa-linkedin-in"></i></a>
        <a href="#"><i class="fab fa-instagram"></i></a>
      </div>
    </div>
  </footer>
```

Home-dashboard.css:

```
body {
  margin: 0;
  font-family: 'Roboto', sans-serif; /* Or your preferred font-family */
  background-color: #f4f7f6; /* Light background for the whole page */
  color: #333;
}

/* Toolbar and Nav */
.navbar-container {
  display: flex;
  align-items: center;
  padding: 0 1.5rem;
  /* Attractive Navbar Color: A darker, rich blue */
  background-color: #008080; /* Deep Ocean Blue */
  color: white;
  height: 50px; /* Standard Material toolbar height */
  box-shadow: 0 4px 10px rgba(0, 0, 0, 0.25); /* Stronger, professional shadow */
```

```
  position: sticky;
  top: 0;
  z-index: 1000;
}

.company-logo {
  width: 35px;
  height: 35px;
  border-radius: 12px;
  object-fit: contain;
  margin-right: 1rem;
  background-color: rgba(255, 255, 255, 0.9); /* Slightly transparent white for logo background */
  padding: 5px;
  box-shadow: 0 2px 5px rgba(0,0,0,0.1); /* Subtle shadow for logo */
}

.navbar-title {
  /* Font size: Not too large, clear and readable */
  font-size: 1.2rem; /* Slightly reduced from 1.8rem for less large feel */
  font-weight: 500;
  color: #e0f2f7; /* A lighter blue-white for title */
  letter-spacing: 0.5px;
}

.spacer {
  flex: 1;
}
```

```
.nav-links {
  display: flex;
  align-items: center;
}

/* Base styles for all nav links/buttons */
.nav-links a.mat-button,
.nav-links button.mat-stroked-button,
.nav-links button.mat-flat-button {
  margin-left: 1.5rem;
  font-size: 0.95rem; /* Consistent smaller font size for all nav text */
  font-weight: 500;
  text-transform: uppercase;
  transition: all 0.3s ease; /* Smooth transition for all properties */
  min-width: unset;
  padding: 8px 16px;
  border-radius: 4px; /* Slightly rounded for all */
}

/* Specific styling for the Home link */
.nav-links a.mat-button {
  color: #e0f2f7; /* Light blue-white for Home */
}
.nav-links a.mat-button mat-icon {
  margin-right: 5px;
}
.nav-links a.mat-button:hover {
```

```
  color: #ffc107; /* Bright yellow on hover */
  background-color: rgba(255, 255, 255, 0.15); /* More visible background on hover */
}



/* Specific styling for Register button (mat-flat-button) */
.nav-links button.mat-flat-button {
  background-color: #ffc107; /* Solid yellow background */
  color: #1a4a75; /* Dark blue text */
  padding: 8px 16px; /* Standard flat button padding */
  border-radius: 20px; /* Pill shape for Register as well Register*/
}
.nav-links button.mat-flat-button:hover {
  background-color: #e0a800; /* Darker yellow on hover */
  color: #0d2e4f; /* Even darker blue text on hover */
  box-shadow: 0 4px 12px rgba(255, 193, 7, 0.6);
  transform: translateY(-2px); /* Slight lift on hover */
}

/* Full-page hero section */
.hero-section {
  width: 100%;
  height: calc(100vh - 64px); /* Subtract navbar height */
  /* Re-introducing the background image (from 10:03 screenshot) */
  background: url('https://plus.unsplash.com/premium_photo-1661776260388-
f5d1b14ce8a2?q=80&w=1332&auto=format&fit=crop&ixlib=rb-
4.1.0&ixid=M3wxMjA3fDB8MHxwaG90by1wYWdlfHx8fGVufDB8fHx8fA%3D%3D') no-
repeat center center/cover;
  position: relative;
```

```
  display: flex;
  align-items: center;
  justify-content: center;
  overflow: hidden;
  color: white;
  box-shadow: inset 0 -10px 15px rgba(0, 0, 0, 0.1);
}

/* Dark semi-transparent overlay */
.hero-overlay {
  background-color: rgba(0, 0, 0, 0.65); /* Keeps the dark overlay */
  width: 100%;
  height: 100%;
  display: flex;
  align-items: center;
  justify-content: center;
  padding: 2rem;
}

/* Text content */
.hero-content {
  text-align: center;
  max-width: 800px;
  opacity: 0;
  transform: translateY(30px);
  animation: fadeInUp 1.2s ease-out 0.2s forwards;
  position: relative;
}
```

```
.welcome-text {
  font-size: 2.6rem;
  font-weight: bold;
  margin-bottom: 1rem;
  color: #fff;
  text-shadow: 3px 3px 8px rgba(0, 0, 0, 0.6);
  letter-spacing: 1px;
}

.subheading {
  font-size: 2.2rem;
  margin-bottom: 1.2rem;
  color: #f1f1f1;
  font-weight: 500;
}

.tagline {
  font-size: 1.3rem;
  color: #e0e0e0;
  margin-bottom: 3rem;
  line-height: 1.7;
}

/* CTA button */
.hero-cta.mat-flat-button {
  padding: 1.2rem 3rem;
  font-size: 1.3rem;
```

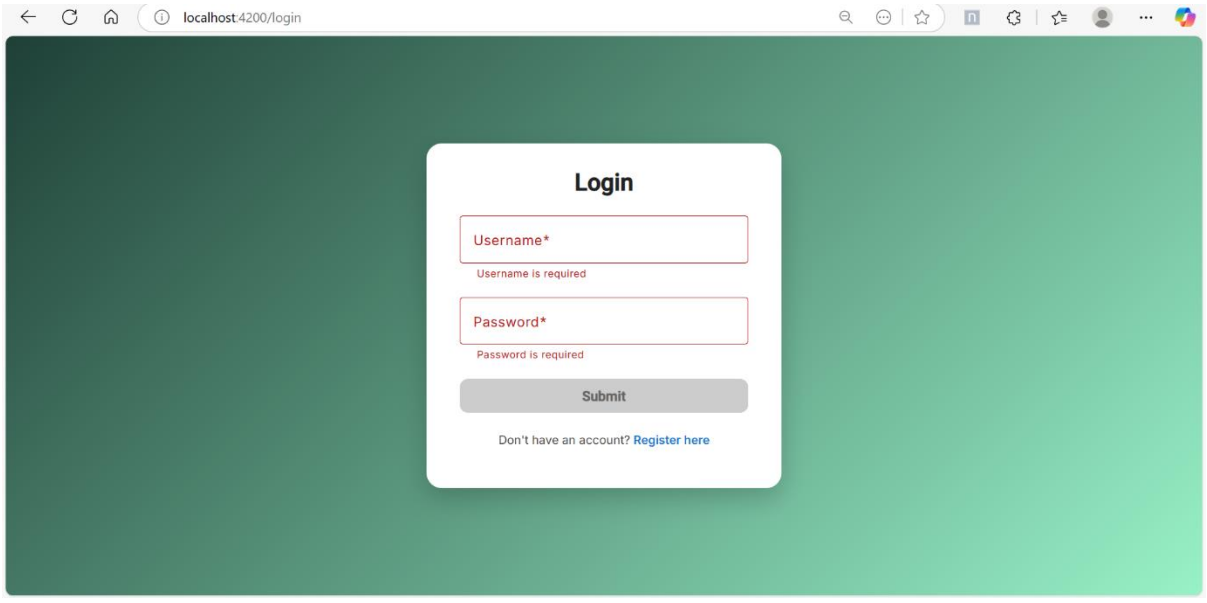
```
  border-radius: 50px;
  background-color: #ffc107;
  color: #1a4a75; /* Matches new navbar color for contrast */
  border: none;
  box-shadow: 0 6px 15px rgba(0, 0, 0, 0.3);
  transition: all 0.3s ease;
}

.hero-cta.mat-flat-button:hover {
  transform: translateY(-5px) scale(1.03);
  background-color: #e0a800;
  color: #0d2e4f;
  box-shadow: 0 10px 20px rgba(0, 0, 0, 0.4);
}

/* Keyframe animation */
@keyframes fadeInUp {
  to {
    opacity: 1;
    transform: translateY(0);
  }
}

/* Responsive tweaks for Hero Section */
@media (max-width: 768px) {
  .welcome-text {
    font-size: 2.8rem;
  }
```

```
  .subheading {
    font-size: 1.8rem;
  }
  .tagline {
    font-size: 1.1rem;
  }
  .hero-cta.mat-flat-button {
    padding: 1rem 2.5rem;
    font-size: 1.1rem;
  }
}

@media (max-width: 500px) {
  .welcome-text {
    font-size: 2rem;
  }
  .subheading {
    font-size: 1.4rem;
  }
  .tagline {
    font-size: 1rem;
  }
  .hero-overlay {
    padding: 1rem;
  }
}
```

```
/* Highlight Grid */
.highlight-section {
  padding: 3rem 1rem;
  text-align: center;
}
.section-title {
  font-size: 2rem;
  margin-bottom: 2rem;
}
.card-grid {
  display: grid;
  grid-template-columns: repeat(auto-fit, minmax(240px, 1fr));
  gap: 2rem;
  justify-items: center;
}

/* Flip Cards */
.flip-card {
  perspective: 1000px;
  width: 240px;
  height: 260px;
}
.flip-inner {
  position: relative;
  width: 100%;
  height: 100%;
  transition: transform 0.7s ease;
  transform-style: preserve-3d;
```

```
}
.flip-card:hover .flip-inner {
 transform: rotateY(180deg);
}
.flip-front,
.flip-back {
 position: absolute;
 width: 100%;
 height: 100%;
 backface-visibility: hidden;
 background: #fff;
 border-radius: 12px;
 box-shadow: 0 10px 24px rgba(0, 0, 0, 0.1);
 padding: 1.25rem;
 display: flex;
 flex-direction: column;
 align-items: center;
 justify-content: center;
}
.flip-back {
 transform: rotateY(180deg);
 background-color: #f3f3f3;
}
.card-icon {
 width: 64px;
 height: 64px;
 margin-bottom: 1rem;
 object-fit: contain;
```

```
}

/* Footer */
.footer {
 background: #222;
 color: white;
 padding: 2rem 1rem;
 text-align: center;
}
.footer .social-icons a {
 color: #fff;
 margin: 0 0.5rem;
 font-size: 1.2rem;
}
.footer .social-icons a:hover {
 color: #ffc107;
}
```



Login.css:

24

```
@import
url('https://fonts.googleapis.com/css2?family=Inter:wght@400;600;700&display=swap'
);
* {
 font-family: 'Inter', sans-serif;
 box-sizing: border-box;
 margin: 0;
 padding: 0;
}
.login-container {
 display: flex;
 justify-content: center;
 align-items: center;
 height: 100vh;
 background: linear-gradient(135deg, #1f4037, #99f2c8);
 padding: 20px;
}
 /* Card appearance */
.login-card {
 width: 100%;
 max-width: 420px;
 background-color: #ffffff;
 border-radius: 18px;
 padding: 32px 24px;
 box-shadow: 0 12px 30px rgba(0, 0, 0, 0.2);
 transition: transform 0.3s ease, box-shadow 0.3s ease;
}

.login-card:hover {
```

```
  transform: translateY(-4px);
  box-shadow: 0 16px 40px rgba(0, 0, 0, 0.25);
}
/* Title styling */
mat-card-title {
  text-align: center;
  font-size: 28px;
  font-weight: 700;
  color: #222;
  margin-bottom: 25px;
}
/* Input fields */
.full-width {
  width: 100%;
  margin-bottom: 20px;
}
mat-form-field {
  transition: box-shadow 0.3s ease;
}
mat-form-field:hover {
  box-shadow: 0 0 0 2px rgba(31, 64, 55, 0.15);
  border-radius: 6px;
}
/* Button styling */
.submit-btn {
  width: 100%;
  padding: 12px;
  font-size: 16px;
```

```
  font-weight: 600;
  background: linear-gradient(135deg, #00c6ff, #0072ff); /* new gradient */
  color: #fff;
  border: none;
  border-radius: 10px;
  cursor: pointer;
  box-shadow: 0 4px 14px rgba(0, 114, 255, 0.3);
  transition: background 0.3s ease, transform 0.2s ease, box-shadow 0.3s ease;
}
.submit-btn:hover:not([disabled]) {
  background: linear-gradient(135deg, #0072ff, #00c6ff); /* reverse gradient */
  transform: translateY(-2px);
  box-shadow: 0 6px 20px rgba(0, 114, 255, 0.4);
}

.submit-btn:disabled {
  background: #cccccc !important;
  color: #666666;
  cursor: not-allowed;
  box-shadow: none;
}
/* Error message */
.error-text {
  color: #d32f2f;
  font-size: 14px;
  margin-top: 12px;
  text-align: center;
}
```

```
/* Register link section */
.register-text {
  margin-top: 24px;
  font-size: 14px;
  text-align: center;
  color: #444;
}
.register-link {
  color: #1976d2;
  text-decoration: none;
  font-weight: 600;
  transition: color 0.3s ease;
}
.register-link:hover {
  color: #0d47a1;
  text-decoration: underline;
}
```

Login.html:

```
<div class="login-container">
   <mat-card class="login-card">
     <mat-card-title>Login</mat-card-title>
     <mat-card-content>
       <form [formGroup]="loginForm" (ngSubmit)="onLogin()">
         <mat-form-field class="full-width" appearance="outline">
           <mat-label>Username</mat-label>
           <input matInput formControlName="username">
           <mat-error *ngIf="loginForm.controls['username'].hasError('required')">
             Username is required
```

```
        </mat-error>
      </mat-form-field>

      <mat-form-field class="full-width" appearance="outline">
        <mat-label>Password</mat-label>
        <input matInput type="password" formControlName="password">
        <mat-error *ngIf="loginForm.controls['password'].hasError('required')">
          Password is required
        </mat-error>
      </mat-form-field>

      <button mat-raised-button color="primary" class="submit-btn" type="submit"
[disabled]="loginForm.invalid">
        Submit
      </button>
    </form>

    <p *ngIf="errorMessage" class="error-text">
      {{ errorMessage }}
    </p>
    <p class="register-text">
      Don't have an account?
      <a routerLink="/register" class="register-link" >Register here</a>
    </p>
  </mat-card-content>
 </mat-card>
</div>
```

Login.ts

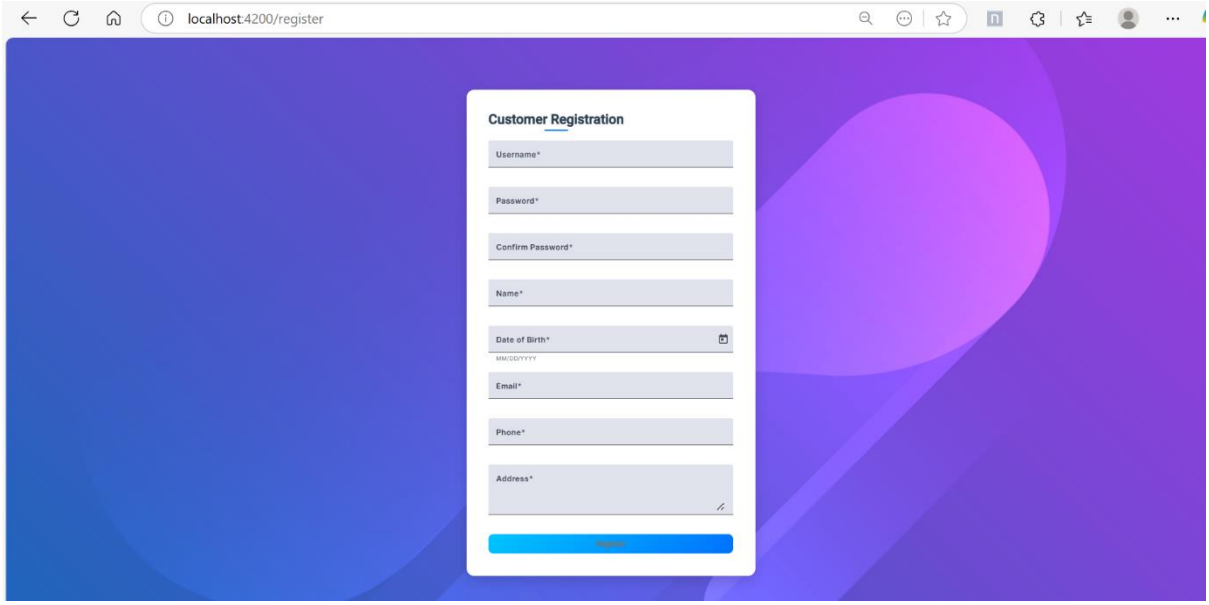
```
// logincomponent.ts
import { Component, Injectable } from '@angular/core';
import { Router, RouterLink } from '@angular/router';
import { FormBuilder, FormGroup, Validators, ReactiveFormsModule } from
'@angular/forms'; // Import ReactiveFormsModule
import { LoginRequestDto } from '../dtos/login-request.dto';
import { AuthService } from '../auth.services';
import { ApiResponse } from '../../../shared/api-response.interface';
import { LoginResponseDto } from '../dtos/loginResponse.dto';
import { CommonModule } from '@angular/common';

// Import Angular Material Modules
import { MatCardModule } from '@angular/material/card';
import { MatFormFieldModule } from '@angular/material/form-field';
import { MatInputModule } from '@angular/material/input';
import { MatButtonModule } from '@angular/material/button';

@Injectable({
 providedIn: 'root'
})

@Component({
 selector: 'app-login',
 templateUrl: './login.component.html',
 imports: [
  ReactiveFormsModule, // Add ReactiveFormsModule
  CommonModule,
  RouterLink,
```

```
    MatCardModule,        // Add MatCardModule
    MatFormFieldModule,   // Add MatFormFieldModule
    MatInputModule,       // Add MatInputModule
    MatButtonModule       // Add MatButtonModule
  ],
  styleUrls: ['./login.component.css']
})
export class LoginComponent {
  loginForm: FormGroup;
  errorMessage: string | null = null; // Declare errorMessage property and initialize it

  constructor(private fb: FormBuilder, private router: Router, private loginService:
AuthService) {
    this.loginForm = this.fb.group({
      username: ['', Validators.required],
      password: ['', Validators.required]
    });
  }

  loginRequest: LoginRequestDto = {
    username: '',
    password: ''
  };

  onLogin() {
    console.log('Login button clicked');
    this.errorMessage = null; // Clear any previous error message on new login attempt
    if (this.loginForm.valid) {
```

```
    this.loginRequest = this.loginForm.value;
    console.log('Attempting login with:', this.loginRequest);
    this.loginService.login(this.loginRequest).subscribe({
     next: (response: ApiResponse<LoginResponseDto>) => {
      if (response.data?.token) {
       localStorage.setItem('token', response.data.token);
      }

      console.log('Login successful:', response);
      if (response.data && response.data.role) {
       if (response.data.role === 'Admin') {
        this.router.navigate(['/admin/admin-dashboard']);
       } else if (response.data.role === 'Customer') {
        this.router.navigate(['/customer/customer-dashboard']);
       } else if (response.data.role === 'Agent') {
        this.router.navigate(['/agent/agent-dashboard']);
       } else {
        console.warn('Unknown role:', response.data.role);
        this.errorMessage = 'Login successful, but role is unknown. Please contact
support.';
       }
      } else {
       this.errorMessage = 'Login successful, but no role information received.';
      }
     },
     error: (error) => {
      console.error('Login failed:', error);
       if (error.status === 401 || error.status === 403) {
```

```
        this.errorMessage = 'Invalid username or password.';
      } else if (error.error && error.error.message) {
        this.errorMessage = error.error.message;
      } else {
        this.errorMessage = 'An unexpected error occurred. Please try again.';
      }
    }
  });
 }
}
}
```

Registration Page:



Registration.css

```
@import
url('https://fonts.googleapis.com/css2?family=Inter:wght@400;600;700&display=swap'
);


* {
```

```
  font-family: 'Inter', sans-serif;
  box-sizing: border-box;
  margin: 0;
  padding: 0;
}

/* Stylish Page Background */
body {
  min-height:200vh;
  display: flex;     /* Also contributes to layout, good for centering content */
  align-items: center;
  /* justify-content: center; */



  background-image: url('image.webp');
  background-size: cover;
  background-position: center;
  background-repeat: no-repeat;
  background-attachment: fixed;



}

/* Main Registration Card */
.registration-card {
  max-width: 600px;
  width: 90%;
  margin: 40px auto;
```

```
  padding: 30px;
  background: #ffffff;
  border-radius: 16px;
  box-shadow: 0 10px 25px rgba(0, 0, 0, 0.15);
  transition: transform 0.3s ease, box-shadow 0.3s ease;
}

.registration-card:hover {
  transform: translateY(-5px);
  box-shadow: 0 16px 35px rgba(0, 0, 0, 0.2);
}

/* Card Title */
.registration-card mat-card-title {
  font-size: 28px;
  font-weight: 700;
  color: #2c3e50;
  text-align: center;
  margin-bottom: 20px;
  position: relative;
}

.registration-card mat-card-title::after {
  content: '';
  display: block;
  height: 3px;
  width: 50px;
  background-color: #007bff;
```

```
  margin: 8px auto 0;
  border-radius: 4px;
}

/* Form Fields */
mat-form-field {
  width: 100%;
  margin-bottom: 20px;
}

/* Stylish Input Fields */
.mat-form-field-appearance-fill .mat-form-field-flex {
  background-color: #f5f8fa;
  border-radius: 6px;
  padding-left: 10px;
}

.mat-form-field-appearance-fill.mat-form-field-invalid .mat-form-field-flex {
  background-color: #fdecea;
}

/* Input Label */
mat-label {
  font-weight: 600;
  color: #374151;
}

/* Hint Text */
```

```
mat-hint {
  font-size: 12px;
  color: #6b7280;
}

/* Error Styling */
mat-error {
  font-size: 13px;
  color: #d32f2f;
  margin-top: 4px;
}

/* Submit Button */
button[type="submit"] {
  width: 100%;
  padding: 12px;
  font-size: 16px;
  font-weight: 600;
  color: #fff;
  background: linear-gradient(135deg, #00c6ff, #0072ff);
  border: none;
  border-radius: 10px;
  cursor: pointer;
  transition: all 0.3s ease;
  box-shadow: 0 6px 20px rgba(0, 114, 255, 0.3);
}

button[type="submit"]:hover:not([disabled]) {
```

```
  background: linear-gradient(135deg, #0072ff, #00c6ff);
  transform: translateY(-2px);
  box-shadow: 0 8px 24px rgba(0, 114, 255, 0.4);
}

button[disabled] {
  background-color: #cfd8dc !important;
  color: #777;
  box-shadow: none;
  cursor: not-allowed;
}

/* Responsive Layout */
@media (max-width: 600px) {
  .registration-card {
    margin: 20px;
    padding: 20px;
  }

  .registration-card mat-card-title {
    font-size: 24px;
  }

  button[type="submit"] {
    font-size: 15px;
  }
}
```

Registration.html:

```
<body><mat-card class="registration-card">
 <mat-card-header>
 <mat-card-title>Customer Registration</mat-card-title>
 </mat-card-header>
 <mat-card-content>
 <form [formGroup]="registrationForm" (ngSubmit)="onSubmit()">
 <mat-form-field appearance="fill">
 <mat-label>Username</mat-label>
 <input matInput formControlName="username" required maxlength="20">
 <mat-error *ngIf="registrationForm.get('username')?.hasError('required') &&
registrationForm.get('username')?.touched">
      Username is required.
 </mat-error>
 <mat-error *ngIf="registrationForm.get('username')?.hasError('pattern') &&
registrationForm.get('username')?.touched">
      Username must start with a letter and can only contain letters, numbers, and
underscores.
 </mat-error>
 <mat-error *ngIf="registrationForm.get('username')?.hasError('maxlength') &&
registrationForm.get('username')?.touched">
      Username cannot exceed 20 characters.
 </mat-error>
 </mat-form-field>

    <mat-form-field appearance="fill">
 <mat-label>Password</mat-label>
 <input matInput type="password" formControlName="password" required>
 <mat-error *ngIf="registrationForm.get('password')?.hasError('required') &&
registrationForm.get('password')?.touched">
```

```
        Password is required.
 </mat-error>
 <mat-error *ngIf="registrationForm.get('password')?.hasError('minlength') &&
registrationForm.get('password')?.touched">
        Password must be at least 6 characters long.
 </mat-error>
 <mat-error *ngIf="registrationForm.get('password')?.hasError('pattern') &&
registrationForm.get('password')?.touched">
        Password must contain at least one uppercase letter, one lowercase letter, one
number, and one special character.
 </mat-error>
 </mat-form-field>


     <mat-form-field appearance="fill">
 <mat-label>Confirm Password</mat-label>
 <input matInput type="password" formControlName="confirmPassword" required>
 <mat-error *ngIf="registrationForm.get('confirmPassword')?.hasError('required') &&
registrationForm.get('confirmPassword')?.touched">
        Confirm Password is required.
 </mat-error>
 <mat-error *ngIf="registrationForm.get('confirmPassword')?.hasError('mismatch') &&
registrationForm.get('confirmPassword')?.touched">
        Passwords do not match.
 </mat-error>
 <mat-error *ngIf="registrationForm.get('confirmPassword')?.hasError('minlength') &&
registrationForm.get('confirmPassword')?.touched">
        Confirm Password must be at least 6 characters long.
 </mat-error>
 <mat-error *ngIf="registrationForm.get('confirmPassword')?.hasError('pattern') &&
registrationForm.get('confirmPassword')?.touched">
```

Confirm Password must contain at least one uppercase letter, one lowercase letter, one number, and one special character.

</mat-error>

</mat-form-field>

<mat-form-field appearance="fill">

<mat-label>Name</mat-label>

<input matInput formControlName="name" required>

<mat-error *ngIf="registrationForm.get('name')?.hasError('required') && registrationForm.get('name')?.touched">

Name is required.

</mat-error>

<mat-error *ngIf="registrationForm.get('name')?.hasError('pattern') && registrationForm.get('name')?.touched">

Name can only contain letters and spaces.

</mat-error>

</mat-form-field>

<mat-form-field appearance="fill">

<mat-label>Date of Birth</mat-label>

<input matInput [matDatepicker]="picker" [max]="maxDate" formControlName="dateOfBirth" required>

<mat-hint>MM/DD/YYYY</mat-hint>

<mat-datepicker-toggle matIconSuffix [for]="picker"></mat-datepicker-toggle>

<mat-datepicker #picker></mat-datepicker>

<mat-error *ngIf="registrationForm.get('dateOfBirth')?.hasError('required') && registrationForm.get('dateOfBirth')?.touched">

Date of Birth is required.

</mat-error>

```
 <mat-error *ngIf="registrationForm.get('dateOfBirth')?.hasError('futureDate') &&
registrationForm.get('dateOfBirth')?.touched">
      Date of Birth cannot be in the future.
 </mat-error>
 </mat-form-field>

    <mat-form-field appearance="fill">
 <mat-label>Email</mat-label>
 <input matInput type="email" formControlName="email" required>
 <mat-error *ngIf="registrationForm.get('email')?.hasError('required') &&
registrationForm.get('email')?.touched">
      Email is required.
 </mat-error>
 <mat-error *ngIf="registrationForm.get('email')?.hasError('email') &&
registrationForm.get('email')?.touched">
      Invalid email format.
 </mat-error>
 </mat-form-field>

    <mat-form-field appearance="fill">
 <mat-label>Phone</mat-label>
 <input matInput type="tel" formControlName="phone" required maxlength="10"
minlength="10">
 <mat-error *ngIf="registrationForm.get('phone')?.hasError('required') &&
registrationForm.get('phone')?.touched">
      Phone number is required.
 </mat-error>
 <mat-error *ngIf="registrationForm.get('phone')?.hasError('pattern') &&
registrationForm.get('phone')?.touched">
      Phone number can only contain numbers.
```

```
 </mat-error>
 <mat-error *ngIf="registrationForm.get('phone')?.hasError('maxlength') &&
registrationForm.get('phone')?.touched">
      Phone number cannot exceed 10 numbers.
 </mat-error>
 <mat-error *ngIf="registrationForm.get('phone')?.hasError('minlength') &&
registrationForm.get('phone')?.touched">
      Phone number minimum contains 10 numbers.
 </mat-error>
 </mat-form-field>

    <mat-form-field appearance="fill">
 <mat-label>Address</mat-label>
 <textarea matInput formControlName="address" required rows="3"></textarea>
 <mat-error *ngIf="registrationForm.get('address')?.hasError('required') &&
registrationForm.get('address')?.touched">
      Address is required.
 </mat-error>
 </mat-form-field>

    <button mat-raised-button color="primary" type="submit"
[disabled]="registrationForm.invalid">Register</button>
 </form>
 </mat-card-content>
 </mat-card>
</body>
```

Registration.ts:

```
import { CommonModule } from '@angular/common';
import { Component, OnInit } from '@angular/core';
```

```
import { FormBuilder, FormGroup, Validators, AbstractControl, ValidationErrors,
FormsModule, ReactiveFormsModule } from '@angular/forms';

import { MatSnackBar, MatSnackBarModule } from '@angular/material/snack-bar';

import { Router, RouterModule } from '@angular/router';

// Import Angular Material modules

import { MatCardModule } from '@angular/material/card';

import { MatFormFieldModule } from '@angular/material/form-field';

import { MatInputModule } from '@angular/material/input';

import { MatButtonModule } from '@angular/material/button';

import { MatDatepickerModule } from '@angular/material/datepicker';

import { MatNativeDateModule } from '@angular/material/core';

import { AuthService } from '../auth.services';

import { CustomerRegisterRequestDto } from
'../../customer/dtos/customerRegisterRequest.dto';



@Component({

 selector: 'app-register',

 imports: [

  CommonModule,

  FormsModule,

  ReactiveFormsModule,

  MatCardModule,

  MatFormFieldModule,

  MatInputModule,

  MatButtonModule,

  MatDatepickerModule,

  MatNativeDateModule,

  MatSnackBarModule,
```

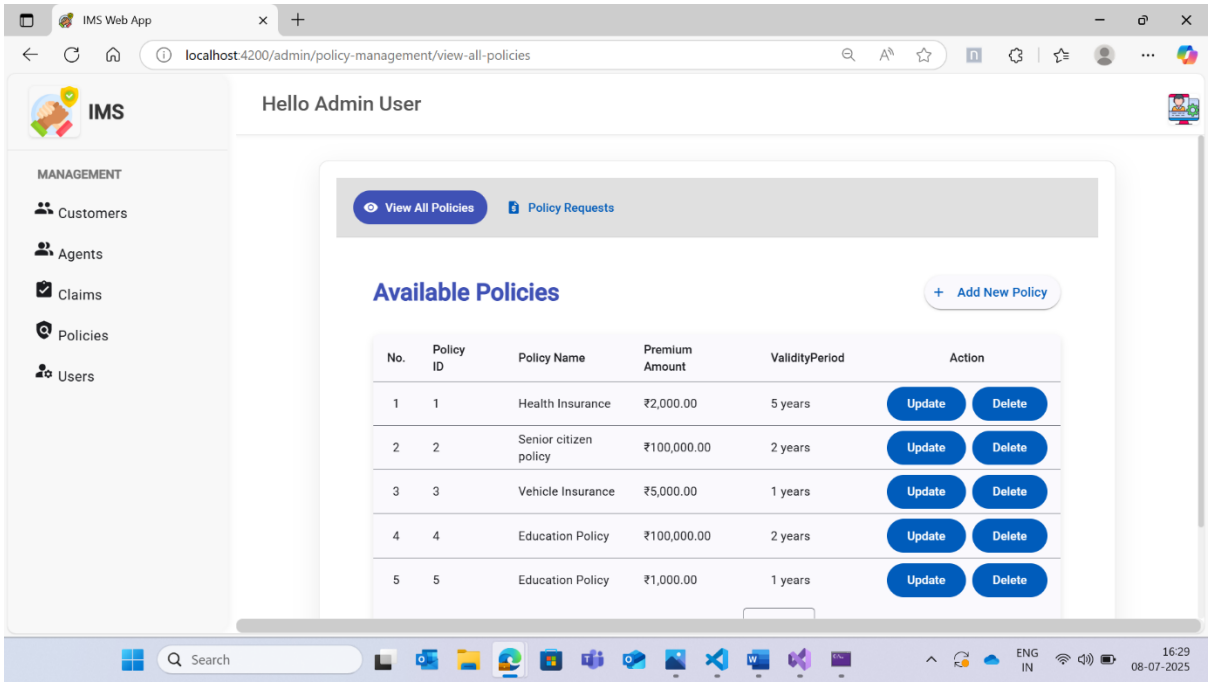
```
    RouterModule,
  ],
  templateUrl: './register.component.html',
  styleUrls: ['./register.component.css']
})

export class RegistrationFormComponent implements OnInit {
  registrationForm!: FormGroup;
  maxDate: Date; // Property to hold the current date
  constructor(private fb: FormBuilder, private _snackBar: MatSnackBar, private
loginService: AuthService, private router: Router) {
    this.maxDate = new Date(); // Initialize maxDate to the current date
  }
  ngOnInit(): void {
    this.registrationForm = this.fb.group({
      username: ['', [
        Validators.required,
        Validators.pattern(/^[a-zA-Z][a-zA-Z0-9_]*$/),
        Validators.maxLength(20)
      ]],
      password: ['', [
        Validators.required,
        Validators.minLength(6),
        Validators.pattern(/^(?=.*[a-z])(?=.*[A-Z])(?=.*\d)(?=.*[@$!%*?&])[A-Za-
z\d@$!%*?&]{6,}$/)
      ]],
      confirmPassword: ['', [
        Validators.required,
        Validators.minLength(6),
```

```
      Validators.pattern(/^(?=.*[a-z])(?=.*[A-Z])(?=.*\d)(?=.*[@$!%*?&])[A-Za-
z\d@$!%*?&]{6,}$/)
    ]],
    name: ['', [
     Validators.required,
     Validators.pattern(/^[a-zA-Z\s]+$/)
    ]],
    dateOfBirth: ['', [
     Validators.required,
     this.futureDateValidator // Add the custom validator here
    ]],

    email: ['', [Validators.required, Validators.email]],
    phone: ['', [
     Validators.required,
     Validators.pattern(/^\d+$/),
     Validators.minLength(10),
     Validators.maxLength(10)
    ]],

    address: ['', Validators.required]
  }, { validators: this.passwordMatchValidator });
 }
 futureDateValidator(control: AbstractControl): ValidationErrors | null {
  const selectedDate = control.value;
  if (!selectedDate) {
   return null; // Don't validate if no date is selected (handled by Validators.required)
  }
```

```
    const today = new Date();
    // Normalize today's date to midnight to compare only dates, not times
    today.setHours(0, 0, 0, 0);
    // Convert selectedDate to Date object if it's a string or other format
    const dateToCompare = new Date(selectedDate);
    dateToCompare.setHours(0, 0, 0, 0); // Normalize selected date
    if (dateToCompare.getTime() > today.getTime()) {
      return { futureDate: true }; // Return error if selected date is in the future
    }
    return null; // No error
  }
  passwordMatchValidator(control: AbstractControl): ValidationErrors | null {
    const password = control.get('password');
    const confirmPassword = control.get('confirmPassword');
    if (!password || !confirmPassword) {
      return null;
    }
    if (confirmPassword.hasError('mismatch') && password.value ===
confirmPassword.value) {
      confirmPassword.setErrors(null);
    }
    if (password.value !== confirmPassword.value) {
      confirmPassword.setErrors({ mismatch: true });
      return { mismatch: true };
    }
    return null;
  }
  registerRequest: CustomerRegisterRequestDto = {} as CustomerRegisterRequestDto;
```

```
 onSubmit(): void {
  if (this.registrationForm.valid) {
   this.registerRequest = this.registrationForm.value;
   console.log('Form Submitted!', this.registerRequest);
   this.loginService.register(this.registerRequest).subscribe({
    next: (response) => {
     console.log('Registration successful:', response);
     this._snackBar.open('Registration successful!', 'Close', {
      duration: 3000,
     });
      this.router.navigate(['/login']);
    },
     error: (error) => {
     console.error('Registration failed:', error);
     let errorMessage = 'Registration failed. Please try again.';
     if (error.error && error.error.message) {
        errorMessage = error.error.message;
     }
     this._snackBar.open(errorMessage, 'Close', {
      duration: 5000,
      panelClass: ['error-snackbar']
     });
    }
   });
  } else {
   this._snackBar.open('Please correct the errors in the form.', 'Close', {
    duration: 3000,
    panelClass: ['error-snackbar']
```

```
    });
    this.registrationForm.markAllAsTouched();
  }
 }
}
```



View-All Policies[Policy-management]:

View-All Policies.css:

```
.policy-management-container {
  width: 100%;
    max-width: 900px;
    padding:20px;
    margin: 10px auto;
    background-color: #ffffff;
    border-radius: 12px;
    box-shadow: 0 10px 30px rgba(0, 0, 0, 0.08);
    overflow: hidden;
    border: 1px solid #e5e5e5;
```

```
}

.policy-nav-links {
  display: flex; /* Makes the buttons appear in a row */
  flex-wrap: wrap; /* Allows buttons to wrap to the next line on smaller screens */
  gap: 10px; /* Space between buttons */
  padding: 15px 20px; /* Padding around the navigation buttons */
  background-color: #e0e0e0; /* Light background for the navigation bar */
  border-bottom: 1px solid #ccc; /* Separator from content */
  justify-content: flex-start; /* Align items to the start */
  align-items: center; /* Vertically center items */
  flex-shrink: 0; /* Prevent these links from shrinking */
}

.policy-nav-links .mat-mdc-button,
.policy-nav-links .mat-mdc-raised-button {
  min-width: unset; /* Allow buttons to size based on content */
  padding: 8px 16px;
}

/* Style for active navigation button */
.policy-nav-links .mat-button.active-button,
.policy-nav-links .mat-mdc-button.active-button {
  background-color: #3f51b5; /* Primary color background */
  color: white; /* White text for active button */
  font-weight: 500;
}
```

```
.policy-nav-links .mat-button.active-button .mat-icon,
.policy-nav-links .mat-mdc-button.active-button .mat-icon {
  color: white; /* Ensure icon is white when button is active */
}

.policy-content-area {
  flex-grow: 1; /* Allows the content area to take up remaining vertical space */
  padding: 20px;
  overflow-y: auto; /* Adds vertical scroll if content overflows */
  box-sizing: border-box; /* Include padding in the element's total width and height */
  /*
  Crucial for static height: Set a minimum height for the content area.
  Adjust this value based on the expected maximum content height,
  or the height you want it to always occupy.
  */
  min-height: 400px; /* Example: Ensure it's at least 400px tall */
  /* Alternatively, if you want it to fill all available space below the nav, and then scroll: */
  /* height: 100%; */ /* This would make it fill the remaining space of its parent (policy-management-container) */
}


/* Optional: Adjust for smaller screens */
@media (max-width: 599px) {
  .policy-nav-links {
    flex-direction: column; /* Stack buttons vertically on small screens */
    align-items: stretch; /* Make buttons fill the width */
  }
```

```
  .policy-nav-links .mat-button,
  .policy-nav-links .mat-mdc-button {
    width: 100%; /* Make buttons full width */
    margin-bottom: 5px; /* Add some vertical spacing */
  }
}
```

View-All Policies.html:

```
<div class="policy-management-container">
  <div class="policy-nav-links">
    <a mat-button routerLink="/admin/policy-management/view-all-policies"
routerLinkActive="active-button">
      <mat-icon>visibility</mat-icon> View All Policies
    </a>
    <a mat-button routerLink="/admin/policy-management/requested-policies"
routerLinkActive="active-button">
      <mat-icon>request_quote</mat-icon> Policy Requests
    </a>
  </div>

  <div class="policy-content-area">
    <router-outlet></router-outlet>
  </div>
</div>
```

View-All Policies.ts:

```
import { Component, OnInit } from '@angular/core';
import { CommonModule } from '@angular/common';
```

```
import { MatButtonModule } from '@angular/material/button';
import { MatIconModule } from '@angular/material/icon';
import { RouterModule, Router, NavigationEnd } from '@angular/router'; // Import Router and NavigationEnd
import { filter } from 'rxjs/operators'; // Import filter for RxJS

@Component({
 selector: 'app-policy-management',
 standalone: true,
 imports: [
  CommonModule,
  MatButtonModule,
  MatIconModule,
  RouterModule // Important for routerLink
 ],
 templateUrl: './policy-management.component.html',
 styleUrl: './policy-management.component.css'
})
export class PolicyManagementComponent implements OnInit {

 constructor(private router: Router) { } // Inject Router

 ngOnInit(): void {
  // --- Logic for automatic navigation to View All Policies ---
  // Subscribe to router events to check the current URL
  this.router.events.pipe(
   filter(event => event instanceof NavigationEnd)
  ).subscribe((event: NavigationEnd) => {
```

```
    // If the current URL is the base policy-management path, redirect to view-all-policies
    if (event.urlAfterRedirects === '/admin/policy-management' || event.urlAfterRedirects === '/admin/policy-management/') {
      this.router.navigate(['/admin/policy-management/view-all-policies']);
    }
  });

  // Also handle the case where the component initializes directly on /admin/policy-management
  if (this.router.url === '/admin/policy-management' || this.router.url === '/admin/policy-management/') {
    this.router.navigate(['/admin/policy-management/view-all-policies']);
  }
 }
}
```

PoliciesController.cs:

```
using System.Collections.Generic;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using IMSIntrim.Application.Interfaces;
using IMSIntrim.Applications.DTOs;
using IMSIntrim.Domain.Models;
using IMSIntrim.Infrastructure.Persistance;
using IMSIntrim.Applications.Interfaces;

namespace IMSIntrim.Controllers
```

```
{
   [Route("api/policies")]
   [ApiController]
   public class PoliciesController : ControllerBase
   {
      private readonly InsuranceDbContext _context;
      private readonly IPolicyServices _policyService;
      private readonly ITokenService _tokenService;

      public PoliciesController(InsuranceDbContext context, IPolicyServices
policyServices,ITokenService tokenService)
      {
         _context = context;
         _policyService = policyServices;
         _tokenService = tokenService;
      }

      [HttpGet("available")]
      public async Task<ActionResult<PagedResult<AvailablePolicyResponseDto>>>
GetAllAvailablePolicies(int page = 1, int size = 10)
      {
         var result = await _policyService.GetAllAvailablePoliciesAsync(page, size);
         if (!result.IsSuccess) return NotFound(result);
         return Ok(result);
      }

      //[HttpGet("requests/admin")]
      //[Authorize(Roles = Roles.Admin)]
```

```
    //public async
Task<ActionResult<PagedResult<PolicyRequestStatusResponseDto>>>
GetAllPolicyRequests([FromQuery] int page, [FromQuery] int size)
    //{
    //    var result = await _policyService.GetAllPolicyRequestsAsync(page, size);
    //    if (!result.IsSuccess) return NotFound(result);
    //    return Ok(result);
    //}

    //new code
    [HttpGet("requests/admin")]
    [Authorize(Roles = Roles.Admin)]
    public async Task<IActionResult> GetPolicyRequests([FromQuery] int page,
[FromQuery] int size)
    {
        var result = await _policyService.GetAllPolicyRequestsAsync(page, size);

        if (!result.IsSuccess)
            return BadRequest(result.Message);

        return Ok(result);
    }

    [HttpPost("available")]
    [Authorize(Roles = Roles.Admin)]
    public async Task<IActionResult> AddAvailablePolicy([FromBody]
AvailablePolicyRequestDto dto)
    {
        var result = await _policyService.AddAvailablePolicyAsync(dto);
```

```
        if (!result.IsSuccess) return BadRequest(result);
        return Ok(result);
    }

    [HttpPut("{policyId}/available")]
    [Authorize(Roles = Roles.Admin)]
    public async Task<IActionResult> UpdateAvailablePolicy([FromBody]
AvailablePolicyRequestDto dto, int policyId)
    {
        var result = await _policyService.UpdateAvailablePolicyAsync(dto, policyId);
        if (!result.IsSuccess) return BadRequest(result);
        return Ok(result);
    }

    [HttpDelete("{id}/available")]
    [Authorize(Roles = Roles.Admin)]
    public async Task<IActionResult> DeleteAvailablePolicy(int id)
    {
        var result = await _policyService.DeleteAvailablePolicyAsync(id);
        if (!result.IsSuccess) return BadRequest(result);
        return Ok(result);
    }

    [HttpPut("{requestId}/approve")]
    [Authorize(Roles = Roles.Admin)]
    public async Task<IActionResult> ApprovePolicyRequest([FromBody]
AssignAgentRequestDto dto, int requestId)
    {
        var result = await _policyService.ApprovePolicyRequestAsync(dto, requestId);
```

```
        if (!result.IsSuccess) return BadRequest(result);
        return Ok(result);
    }

    [HttpPost("{requestId}/reject")]
    [Authorize(Roles = Roles.Admin)]
    public async Task<IActionResult> RejectPolicyRequest(int requestId)
    {
        var result = await _policyService.RejectPolicyRequestAsync(requestId);
        if (!result.IsSuccess) return BadRequest(result);
        return Ok(result);
    }

    [HttpGet("available/{policyId}")]
    public async Task<ActionResult<AvailablePolicyResponseDto>>
GetAvailablePolicyByIdAsync(int policyId)
    {
        var result = await _policyService.GetAvailablePolicyByIdAsync(policyId);
        if (!result.IsSuccess) return NotFound(result);
        return Ok(result);
    }

    [HttpGet("assigned")]
    [Authorize(Roles = Roles.Agent)]
    public async Task<IActionResult> GetAssignedPolicies()
    {
        var policies = await _policyService.GetAssignedPoliciesAsync();
        if (!policies.IsSuccess) return NotFound(policies);
```

```
        return Ok(policies);
    }

    [HttpGet("customer-registered")]
    [Authorize(Roles = Roles.Customer)]
    public async Task<IActionResult> GetRegisteredPolicies()
    {
        var policies = await _policyService.GetRegisteredPoliciesAsync();
        return Ok(policies);
    }

    [HttpGet("customer-requests")]
    [Authorize(Roles = Roles.Customer)]
    public async Task<IActionResult> GetPolicyRequests()
    {
        var requests = await _policyService.GetPolicyRequestsAsync();
        if (!requests.IsSuccess) return NotFound(requests);
        return Ok(requests);
    }

    [HttpPost("request")]
    [Authorize(Roles = Roles.Customer)]
    public async Task<IActionResult> RequestPolicy([FromBody] PolicyRequestDto
requestDto)
    {
        var loggedInCustomerId=_tokenService.GetCustomerIdFromCurrentRequest();
        if (loggedInCustomerId==null)
        {
```

```
            return Unauthorized("User ID not found in token.");
        }

        if (requestDto.CustomerId != loggedInCustomerId)
        {
            return BadRequest("Invalid policy or unauthoritized Request");
        }

        //var existingPolicy = await
_policyService.GetExistingPolicyAsync(requestDto.CustomerId,
requestDto.AvailablePolicyName);
        var existingPolicy = await _policyService.GetExistingPolicyByNameAsync(
        requestDto.CustomerId,
        requestDto.AvailablePolicyName);
        if (existingPolicy != null)
        {
            return Conflict("Policy request already exists for this customer.");
        }

        var result = await _policyService.RequestPolicyAsync(requestDto);
        if (!result.IsSuccess) return BadRequest(result);
        return Ok(result);
    }

    [HttpGet("request/{requestId}")]
    [Authorize(Roles = Roles.Customer)]
    public async Task<ActionResult<PolicyRequestStatusResponseDto>>
GetPolicyRequestByIdAsync(int requestId)
    {
```

```
        var result = await _policyService.GetPolicyRequestByIdAsync(requestId);
        if (!result.IsSuccess) return NotFound(result);
        return Ok(result);
      }
   }
}
```

IPolicyRepository.cs:

```
using IMSIntrim.Domain.Models;
using IMSIntrim.Shared.Common;

namespace IMSIntrim.Domain.Interfaces
{
   public interface IPolicyRepository
   {
      public Task<OperationResult<IEnumerable<Policy>>> GetAllAsync();
      public Task<OperationResult<Policy?>> GetByIdAsync(int id);
      public Task<OperationResult<IEnumerable<Policy>>> GetByCustomerIdAsync(int
customerId);
      public Task<OperationResult<IEnumerable<Policy>>> GetByAgentIdAsync(int
agentId);
      public Task<OperationResult<bool>> AddAsync(Policy policy);
      public Task<OperationResult<bool>> UpdateAsync(Policy policy);
      public Task<OperationResult<bool>> DeleteAsync(int id);
   }
}
```

PolicyService.cs:

```
using AutoMapper;
using IMSIntrim.Application.Interfaces;
```

```
using IMSIntrim.Applications.DTOs;
using IMSIntrim.Applications.Interfaces;
using IMSIntrim.Domain.Interfaces;
using IMSIntrim.Domain.Models;
using IMSIntrim.Infrastructure.Persistance;
using IMSIntrim.Infrastructure.Persistance.Repositories;
using IMSIntrim.Shared.Common;
using Microsoft.AspNetCore.Http.HttpResults;
using Microsoft.EntityFrameworkCore;

namespace IMSIntrim.Application.Services
{
   public class PolicyServices : IPolicyServices
   {
      private readonly ITokenService _tokenService;
      private readonly InsuranceDbContext _dbContext;

     private readonly IPolicyRequestRepository _requestRepo;
      private readonly ICustomerRepository _customerRepo;
      private readonly IAgentRepository _agentRepo;
      private readonly IAvailablePolicyRepository _availablePolicyRepo;
      private readonly IPolicyRequestRepository _policyRequestRepo;
      private readonly IClaimRepository _claimRepo;
      private readonly IPolicyRepository _policyRepo;
      private readonly IUserRepository _userRepo;
      private readonly IRoleRepository _roleRepo;
      private readonly IUserRoleRepository _userRoleRepo;
      private readonly IPasswordHasherService _passwordHasherService;
```

```
    private readonly IAuthServices _authService;
    private readonly IMapper _mapper;
    //private readonly ILogger<AdminService> _logger;
    private readonly INotificationRepository _notificationRepo;
    public PolicyServices
      (

    ICustomerRepository customerRepo,
      IAgentRepository agentRepo,
      IAvailablePolicyRepository availablePolicyRepo,
      IPolicyRequestRepository policyRequestRepo,
      IClaimRepository claimRepo,
      IPolicyRepository policyRepo,
      IAuthServices authService,
      IUserRepository userRepository,
      IPasswordHasherService passwordHasher,
      IRoleRepository roleRepository,
      IUserRoleRepository userRoleRepository,
      IAgentRepository agentRepository,
     INotificationRepository notificationRepository,
      ITokenService tokenService,
      InsuranceDbContext dbContext,
    // ILogger<AdminService> logger,

    IMapper mapper
      )
    {
      _customerRepo = customerRepo;
```

```
        _agentRepo = agentRepo;
        _availablePolicyRepo = availablePolicyRepo;
        _policyRequestRepo = policyRequestRepo;
        _claimRepo = claimRepo;
        _policyRepo = policyRepo;
        _authService = authService;
        _userRepo = userRepository;
        _passwordHasherService = passwordHasher;
        _roleRepo = roleRepository;
        _userRoleRepo = userRoleRepository;
        _agentRepo = agentRepository;
        _notificationRepo = notificationRepository;
        _mapper = mapper;
        _tokenService = tokenService;
        _dbContext = dbContext;
       //   _logger = logger;
      }
      public async Task<OperationResult<bool>>
AddAvailablePolicyAsync(AvailablePolicyRequestDto dto)
      {
        var policy = _mapper.Map<AvailablePolicy>(dto);

        await _availablePolicyRepo.AddAsync(policy);
        return OperationResult<bool>.Success(true, "New Policy Launched
Successfully");

      }
```

```
    public async Task<OperationResult<bool>>
ApprovePolicyRequestAsync(AssignAgentRequestDto dto, int requestId)
    {
        var result = await _policyRequestRepo.GetByIdAsync(requestId);



        if (!result.IsSuccess)
        {
            return OperationResult<bool>.Failure($"Policy request with ID '{requestId}' not
found.");
        }

        PolicyRequest? policyRequest = result.Data;



        var updateResult = await _policyRequestRepo.UpdateStatusAsync(requestId,
"Approved");

        if (!updateResult.IsSuccess)
        {
            return OperationResult<bool>.Failure($"Failed to update the status of policy
request with ID '{requestId}' to 'Approved'.");
        }

        if (policyRequest != null && policyRequest.AvailablePolicy != null)
        {
            var newPolicy = _mapper.Map<Policy>(dto);
            newPolicy.IssuedDate = DateTime.UtcNow;
```

```
        newPolicy.ExpiryDate =
DateTime.UtcNow.AddYears(policyRequest.AvailablePolicy.ValidityPeriod);
        _mapper.Map(policyRequest, newPolicy);

        var response = await _policyRepo.AddAsync(newPolicy);

        if (!response.IsSuccess)
        {
          return OperationResult<bool>.Failure($"Failed to register a new policy for
Customer ID '{policyRequest.CustomerId}' and Available Policy ID
'{policyRequest.AvailablePolicyId}'.");
        }

        var customerNotification = new Notification
        {
          CustomerId = policyRequest.CustomerId,
          AgentId = null,
          Message = $"Congratulations! Your policy request for Registering a new policy
of Available Policy ID {policyRequest.AvailablePolicyId} with Request ID
{policyRequest.RequestId} has been approved and Assigned with an
Agent{dto.AgentId}. Please review the details and proceed with the next steps.",
          CreatedAt = DateTime.UtcNow
        };

        var agentNotification = new Notification
        {
          CustomerId = null,
          AgentId = dto.AgentId,
```

```
            Message = $"Dear Agent, you have been assigned a new policy for Customer ID {policyRequest.CustomerId}. The details are as follows:Available Policy ID: {policyRequest.AvailablePolicyId}, Policy Request ID: {policyRequest.RequestId}. Please review and take the necessary actions."
,
            CreatedAt = DateTime.UtcNow
        };



        var customerNotificationResult = await _notificationRepo.AddAsync(customerNotification);
        var agentNotificationResult = await _notificationRepo.AddAsync(agentNotification);


        if (!customerNotificationResult.IsSuccess || !agentNotificationResult.IsSuccess)
        {
            return OperationResult<bool>.Failure("Failed to add notifications");
        }
      }



      return OperationResult<bool>.Success(true, "policy added and Notification Sent");


    }


    public async Task<OperationResult<bool>> DeleteAvailablePolicyAsync(int id)
    {
      await _availablePolicyRepo.DeleteAsync(id);
```

```
        return OperationResult<bool>.Success(true, "Available Policy deleted
successfully");


    }


    public async Task<OperationResult<PagedResult<AvailablePolicyResponseDto>>>
GetAllAvailablePoliciesAsync(int page, int size)
    {
        //_logger.LogInformation("GetAllAvailablePoliciesAsync started");
        var allAvailablePolicies = await _availablePolicyRepo.GetAllAsync();


        if (!allAvailablePolicies.IsSuccess)
        {
            return OperationResult<PagedResult<AvailablePolicyResponseDto>>.Failure("
Failed To fetch the Available Policy");
        }


        var allAvailablePoliciesList = allAvailablePolicies.Data?.ToList();


        if (allAvailablePoliciesList == null)
        {
            return OperationResult<PagedResult<AvailablePolicyResponseDto>>.Failure("
There is no data  in the Available Policy List");
        }



        var paged = allAvailablePoliciesList.Skip((page - 1) * size).Take(size).ToList();
        if (paged != null && paged.Any())
        {
```

```
            var policyResult = new PagedResult<AvailablePolicyResponseDto>
            {

                Items = paged.Select(p => _mapper.Map<AvailablePolicyResponseDto>(p)),

                PageNumber = page,
                PageSize = size,
                TotalCount = allAvailablePoliciesList.Count

            };
            return
OperationResult<PagedResult<AvailablePolicyResponseDto>>.Success(policyResult);




        }
        return OperationResult<PagedResult<AvailablePolicyResponseDto>>.Failure("
There is no Available Policy in the specific page");

    }



    public async
Task<OperationResult<PagedResult<PolicyRequestStatusResponseDto>>>
GetAllPolicyRequestsAsync(int page, int size)
    {
        var pagedRequests = await _policyRequestRepo.GetAllPagedAsync(page, size);
        if (!pagedRequests.IsSuccess || pagedRequests.Data == null)
        {
```

```
            return
OperationResult<PagedResult<PolicyRequestStatusResponseDto>>.Failure("Failed to
fetch policy requests");
        }

        var mapped =
_mapper.Map<List<PolicyRequestStatusResponseDto>>(pagedRequests.Data.Items);

        var result = new PagedResult<PolicyRequestStatusResponseDto>
        {
            Items = mapped,
            PageNumber = page,
            PageSize = size,
            TotalCount = pagedRequests.Data.TotalCount
        };

        return
OperationResult<PagedResult<PolicyRequestStatusResponseDto>>.Success(result);
    }

    public async
Task<OperationResult<IEnumerable<AgentAssignedPolicyResponseDto>>>
GetAssignedPoliciesAsync()
    {
        int? agentId = _tokenService.GetAgentIdFromCurrentRequest();

        if (agentId == null)
        {
```

```
            return
OperationResult<IEnumerable<AgentAssignedPolicyResponseDto>>.Failure("Unable to
Get the AgentId from the Jwt Token");
        }
        var result = await _policyRepo.GetByAgentIdAsync((int)agentId);
        if (result.Data?.Any() != true)
        {
            return
OperationResult<IEnumerable<AgentAssignedPolicyResponseDto>>.Failure("No
policies found for the specified agent.");
        }
        var policies = result.Data;
        if (policies?.Any() != true)
        {
            return
OperationResult<IEnumerable<AgentAssignedPolicyResponseDto>>.Failure("No
Policies Found");
        }
        var assignedPolicies = policies.Select(p =>
_mapper.Map<AgentAssignedPolicyResponseDto>(p)).ToList();

        return
OperationResult<IEnumerable<AgentAssignedPolicyResponseDto>>.Success(assigned
Policies);

    }

    public async Task<OperationResult<AvailablePolicyResponseDto>>
GetAvailablePolicyByIdAsync(int policyId)
    {
        var availablePolicy = await _availablePolicyRepo.GetByIdAsync(policyId);
```

```
        if (!availablePolicy.IsSuccess)
        {
            return OperationResult<AvailablePolicyResponseDto>.Failure($"  Failed To fetch the Available Policy of Id{policyId}");

        }
        var availablePolicyResult = availablePolicy.Data;
        var result = _mapper.Map<AvailablePolicyResponseDto>(availablePolicyResult);
        return OperationResult<AvailablePolicyResponseDto>.Success(result);
    }

    public async Task<OperationResult<PolicyRequestStatusResponseDto>> GetPolicyRequestByIdAsync(int requestId)
    {
        var loggedInCustomerId = _tokenService.GetCustomerIdFromCurrentRequest();
        if (loggedInCustomerId == null)
        {
            return OperationResult<PolicyRequestStatusResponseDto>.Failure("Unauthorized: Customer ID not found.");
        }

        // Fetch the request details
        var request = await _policyRequestRepo.GetByIdAsync(requestId);
        if (!request.IsSuccess)
        {
            return OperationResult<PolicyRequestStatusResponseDto>.Failure($"Failed to fetch the request with ID {requestId}");
        }
```

```
        var requestResult = request.Data;


        // Validate ownership: Ensure request belongs to logged-in customer
        if (requestResult.CustomerId != loggedInCustomerId)
        {
            return
OperationResult<PolicyRequestStatusResponseDto>.Failure("Unauthorized: You can
only access your own policy requests.");
        }
        // Map and return response
        var result = _mapper.Map<PolicyRequestStatusResponseDto>(requestResult);
        return OperationResult<PolicyRequestStatusResponseDto>.Success(result);
    }


    public async
Task<OperationResult<IEnumerable<PolicyRequestStatusResponseDto>>>
GetPolicyRequestsAsync()
    {
        int? customerId = _tokenService.GetCustomerIdFromCurrentRequest();
        if (customerId == null)
        {
            return
OperationResult<IEnumerable<PolicyRequestStatusResponseDto>>.Failure("Unable to
Get the CustomerId from the Jwt Token");
        }
        var requests = await
_policyRequestRepo.GetByCustomerIdAsync((int)customerId);
        if (!requests.IsSuccess)
        {
```

```
            return
OperationResult<IEnumerable<PolicyRequestStatusResponseDto>>.Failure("Failed to
retrieve policy requests .");
        }
        var result = requests.Data;
        if (result == null)
        {
            return
OperationResult<IEnumerable<PolicyRequestStatusResponseDto>>.Failure("No policy
requests found.");
        }

        var policyRequests = result.Select(r =>
_mapper.Map<PolicyRequestStatusResponseDto>(r));

        return
OperationResult<IEnumerable<PolicyRequestStatusResponseDto>>.Success(policyRe
quests);

    }

    public async
Task<OperationResult<IEnumerable<CustomerPoliciesResponseDto>>>
GetRegisteredPoliciesAsync()
    {
        int? customerId = _tokenService.GetCustomerIdFromCurrentRequest();
        if (customerId == null)
        {
            return
OperationResult<IEnumerable<CustomerPoliciesResponseDto>>.Failure("Unable to
Get the CustomerId from the Jwt Token");
```

```
        }
        var existingPolicies = await _policyRepo.GetByCustomerIdAsync((int)customerId);
        if (!existingPolicies.IsSuccess)
        {
            return
OperationResult<IEnumerable<CustomerPoliciesResponseDto>>.Failure("Failed to
retrieve Registered policies for the customer .");
        }
        var policyResult = existingPolicies.Data;
        if (policyResult == null)
        {
            return
OperationResult<IEnumerable<CustomerPoliciesResponseDto>>.Failure("Registered
Policies not found .");
        }

        var Registeredpolicies = policyResult.Select(p =>
_mapper.Map<CustomerPoliciesResponseDto>(p));
        Registeredpolicies = policyResult.Select(p => new CustomerPoliciesResponseDto
        {
            PolicyId = p.PolicyId,
            AvailablePolicyName = p.AvailablePolicy.Name,
            AgentName = p.Agent.Name,
            AgentContact = p.Agent.ContactInfo,
            IssuedDate = p.IssuedDate,
            ExpiryDate = p.ExpiryDate,
            PremiumAmount = p.AvailablePolicy.BasePremium // ← magic line!
        });
```

```
        return
OperationResult<IEnumerable<CustomerPoliciesResponseDto>>.Success(Registeredp
olicies);
    }

    public async Task<OperationResult<bool>> RejectPolicyRequestAsync(int
requestId)
    {
        var updateResult = await _policyRequestRepo.UpdateStatusAsync(requestId,
"Rejected");

        if (!updateResult.IsSuccess)
        {
            return OperationResult<bool>.Failure("Failed to update the policy request
status");
        }

        var result = await _policyRequestRepo.GetByIdAsync(requestId);

        if (!result.IsSuccess)
        {
            return OperationResult<bool>.Failure("Policy request not found");
        }

        PolicyRequest? policyRequest = result.Data;

        if (policyRequest != null)
        {
```

```
            var customerNotification = new Notification
            {
                CustomerId = policyRequest.CustomerId,
                AgentId = null,
                Message = $"Your policy request For the AvailablePolicyId {policyRequest.AvailablePolicyId} with ID {policyRequest.RequestId} has been rejected.",
                CreatedAt = DateTime.UtcNow
            };

            var notificationResult = await _notificationRepo.AddAsync(customerNotification);

            if (!notificationResult.IsSuccess)
            {
                return OperationResult<bool>.Failure("Failed to add notification");
            }
        }

        return OperationResult<bool>.Success(true, "Policy request rejected and notification sent");

    }

    public async Task<OperationResult<bool>> RequestPolicyAsync(PolicyRequestDto requestDto)
    {
        // Step 1: Lookup the actual AvailablePolicy entity
        var policy = await _dbContext.AvailablePolicies
```

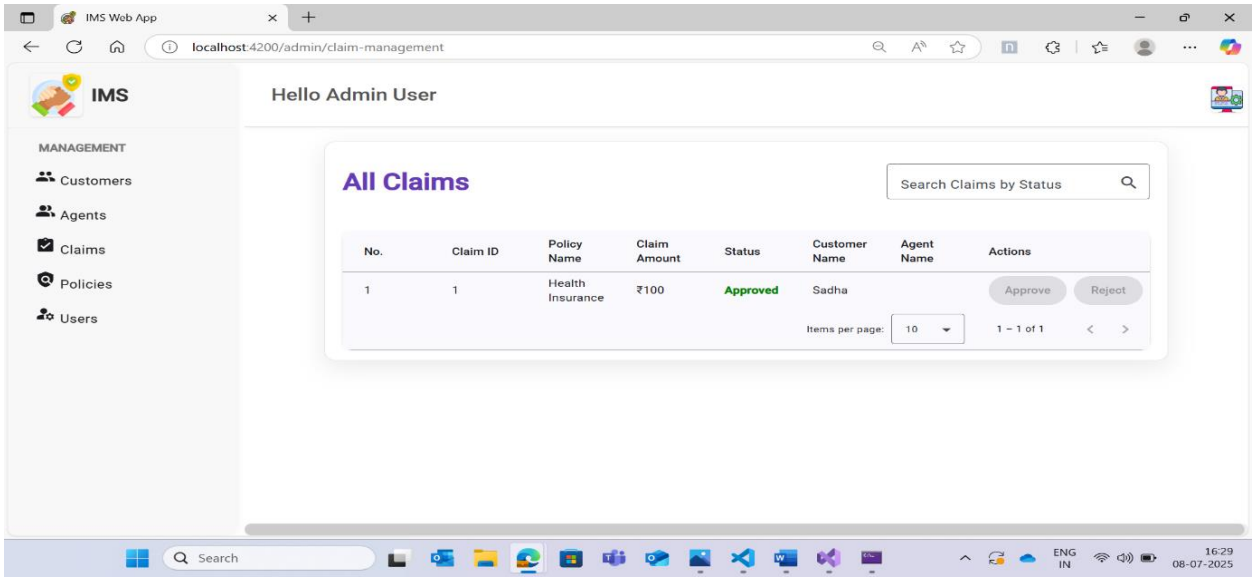
```
            .Include(p => p.IssuedPolicies)
            .Include(p => p.PolicyRequests)
            .FirstOrDefaultAsync(p => p.Name == requestDto.AvailablePolicyName);

        if (policy == null)
        {
            return OperationResult<bool>.Failure("No policy found with the specified
name.");
        }

        // Step 2: Lookup the actual Customer entity
        var customer = await _dbContext.Customers
            .FirstOrDefaultAsync(c => c.CustomerId == requestDto.CustomerId);

        if (customer == null)
        {
            return OperationResult<bool>.Failure("No customer found with the specified
ID.");
        }

        // Step 3: Check for existing requests by CustomerId + AvailablePolicyId
        var exists = await _dbContext.PolicyRequests.AnyAsync(r =>
            r.CustomerId == requestDto.CustomerId &&
            r.AvailablePolicyId == policy.AvailablePolicyId);

        if (exists)
        {
            return OperationResult<bool>.Failure("Policy request already exists for this
customer.");
```

```
        }

        // Step 4: Create and save the new request
        var newRequest = new PolicyRequest
        {
            CustomerId = requestDto.CustomerId,
            Customer = customer,
            AvailablePolicyId = policy.AvailablePolicyId,
            AvailablePolicy = policy,
            RequestedOn = DateTime.UtcNow,
            Status = "Pending"
        };

        await _policyRequestRepo.AddAsync(newRequest);

        return OperationResult<bool>.Success(true, "Request submitted successfully.");
    }



    //public async Task<OperationResult<bool>>
RequestPolicyAsync(PolicyRequestDto requestDto)
    //{
    //    //var customerId = int.Parse(User.FirstMethodOrDefault("CustomerId").Value);

    //    //var result = await _policyRepo.requestPolicyAsync(requestDto, customerId);

    //    //if (result.IsSucess)
    //    //    return Ok(result);
```

```
    //    //return BadRequest(result);









    //    var request = _mapper.Map<PolicyRequest>(requestDto);
    //    request.RequestedOn = DateTime.UtcNow;
    //    request.Status = "Pending";

    //    //var customerExists = await
_customerRepo.ExistsAsync(requestDto.CustomerId);

    //    //if (!customerExists)
    //    //{
    //    //    return OperationResult<bool>.Failure("Customer does not exist.");
    //    //}

    //    await _policyRequestRepo.AddAsync(request);
    //    return OperationResult<bool>.Success(true, "Request Submitted
Successfully");
    //}

    public async Task<OperationResult<bool>>
UpdateAvailablePolicyAsync(AvailablePolicyRequestDto dto, int policyId)
    {
        var availablePolicy = await _availablePolicyRepo.GetByIdAsync(policyId);
        if (!availablePolicy.IsSuccess)
        {
```

```
            return OperationResult<bool>.Failure($"Failed to fetch the available policy of
Id{policyId}");
        }

        var policy = _mapper.Map<AvailablePolicy>(dto);

        await _availablePolicyRepo.UpdateAsync(policy, policyId);
        return OperationResult<bool>.Success(true, "Available Policy updated
successfully");

    }

    public async Task<Policy?> GetExistingPolicyAsync(int customerId, int policyId)
    {
        return await _dbContext.Policies.FirstOrDefaultAsync(p => p.CustomerId ==
customerId && p.PolicyId == policyId);
    }

    public async Task<Policy?> GetExistingPolicyByNameAsync(int customerId, string
policyName)
    {
        return await _dbContext.Policies
            .Include(p => p.AvailablePolicy)
            .FirstOrDefaultAsync(p =>
                p.CustomerId == customerId &&
                p.AvailablePolicy.Name == policyName);
    }
```

```
  }
}
```



Claim-Management.css:

```
.container {
  width: 100%;
  max-width: 1000px;
  padding: 5px 20px 15px 20px;
  margin: 0px auto;
  background-color: #ffffff;
  border-radius: 12px;
  box-shadow: 0 10px 30px rgba(0, 0, 0, 0.08);
  overflow: hidden;
  border: 1px solid #e5e5e5;
}


.container {
```

```
  width: 920px;
}



.all-claims {
  color: #673ab7;
}
.header {
  display: flex;
  height: 100px;
  padding: auto;
  justify-content: space-between;
  align-items: center;
  margin-bottom: 20px;
}

.claims-card {
  padding: auto;
  width: 100%;
  display: flex;
}

.text-success {
  font-weight: bold;
  color: green;
}

.text-warning {
```

```
  font-weight: bold;
  color: orange;
}

.text-danger {
  font-weight: bold;
  color: red;
}

.mat-column-customerId,
.mat-column-agentId,
.mat-column-claimId {
  flex: 0 0 110px; /* Adjust this value as needed */
  /* This sets flex-grow to 0 (don't grow), flex-shrink to 0 (don't shrink),
     and a base width of 100px. */
  white-space: nowrap; /* Prevents text from wrapping */
  overflow: hidden; /* Hides overflowed content */
  text-overflow: ellipsis; /* Adds ellipsis for overflowed text */
}

/* Optional: To ensure the header cells also respect the width */
.mat-header-cell.mat-column-customerId,
.mat-header-cell.mat-column-agentId {
  flex: 0 0 110px; /* Same as above */
}
.mat-column-actions {
  flex: 0 0 200px;
  min-width: 200px;
```

```
  white-space: nowrap;
  padding-right: 8px;
}

/* Style for the buttons within the actions column */
.mat-column-actions button {
  margin-right: 8px;
}

.mat-column-actions button:last-of-type {
  margin-right: 0;
}



/* Search field styles */
.search-field {
  width: 300px;
}

.search-field ::ng-deep .mat-mdc-form-field-subscript-wrapper {
  display: none !important;
}

.search-field ::ng-deep .mat-mdc-form-field-bottom-align::before {
  display: none !important;
}

.search-field ::ng-deep .mat-mdc-text-field-wrapper {
```

```
  height: 50px !important;
}

.search-field ::ng-deep .mat-mdc-form-field-infix {
  min-height: 20px !important;
  padding-top: 15px !important;
  padding-bottom: 15px !important;
}

.search-field ::ng-deep .mat-mdc-form-field-hint-wrapper,
.search-field ::ng-deep .mat-mdc-form-field-error-wrapper {
  display: none !important;
}

.search-field ::ng-deep .mdc-notched-outline__leading,
.search-field ::ng-deep .mdc-notched-outline__notch,
.search-field ::ng-deep .mdc-notched-outline__trailing {
  height: 50px !important;
}
```

Claim-Management.html:

```
<div class="container">
  <div class="header">
    <h1 class="all-claims">All Claims</h1>
    <mat-form-field appearance="outline" class="search-field"
subscriptSizing="dynamic">
      <mat-label>Search Claims by Status</mat-label>
      <input
        matInput
```

```
      placeholder="e.g., Pending, Approved"
      (input)="applyFilter($event)"
    />
    <mat-icon matSuffix>search</mat-icon>
  </mat-form-field>
 </div>

 <mat-card class="claims-card">
  <mat-table [dataSource]="dataSource" class="mat-elevation-z4 cell" matSort>

   <!-- Serial No. -->
   <ng-container matColumnDef="sno">
    <mat-header-cell *matHeaderCellDef>No.</mat-header-cell>
    <mat-cell *matCellDef="let policy; let i = index">
     {{ (paginator.pageIndex || 0) * (paginator.pageSize || 10) + i + 1 }}
    </mat-cell>
   </ng-container>

   <!-- Claim ID -->
   <ng-container matColumnDef="claimId">
    <mat-header-cell *matHeaderCellDef>Claim ID</mat-header-cell>
    <mat-cell *matCellDef="let claim"> {{ claim.claimId }} </mat-cell>
   </ng-container>

   <!-- Policy Name -->
   <ng-container matColumnDef="policyName">
    <mat-header-cell *matHeaderCellDef>Policy Name</mat-header-cell>
    <mat-cell *matCellDef="let claim"> {{ claim.policyName }} </mat-cell>
```

```
    </ng-container>

    <!-- Claim Amount -->
    <ng-container matColumnDef="claimAmount">
     <mat-header-cell *matHeaderCellDef>Claim Amount</mat-header-cell>
     <mat-cell *matCellDef="let claim"> ₹{{ claim.claimAmount }} </mat-cell>
    </ng-container>

    <!-- Status -->
    <ng-container matColumnDef="status">
     <mat-header-cell *matHeaderCellDef>Status</mat-header-cell>
     <mat-cell *matCellDef="let claim">
       <span [ngClass]="getStatusClass(claim.status)">
        {{ claim.status }}
       </span>
     </mat-cell>
    </ng-container>

    <!-- Customer Name -->
    <ng-container matColumnDef="customerName">
     <mat-header-cell *matHeaderCellDef>Customer Name</mat-header-cell>
     <mat-cell *matCellDef="let claim"> {{ claim.customerName }} </mat-cell>
    </ng-container>

    <!-- Agent Name -->
    <ng-container matColumnDef="agentName">
     <mat-header-cell *matHeaderCellDef>Agent Name</mat-header-cell>
     <mat-cell *matCellDef="let claim"> {{ claim.agentName }} </mat-cell>
```

```
    </ng-container>

    <!-- Actions -->
    <ng-container matColumnDef="actions">
     <mat-header-cell *matHeaderCellDef>Actions</mat-header-cell>
     <mat-cell *matCellDef="let claim">
      <button
       mat-raised-button
       color="primary"
       (click)="updateStatus(claim, 'Approved')"
       [disabled]="claim.status === 'Approved' || claim.status === 'Rejected'"
      >
       Approve
      </button>
      <button
       mat-raised-button
       color="warn"
       (click)="updateStatus(claim, 'Rejected')"
       [disabled]="claim.status === 'Approved' || claim.status === 'Rejected'"
      >
       Reject
      </button>
     </mat-cell>
    </ng-container>

    <!-- Header and Row Rendering -->
    <mat-header-row *matHeaderRowDef="displayedColumns"></mat-header-row>
    <mat-row *matRowDef="let row; columns: displayedColumns"></mat-row>
```

```
  </mat-table>

  <!-- Paginator (no more (page) binding needed) -->
  <mat-paginator
   [length]="totalClaims"
   [pageSize]="10"
   [pageSizeOptions]="[5, 10, 15]"
   aria-label="Select page of claims"
  >
  </mat-paginator>
 </mat-card>
</div>
```

Claim-Management.ts :

```
import { Component, OnInit, ViewChild, AfterViewInit } from '@angular/core';
import { CommonModule } from '@angular/common';
import { FormsModule } from '@angular/forms';
import { MatTableDataSource, MatTableModule } from '@angular/material/table';
import { MatCardModule } from '@angular/material/card';
import { MatButtonModule } from '@angular/material/button';
import { MatPaginator, MatPaginatorModule, PageEvent } from
'@angular/material/paginator';
import { MatFormFieldModule } from '@angular/material/form-field';
import { MatInputModule } from '@angular/material/input';
import { MatIconModule } from '@angular/material/icon';
import { tap, catchError } from 'rxjs/operators';
import { Observable, of } from 'rxjs';
import { AdminService } from '../admin.service';
import { MatSnackBar } from '@angular/material/snack-bar';
```

```
import { ClaimStatusResponseDtoForAdmin } from
'../dtos/claimStatusResponseDtoForAdmin.dto';
import { ApiResponse } from '../../../shared/api-response.interface';
import { RejectDialogComponent } from '../../../shared/reject-dialog/reject-
dialog.component';
import { MatDialog } from '@angular/material/dialog';
import { MatSort } from '@angular/material/sort';

@Component({
 selector: 'app-claim-management',
 standalone: true,
 imports: [
  CommonModule,
  FormsModule,
  MatTableModule,
  MatCardModule,
  MatButtonModule,
  MatPaginatorModule,
  MatFormFieldModule,
  MatInputModule,
  MatIconModule,
 ],
 templateUrl: './claim-management.component.html',
 styleUrls: ['./claim-management.component.css']
})
export class ClaimManagementComponent implements OnInit, AfterViewInit {
 totalClaims = 0;
 searchTerm = '';
 dataSource = new MatTableDataSource<ClaimStatusResponseDtoForAdmin>([]);
```

```
 displayedColumns: string[] = ['sno', 'claimId', 'policyName', 'claimAmount', 'status',
'customerName', 'agentName', 'actions'];


 @ViewChild(MatPaginator) paginator!: MatPaginator;
 @ViewChild(MatSort) sort!: MatSort;


 constructor(
  private adminService: AdminService,
  private snackBar: MatSnackBar,
  private dialog: MatDialog
 ) {}


 ngOnInit(): void {
  this.dataSource.filterPredicate = (data: ClaimStatusResponseDtoForAdmin, filter:
string): boolean => {
    return data.status.toLowerCase().includes(filter.trim().toLowerCase());
  };
 }


 ngAfterViewInit(): void {
  this.paginator.page
    .pipe(
     tap(() => this.fetchClaims(this.paginator.pageIndex + 1, this.paginator.pageSize))
    )
    .subscribe();


  this.fetchClaims(this.paginator.pageIndex + 1, this.paginator.pageSize);
 }
```

```
  fetchClaims(page: number = 1, size: number = 10): void {
    this.adminService.getAllClaims(page, size).pipe(
      tap((response: ApiResponse<{ items: ClaimStatusResponseDtoForAdmin[], totalCount: number }>) => {
        if (response?.isSuccess && response.data) {
          this.dataSource.data = response.data.items;
          this.totalClaims = response.data.totalCount;
        } else {
          this.dataSource.data = [];
          this.totalClaims = 0;
          this.snackBar.open(` Failed to fetch claims: ${response?.message || 'Unknown error'}` , 'Close', {
            duration: 3000,
            panelClass: ['snackbar-error']
          });
        }
      }),
      catchError(error => {
        console.error('Error fetching claims:', error);
        this.dataSource.data = [];
        this.totalClaims = 0;
        this.snackBar.open('An error occurred while fetching claims.', 'Close', {
          duration: 3000,
          panelClass: ['snackbar-error']
        });
        return of({} as ApiResponse<{ items: ClaimStatusResponseDtoForAdmin[], totalCount: number }>);
      })
    ).subscribe();
```

```
 }

 applyFilter(event: Event): void {
  const filterValue = (event.target as HTMLInputElement).value;
  this.searchTerm = filterValue;
  this.dataSource.filter = this.searchTerm.trim().toLowerCase();

  if (this.dataSource.paginator) {
   this.dataSource.paginator.firstPage();
  }
 }

 updateStatus(claim: ClaimStatusResponseDtoForAdmin, newStatus: string): void {
  if (newStatus === 'Approved') {
   this.handleStatusChange(() => this.adminService.approveClaim(claim.claimId),
claim, newStatus);
  } else if (newStatus === 'Rejected') {
   const dialogRef = this.dialog.open(RejectDialogComponent);
   dialogRef.afterClosed().subscribe(confirmed => {
    if (confirmed) {
     this.handleStatusChange(() => this.adminService.rejectClaim(claim.claimId),
claim, newStatus);
    } else {
     this.snackBar.open('Claim rejection cancelled.', 'Close', {
      duration: 3000,
      panelClass: ['snackbar-info']
     });
    }
   });
```

```
    } else {
      this.snackBar.open('Invalid status action.', 'Close', {
        duration: 3000,
        panelClass: ['snackbar-error']
      });
    }
  }

  private handleStatusChange(
    apiCallFn: () => Observable<ApiResponse<boolean>>,
    claim: ClaimStatusResponseDtoForAdmin,
    newStatus: string
  ): void {
    apiCallFn().pipe(
      tap(response => {
        if (response.isSuccess) {
          claim.status = newStatus;
          this.snackBar.open(` Claim ${claim.claimId} ${newStatus} successfully! ` , 'Close', {
            duration: 3000,
            panelClass: ['snackbar-success']
          });
          this.fetchClaims(this.paginator.pageIndex + 1, this.paginator.pageSize);
        } else {
          this.snackBar.open(` Failed to ${newStatus.toLowerCase()} claim ${claim.claimId}:
${response.message}` , 'Close', {
            duration: 5000,
            panelClass: ['snackbar-error']
          });
```

```
      }
    }),
    catchError(error => {
      console.error(`Error updating claim ${claim.claimId}:`, error);
      this.snackBar.open(`An error occurred while updating claim ${claim.claimId}.`,
'Close', {
        duration: 5000,
        panelClass: ['snackbar-error']
      });
      return of({ isSuccess: false, message: 'Error', data: false } as
ApiResponse<boolean>);
    })
  ).subscribe();
 }


 getStatusClass(status: string): { [key: string]: boolean } {
  return {
    'text-success': status === 'Approved',
    'text-warning': status === 'Pending',
    'text-danger': status === 'Rejected'
  };
 }
}
```

WEBAPI

Claim Controller :

```
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using IMSIntrim.Application.Interfaces;
```

```
using IMSIntrim.Applications.DTOs;
using IMSIntrim.Infrastructure.Persistance;
using IMSIntrim.Domain.Models;
using IMSIntrim.Applications.Interfaces;

namespace IMSIntrim.Controllers
{
  [Route("api/[controller]")]
  [ApiController]
  public class ClaimsController : ControllerBase
  {
    private readonly InsuranceDbContext _context;
    private readonly IClaimServices _claimService;
    private readonly ITokenService _tokenService;

    public ClaimsController(InsuranceDbContext context, IClaimServices
claimServices,ITokenService tokenService)
    {
      _context = context;
      _claimService = claimServices;
      _tokenService = tokenService;
    }

    [HttpGet("admin/claims")]
    [Authorize(Roles = Roles.Admin)]
    public async Task<IActionResult> GetAllClaims(int page = 1, int size = 10)
    {
      var result = await _claimService.GetAllClaimsAsync(page, size);
```

```
        if (!result.IsSuccess) return NotFound(result);
        return Ok(result);
    }

    [HttpPost("admin/{claimId}/approve")]
    [Authorize(Roles = Roles.Admin)]
    public async Task<IActionResult> ApproveClaim(int claimId)
    {
        var result = await _claimService.ApproveClaimAsync(claimId);
        if (!result.IsSuccess) return BadRequest(result);
        return Ok(result);
    }

    [HttpPost("admin/{claimId}/reject")]
    [Authorize(Roles = Roles.Admin)]
    public async Task<IActionResult> RejectClaim(int claimId)
    {
        var result = await _claimService.RejectClaimAsync(claimId);
        if (!result.IsSuccess) return BadRequest(result);
        return Ok(result);
    }

    [HttpGet("admin/{customerId}/claims")]
    [Authorize(Roles = Roles.Admin)]
    public async Task<IActionResult> GetClaimsByCustomerId(int customerId)
    {
        var claims = await _claimService.GetClaimsByCustomerIdAsync(customerId);
        if (claims == null) return NotFound();
```

```
        return Ok(claims);
    }

    [HttpGet("agent/filed-claims")]
    [Authorize(Roles = Roles.Agent)]
    public async Task<IActionResult> GetFiledClaims()
    {
        var claims = await _claimService.GetFiledClaimsAsync();
        if (!claims.IsSuccess) return NotFound(claims);
        return Ok(claims);
    }

    [HttpPost("agent/file")]
    [Authorize(Roles = Roles.Agent)]
    public async Task<IActionResult> FileClaimAsAgent([FromBody]
ClaimFilingRequestDtoForAgent dto)
    {
        var loggedInAgentId = _tokenService.GetAgentIdFromCurrentRequest();
        if (loggedInAgentId == null)
        {
            return Unauthorized("User ID not found in token.");
        }

        if (dto.AgentId != loggedInAgentId)
        {
            return BadRequest("Invalid Agent or unauthoritized Request");
        }
        var result = await _claimService.FileClaimAsync(dto);
```

```
        if (!result.IsSuccess) return BadRequest(result);
        return Ok(result);
    }

    [HttpPost("customer/file")]
    [Authorize(Roles = Roles.Customer)]
    public async Task<IActionResult> FileClaimAsCustomer([FromBody]
ClaimFilingRequestDtoForCustomer dto)
    {
        var loggedInCustomerId = _tokenService.GetCustomerIdFromCurrentRequest();
        if (loggedInCustomerId == null)
        {
            return Unauthorized("User ID not found in token.");
        }

        if (dto.CustomerId != loggedInCustomerId)
        {
            return BadRequest("Invalid claim or unauthoritized Request");
        }
        var result = await _claimService.FileClaimAsync(dto);
        if (!result.IsSuccess) return BadRequest(result);
        return Ok(result);
    }

    [HttpGet("customer/my-claims")]
    [Authorize(Roles = Roles.Customer)]
    public async Task<IActionResult> GetMyClaims()
    {
```

```
        var claims = await _claimService.GetMyClaimsAsync();
        if (!claims.IsSuccess) return NotFound(claims);
        return Ok(claims);
      }
  }
  public static class Roles
  {
    public const string Admin = "Admin";
    public const string Agent = "Agent";
    public const string Customer = "Customer";
  }
}
```

ClaimService.cs:

```
using AutoMapper;
using Humanizer;
using IMSIntrim.Application.Interfaces;
using IMSIntrim.Applications.DTOs;
using IMSIntrim.Applications.Interfaces;
using IMSIntrim.Domain.Interfaces;
using IMSIntrim.Domain.Models;
using IMSIntrim.Infrastructure.Persistance;
using IMSIntrim.Infrastructure.Persistance.Repositories;
using IMSIntrim.Shared.Common;
using Microsoft.AspNetCore.Identity;
using Microsoft.EntityFrameworkCore;

namespace IMSIntrim.Application.Services
{
```

```
public class ClaimServices : IClaimServices
{
    private readonly ITokenService _tokenService;

    private readonly ICustomerRepository _customerRepo;
    private readonly IAgentRepository _agentRepo;
    private readonly IAvailablePolicyRepository _availablePolicyRepo;
    private readonly IPolicyRequestRepository _policyRequestRepo;
    private readonly IClaimRepository _claimRepo;
    private readonly IPolicyRepository _policyRepo;
    private readonly IUserRepository _userRepo;
    private readonly IRoleRepository _roleRepo;
    private readonly IUserRoleRepository _userRoleRepo;
    private readonly IPasswordHasherService _passwordHasher;
    private readonly IAuthServices _authService;
    private readonly IMapper _mapper;
    //private readonly ILogger<AdminService> _logger;
    private readonly INotificationRepository _notificationRepo;
    private readonly InsuranceDbContext _context;

    public ClaimServices(
      InsuranceDbContext context,
       ICustomerRepository customerRepo,
IAgentRepository agentRepo,
IAvailablePolicyRepository availablePolicyRepo,
IPolicyRequestRepository policyRequestRepo,
IClaimRepository claimRepo,
IPolicyRepository policyRepo,
```

```
IAuthServices authService,
IUserRepository userRepository,
IPasswordHasherService passwordHasher,
IRoleRepository roleRepository,
IUserRoleRepository userRoleRepository,
IAgentRepository agentRepository,
INotificationRepository notificationRepository,
ITokenService tokenService,
    //ILogger<AdminService> logger,

    IMapper mapper
      )
    {
      _customerRepo = customerRepo;
      _agentRepo = agentRepo;
      _availablePolicyRepo = availablePolicyRepo;
      _policyRequestRepo = policyRequestRepo;
      _claimRepo = claimRepo;
      _policyRepo = policyRepo;
      _authService = authService;
      _userRepo = userRepository;
      _passwordHasher = passwordHasher;
      _roleRepo = roleRepository;
      _userRoleRepo = userRoleRepository;
      _agentRepo = agentRepository;
      _notificationRepo = notificationRepository;
      _mapper = mapper;
      _tokenService = tokenService;
```

```
        _context = context;
        //    _logger = logger;
    }

    public async Task<OperationResult<bool>> IsFiledByAgentAsync(int claimId)
    {
        var claim = await _claimRepo.GetByIdAsync(claimId);
        var agentClaimResult = claim.Data;
        if (agentClaimResult == null)
        {
            return OperationResult<bool>.Failure("No Agnet claim found");
        }
        var result = agentClaimResult.AgentId.HasValue;
        return OperationResult<bool>.Success(result);
    }

    public async Task<OperationResult<bool>> ApproveClaimAsync(int claimId)
    {
        if (claimId <= 0)
        {
            return OperationResult<bool>.Failure("Invalid claim ID.");
        }

        await _claimRepo.UpdateStatusAsync(claimId, "Approved");

        var claim = await _claimRepo.GetByIdAsync(claimId);
        if (claim == null || !claim.IsSuccess)
        {
```

```
            return OperationResult<bool>.Failure("Claim not found.");
        }

        var policyId = claim.Data?.PolicyId;
        var customerId = claim.Data?.CustomerId;

        var isFiledByAgentResult = await IsFiledByAgentAsync(claimId);
        if (!isFiledByAgentResult.IsSuccess)
        {
            return OperationResult<bool>.Failure("Failed to determine if claim was filed by
an agent.");
        }

        if (!isFiledByAgentResult.Data)
        {
            var notification = new Notification
            {
                CustomerId = customerId,
                AgentId = null,
                Message = $"Great news! Your claim for Policy ID {policyId} has been
successfully approved. Thank you for your patience.",
                CreatedAt = DateTime.UtcNow
            };
            await _notificationRepo.AddAsync(notification);
        }
        else
        {

            var agentNotification = new Notification
```

```
            {
                AgentId = claim.Data?.AgentId,
                CustomerId = null,
                Message = $"Good news! The claim associated with Policy ID {policyId} for Customer ID {customerId} has been approved. Thank you for your diligent work.",
                CreatedAt = DateTime.UtcNow
            };
            await _notificationRepo.AddAsync(agentNotification);



            var customerNotification = new Notification
            {
                CustomerId = customerId,
                AgentId = null,
                Message = $"Great news! Your claim for Policy ID {policyId} has been successfully approved. Thank you for your patience.",
                CreatedAt = DateTime.UtcNow
            };
            await _notificationRepo.AddAsync(customerNotification);
        }



        return OperationResult<bool>.Success(true, "Policy claim approved and notifications sent.");


    }
```

```
    public async Task<OperationResult<bool>>
FileClaimAsync(ClaimFilingRequestDtoForAgent claimDto)
    {
        var claim = _mapper.Map<Claim>(claimDto);
        claim.Status = "Pending";
        claim.FiledDate = DateTime.UtcNow;

        await _claimRepo.AddAsync(claim);
        return OperationResult<bool>.Success(true, "Claim filed Succesfully");
    }

    public async Task<OperationResult<bool>>
FileClaimAsync(ClaimFilingRequestDtoForCustomer claimDto)
    {
        var policy = await _context.Policies
            .Include(p => p.AvailablePolicy)
            .Include(p => p.Customer) // Ensure Customer is included
            .FirstOrDefaultAsync(p =>
                p.CustomerId == claimDto.CustomerId &&
                p.AvailablePolicy.Name == claimDto.PolicyName);

        if (policy == null)
        {
            return OperationResult<bool>.Failure("No matching policy found for the
customer.");
        }

        var claim = new Claim
        {
```

```
            PolicyId = policy.PolicyId,
            Policy = policy, // Set the required Policy property
            CustomerId = claimDto.CustomerId,
            Customer = policy.Customer, // Set the required Customer property
            ClaimAmount = claimDto.ClaimAmount,
            Status = "Pending",
            FiledDate = DateTime.UtcNow
        };

        await _claimRepo.AddAsync(claim);

        return OperationResult<bool>.Success(true, "Claim filed successfully.");
    }

    //public async Task<OperationResult<bool>>
FileClaimAsync(ClaimFilingRequestDtoForCustomer claimDto)
    //{

    //    var claim = _mapper.Map<Claim>(claimDto);
    //    claim.Status = "Pending";
    //    claim.FiledDate = DateTime.UtcNow;

    //    await _claimRepo.AddAsync(claim);
    //    return OperationResult<bool>.Success(true, "Claim filed Successfully ");
    //}

    //public async
Task<OperationResult<PagedResult<ClaimStatusResponseDtoForAdmin>>>
GetAllClaimsAsync(int page, int size)
```

```
    //{
    //   var allClaims = await _claimRepo.GetAllAsync();
    //   if (!allClaims.IsSuccess)
    //   {
    //      return
OperationResult<PagedResult<ClaimStatusResponseDtoForAdmin>>.Failure(" Failed
To fetch the Claims");
    //   }
    //   var allClaimsList = allClaims.Data?.ToList();
    //   if (allClaimsList == null)
    //   {
    //      return
OperationResult<PagedResult<ClaimStatusResponseDtoForAdmin>>.Failure(" No Data
Available in the Claims List");
    //   }


    //   var paged = allClaimsList.Skip((page - 1) * size).Take(size).ToList();
    //   if (paged != null && paged.Any())
    //   {
    //      var claimResult = new PagedResult<ClaimStatusResponseDtoForAdmin>
    //      {
    //         Items = paged.Select(c =>
_mapper.Map<ClaimStatusResponseDtoForAdmin>(c)),
    //         PageNumber = page,
    //         PageSize = size,
    //         TotalCount = allClaimsList.Count
    //      };
    //      return
OperationResult<PagedResult<ClaimStatusResponseDtoForAdmin>>.Success(claimR
esult);
```

```
    //   }
    //}

    public async
Task<OperationResult<IEnumerable<ClaimsFiledByCustomerResponseDtoForAdmin>
>> GetClaimsByCustomerIdAsync(int customerId)
    {
        var claims = await _claimRepo.GetByCustomerIdAsync(customerId);


        if (!claims.IsSuccess)
        {
            return
OperationResult<IEnumerable<ClaimsFiledByCustomerResponseDtoForAdmin>>.Fail
ure("Cannot retrive Claims");
        }
        var claimsDto = claims.Data;
        if (claimsDto == null)
        {
            return
OperationResult<IEnumerable<ClaimsFiledByCustomerResponseDtoForAdmin>>.Fail
ure("Claims Not Found ");
        }
        var myClaims = claimsDto.Select(c =>
_mapper.Map<ClaimsFiledByCustomerResponseDtoForAdmin>(c)
        );
        return
OperationResult<IEnumerable<ClaimsFiledByCustomerResponseDtoForAdmin>>.Suc
cess(myClaims);
```

```
    }

    public async
Task<OperationResult<IEnumerable<ClaimStatusResponseDtoForAgent>>>
GetFiledClaimsAsync()
    {
        int? agentId = _tokenService.GetAgentIdFromCurrentRequest();

        if (agentId == null)
        {
            return
OperationResult<IEnumerable<ClaimStatusResponseDtoForAgent>>.Failure("Unable
to Get the AgentId from the Jwt Token");
        }

        var result = await _claimRepo.GetByAgentIdAsync((int)agentId);
        if (result.Data?.Any() != true)
        {
            return
OperationResult<IEnumerable<ClaimStatusResponseDtoForAgent>>.Failure("No
claims found for the specified agent.");
        }
        var claims = result.Data;
        if (claims?.Any() != true)
        {
            return
OperationResult<IEnumerable<ClaimStatusResponseDtoForAgent>>.Failure("No
claims found.");
        }
```

```
        var filedClaims = claims.Select(c =>
_mapper.Map<ClaimStatusResponseDtoForAgent>(c));


        return
OperationResult<IEnumerable<ClaimStatusResponseDtoForAgent>>.Success(filedClai
ms);


    }


    public async
Task<OperationResult<IEnumerable<ClaimStatusResponseDtoForCustomer>>>
GetMyClaimsAsync()
    {
        int? customerId = _tokenService.GetCustomerIdFromCurrentRequest();
        if (customerId == null)
        {
            return
OperationResult<IEnumerable<ClaimStatusResponseDtoForCustomer>>.Failure("Una
ble to Get the CustomerId from the Jwt Token");
        }
        var claims = await _claimRepo.GetByCustomerIdAsync((int)customerId);


        if (!claims.IsSuccess)
        {
            return
OperationResult<IEnumerable<ClaimStatusResponseDtoForCustomer>>.Failure("Faile
d to retrive the Claims ");
        }
        var claimsDto = claims.Data;
        if (claimsDto == null)
        {
```

```
            return
OperationResult<IEnumerable<ClaimStatusResponseDtoForCustomer>>.Failure("Clai
ms Not Found  ");
        }
        var myClaims = claimsDto.Select(c =>
_mapper.Map<ClaimStatusResponseDtoForCustomer>(c));
        return
OperationResult<IEnumerable<ClaimStatusResponseDtoForCustomer>>.Success(my
Claims);
    }

    public async Task<OperationResult<bool>> RejectClaimAsync(int claimId)
    {
        if (claimId <= 0)
        {
            return OperationResult<bool>.Failure("Invalid claim ID.");
        }

        var result = await _claimRepo.UpdateStatusAsync(claimId, "Rejected");
        if (result == null)
        {
            return OperationResult<bool>.Failure("Policy not found.");
        }

        var claim = await _claimRepo.GetByIdAsync(claimId);
        if (claim == null || !claim.IsSuccess)
        {
            return OperationResult<bool>.Failure("Claim not found.");
        }
```

```
        var policyId = claim.Data?.PolicyId;
        var customerId = claim.Data?.CustomerId;

        var isFiledByAgentResult = await IsFiledByAgentAsync(claimId);
        if (!isFiledByAgentResult.IsSuccess)
        {
            return OperationResult<bool>.Failure("Failed to determine if claim was filed by
an agent.");
        }

        if (isFiledByAgentResult.Data)
        {
            var agentNotification = new Notification
            {
                AgentId = claim.Data?.AgentId,
                CustomerId = null,
                Message = $"Unfortunately, the claim associated with Policy ID {policyId} for
Customer ID {customerId} has been rejected. Please review the details and take
necessary actions.",
                CreatedAt = DateTime.UtcNow
            };
            await _notificationRepo.AddAsync(agentNotification);

            var customerNotification = new Notification
            {
                CustomerId = customerId,
                AgentId = null,
```

```
            Message = $"We regret to inform you that your claim for Policy ID {policyId} has
been rejected. Please contact us for further assistance.",
            CreatedAt = DateTime.UtcNow
          };
          await _notificationRepo.AddAsync(customerNotification);
        }
        else
        {
          var notification = new Notification
          {
            CustomerId = customerId,
            AgentId = null,
            Message = $"We regret to inform you that your claim for Policy ID {policyId} has
been rejected. Please contact us for further assistance.",
            CreatedAt = DateTime.UtcNow
          };
          await _notificationRepo.AddAsync(notification);
        }

        return OperationResult<bool>.Success(true, "Policy claim rejected and
notifications sent.");

      }
      public async
Task<OperationResult<PagedResult<ClaimStatusResponseDtoForAdmin>>>
GetAllClaimsAsync(int page, int size)
      {
        var allClaims = await _claimRepo.GetAllAsync();
        if (!allClaims.IsSuccess)
```

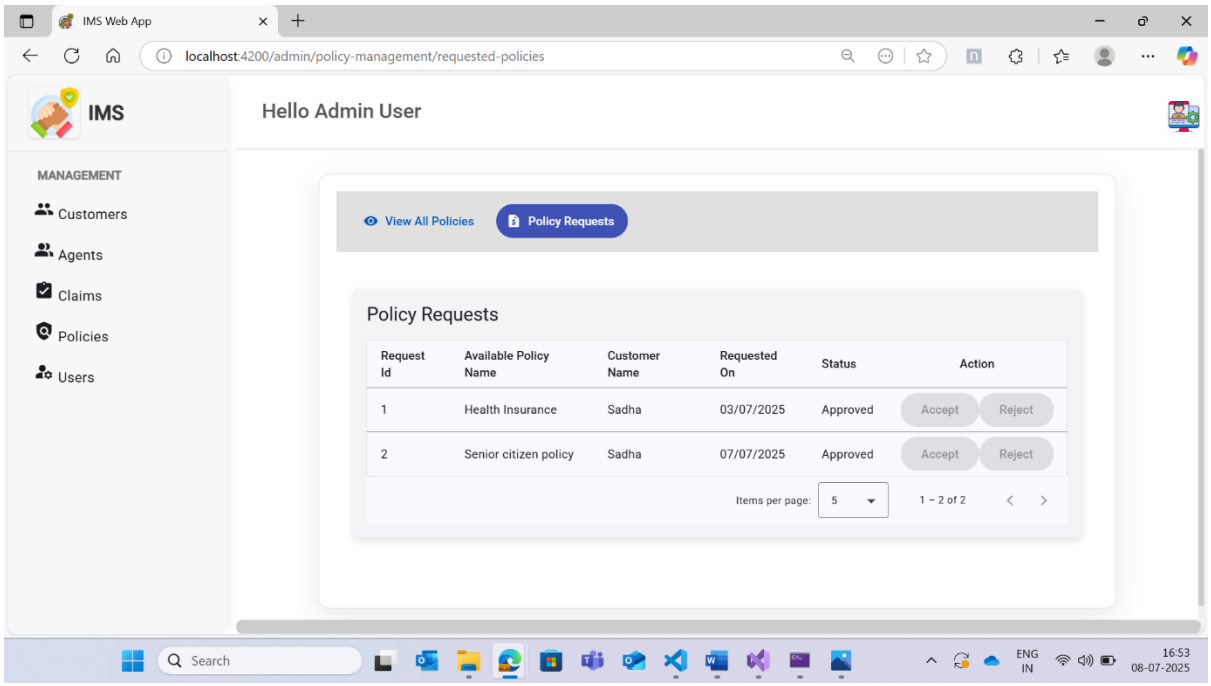
```
        {
            return
OperationResult<PagedResult<ClaimStatusResponseDtoForAdmin>>.Failure(" Failed
To fetch the Claims");
        }
        var allClaimsList = allClaims.Data?.ToList();
        if (allClaimsList == null)
        {
            return
OperationResult<PagedResult<ClaimStatusResponseDtoForAdmin>>.Failure(" No Data
Available in the Claims List");
        }

        var paged = allClaimsList.Skip((page - 1) * size).Take(size).ToList();
        if (paged != null && paged.Any())
        {
            var claimResult = new PagedResult<ClaimStatusResponseDtoForAdmin>
            {
                Items = paged.Select(c =>
_mapper.Map<ClaimStatusResponseDtoForAdmin>(c)),
                PageNumber = page,
                PageSize = size,
                TotalCount = allClaimsList.Count
            };
            return
OperationResult<PagedResult<ClaimStatusResponseDtoForAdmin>>.Success(claimR
esult);

        }
```

```
        return
OperationResult<PagedResult<ClaimStatusResponseDtoForAdmin>>.Failure(" Claims
not Found in the specific page");




    }


    //public
Task<OperationResult<PagedResult<ClaimStatusResponseDtoForAdmin>>>
GetAllClaimsAsync(int page, int size)
    //{
    //    throw new NotImplementedException();
    //}
  }
}
```

IClaimRepository.cs:

```
using IMSIntrim.Domain.Models;
using IMSIntrim.Shared.Common;
namespace IMSIntrim.Domain.Interfaces
{
  public interface IClaimRepository
  {
    public Task<OperationResult<IEnumerable<Claim>>> GetAllAsync();
    public Task<OperationResult<Claim?>> GetByIdAsync(int id);
    public Task<OperationResult<IEnumerable<Claim>>> GetByCustomerIdAsync(int
customerId);
    public Task<OperationResult<IEnumerable<Claim>>> GetByAgentIdAsync(int
agentId);
    public Task<OperationResult<bool>> AddAsync(Claim claim);
```

```
    public Task<OperationResult<bool>>UpdateStatusAsync(int claimId, string status);
}

}
```



[Policy-Management] Requested-Policies.css:

```
.policy-card {
  margin: 24px; /* Consistent margin around the card */
  max-width: 900px; /* Max width for readability on larger screens */
  margin-left: auto;
  margin-right: auto; /* Center the card */
  border-radius: 8px; /* Slightly rounded corners */
  box-shadow: 0 4px 8px rgba(0,0,0,0.1); /* Subtle shadow for depth */
}


.mat-card-header {
  padding-bottom: 16px; /* Space below header */
}
```

```
.mat-card-title {
  font-size: 24px; /* Larger title font */
  font-weight: 500;
  color: #3f51b5; /* Primary color */
}

.table-container {
  overflow-x: auto; /* Enables horizontal scrolling for the table on small screens */
  margin-top: 16px;
  border-radius: 4px; /* Slightly rounded corners for the table */
  border: 1px solid #e0e0e0; /* Light border for definition */
}

table {
  width: 100%;
  border-collapse: collapse; /* Ensure cells collapse borders */
}

th.mat-header-cell {
  background-color: #f5f5f5; /* Light background for table headers */
  color: rgba(0, 0, 0, 0.87);
  font-weight: 600; /* Bolder header text */
  font-size: 15px;
}

td.mat-cell, th.mat-header-cell {
  padding: 12px 16px; /* Padding for table cells */
```

```
  border-bottom: 1px solid #e0e0e0; /* Separator lines */
}

tr.mat-header-row {
  height: 56px; /* Standard header row height */
}

tr.mat-row {
  height: 48px; /* Standard data row height */
}

tr.mat-row:last-child td {
  border-bottom: none; /* No border for the last row */
}

.mat-column-action {
  width: 200px; /* Adjust width for buttons column */
  text-align: center;
  white-space: nowrap; /* Prevent buttons from wrapping */
}

.reject-button {
  margin-left: 8px; /* Space between Accept and Reject buttons */
}

.no-policies-message {
  text-align: center;
  padding: 30px;
```

```
 color: #757575;
 font-style: italic;
 font-size: 16px;
 background-color: #f9f9f9;
 border-radius: 4px;
 margin-top: 20px;
 border: 1px dashed #cccccc;
}

.color{
 background-color: red;
}
```

[Policy-Management] Requested-Policies.html:

```
<mat-card class="policy-card">
 <mat-card-header>
  <mat-card-title>Policy Requests</mat-card-title>
 </mat-card-header>

 <mat-card-content>
  <div class="table-container mat-elevation-z2">
   <table mat-table [dataSource]="dataSource" class="mat-elevation-z2">

    <ng-container matColumnDef="requestId">
     <th mat-header-cell *matHeaderCellDef>Request Id</th>
     <td mat-cell *matCellDef="let policy">{{ policy.requestId }}</td>
    </ng-container>

    <ng-container matColumnDef="availablePolicyName">
```

```
      <th mat-header-cell *matHeaderCellDef>Available Policy Name</th>
      <td mat-cell *matCellDef="let policy">{{ policy.availablePolicyName }}</td>
    </ng-container>

    <ng-container matColumnDef="customerName">
      <th mat-header-cell *matHeaderCellDef>Customer Name</th>
      <td mat-cell *matCellDef="let policy">{{ policy.customerName }}</td>
    </ng-container>

    <ng-container matColumnDef="requestedOn">
      <th mat-header-cell *matHeaderCellDef>Requested On</th>
      <td mat-cell *matCellDef="let policy">{{ policy.requestedOn | date: 'dd/MM/yyyy' }}</td>
    </ng-container>

    <ng-container matColumnDef="status">
      <th mat-header-cell *matHeaderCellDef>Status</th>
      <td mat-cell *matCellDef="let policy">{{ policy.status }}</td>
    </ng-container>

    <ng-container matColumnDef="action">
      <th mat-header-cell *matHeaderCellDef>Action</th>
      <td mat-cell *matCellDef="let policy">
        <button mat-raised-button color="primary"
                (click)="onAccept(policy)"
                [disabled]="policy.status === 'Approved' || policy.status === 'Rejected'">
          Accept
        </button>
```

```
        <button mat-raised-button color="warn"
              (click)="onReject(policy)"
              [disabled]="policy.status === 'Approved' || policy.status === 'Rejected'">
          Reject
        </button>
       </td>
      </ng-container>

      <tr mat-header-row *matHeaderRowDef="displayedColumns"></tr>
      <tr mat-row *matRowDef="let row; columns: displayedColumns;"></tr>
     </table>
    </div>

    <mat-paginator [length]="totalRequests"
                [pageSize]="pageSize"
                [pageSizeOptions]="pageSizeOptions"
                (page)="onPageChange($event)">
    </mat-paginator>

    <div *ngIf="dataSource.length === 0" class="no-policies-message">
     No policy requests to display.
    </div>
  </mat-card-content>
</mat-card>
```

[Policy-Management] Requested-Policies.ts:

```
import { AfterViewInit, Component, OnInit, ViewChild } from '@angular/core';
import { CommonModule } from '@angular/common';
import { MatTableDataSource, MatTableModule } from '@angular/material/table';
```

```
import { MatButtonModule } from '@angular/material/button';
import { MatCardModule } from '@angular/material/card';
import { MatDialog, MatDialogModule } from '@angular/material/dialog';
import { MatFormFieldModule } from '@angular/material/form-field';
import { MatInputModule } from '@angular/material/input';
import { MatPaginator, MatPaginatorModule, PageEvent } from
'@angular/material/paginator';

import { AgentAssignmentDialogComponent } from '../agent-assignment-dialog/agent-
assignment-dialog.component';
import { PolicyRequestStatusResponseDto } from
'../../../customer/dtos/policyRequestStatusResponse.dto';
import { MatSort } from '@angular/material/sort';
import { AdminService } from '../../admin.service';
import { PagedResult } from '../../../../shared/pagedResult.dto';
import { ApiResponse } from '../../../../shared/api-response.interface';
import { AssignAgentRequestDto } from '../../dtos/assignAgentRequest.dto';
import { MatSnackBar } from '@angular/material/snack-bar';
import { Router } from '@angular/router';

interface RequestedPolicy {
  policyId: string;
  policyNumber: string;
  policyName: string;
}

@Component({
  selector: 'app-requested-policies',
  standalone: true,
```

```
 imports: [
  CommonModule,
  MatTableModule,
  MatButtonModule,
  MatCardModule,
  MatDialogModule,
  MatFormFieldModule,
  MatInputModule,
  MatPaginatorModule
 ],
 templateUrl: './requested-policies.component.html',
 styleUrl: './requested-policies.component.css'
})

export class RequestedPoliciesComponent implements OnInit {
 dataSource: PolicyRequestStatusResponseDto[] = [];

 currentPage = 1;
 agentToAssign!:PolicyRequestStatusResponseDto;
 searchText:string='';
 totalRequests:number=0;
 policyRequests: PolicyRequestStatusResponseDto[] = [];
// dataSource = new MatTableDataSource<PolicyRequestStatusResponseDto>([]);
 displayedColumns: string[] = ['requestId',
'availablePolicyName','customerName','requestedOn','status', 'action'];

 @ViewChild(MatPaginator) paginator!: MatPaginator;
 @ViewChild(MatSort) sort!: MatSort;
```

```
 pageSize = 5;
 pageSizeOptions: number[] = [5, 10, 20];

 constructor(public dialog: MatDialog,private adminService:AdminService, private
snackBar: MatSnackBar,
  private router: Router
 ) { }

 ngOnInit(): void {
  this.getPolicies();
 }

 // ngAfterViewInit() {
 //   this.dataSource.paginator = this.paginator;
 // }

 getPolicies(page: number = this.currentPage, size: number = this.pageSize): void {
  this.currentPage = page;
  //this.adminService.getPolicyRequests(page, size).subscribe({ ... });
 this.adminService.getPolicyRequests(page,size).subscribe({
  next:(
   response:ApiResponse<PagedResult<PolicyRequestStatusResponseDto>>)=>{
    if(response.isSuccess&& response.data){
     this.policyRequests=response.data.items;
     this.dataSource =[...response.data.items];

     //this.dataSource.data=this.policyRequests;
```

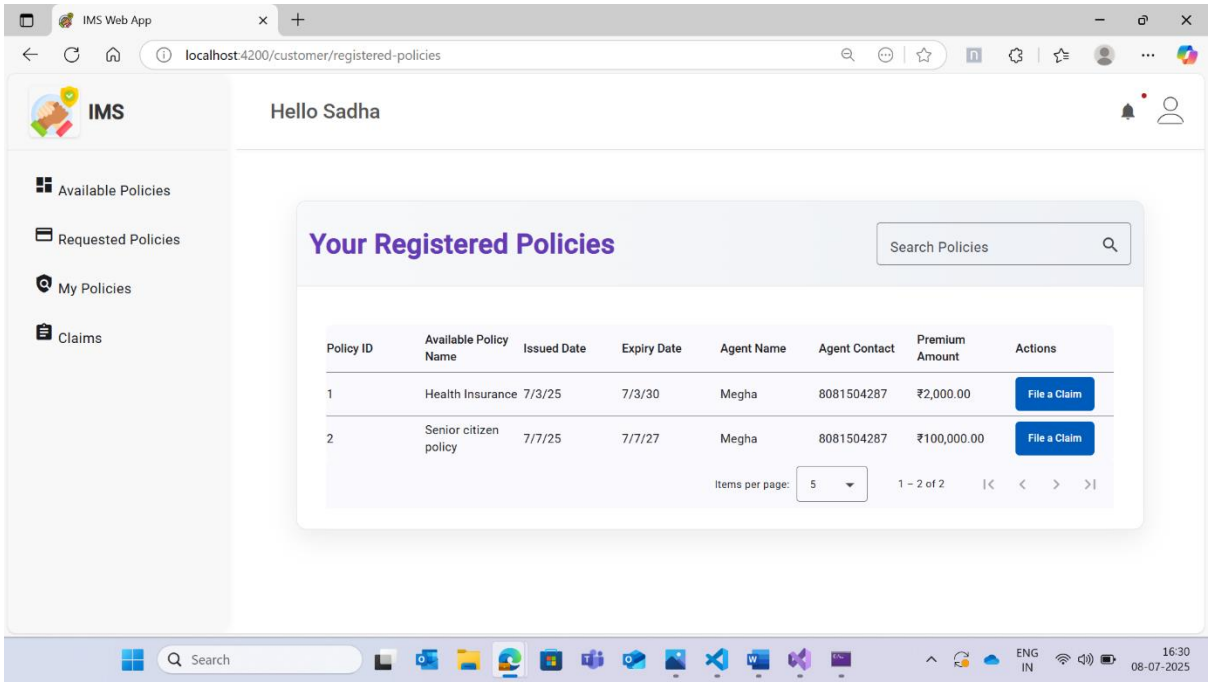
```
        this.totalRequests=response.data.totalCount;
      }
      else{
        console.error('Failed to fetch the Policy requests',response.message);
      }
    },
    error:(error:any)=>{
      console.error('Error While fetching the Policy requests',error);
    },

  });
 }

// onPageChange(event: PageEvent): void {
//   this.pageSize = event.pageSize;
// }
// onPageChange(event: PageEvent): void {
//   this.pageSize = event.pageSize;
//   const currentPage = event.pageIndex + 1;
//   console.log('Fetching page:', currentPage, 'with size:', this.pageSize);
//   this.getPolicies(currentPage, this.pageSize);
// }
 onPageChange(event: PageEvent) {
  this.getPolicies(event.pageIndex + 1, event.pageSize);
 }
```

```
 onAccept(policy: PolicyRequestStatusResponseDto):void
 {
    const dialogRef = this.dialog.open(AgentAssignmentDialogComponent,{
     width: '450px',
     disableClose: true,
     data: {policyId: policy.requestId}
    });

    dialogRef.afterClosed().subscribe((result) => {
     if(result && result.agentId)
     {
       const assignAgent : AssignAgentRequestDto = {
        agentId: Number(result.agentId)
       }

       this.adminService
       .approvePolicyRequest(assignAgent, policy.requestId)
       .subscribe({
         next: (response: ApiResponse<boolean>) => {
           if(response.isSuccess && response.data)
           {
             this.snackBar.open(
               'Policy request approved and agent assigned!',
               'Close',
               {duration: 3000}
               );
               policy.status = 'Approved';
               this.getPolicies();
```

```
          }
          else
          {
            alert('Failed to approve policy request: '+ response.message);
          }
         },
         error: (error: any) => {
          alert('Error approving policy request: ' + error);
         }
        }
        )
     };
   });
}

onReject(policy: PolicyRequestStatusResponseDto): void {
  this.adminService.rejectPolicyRequest(policy.requestId).subscribe({
    next:(response:ApiResponse<boolean>)=>{
      if(response.isSuccess&& response.data)
      {
        this.snackBar.open(
          'Policy Request rejected Successfully','Close',{duration:3000}
        );
        policy.status = 'Rejected';
        this.getPolicies();
        console.log('Rejected policy:', policy.requestId);

      }
```

```
      else{
        alert('Failed to reject the policy request'+response.message);
      }
    },
    error:(error:any)=>{
      alert('Error while reject the policy request'+error)
    }
  });
 }

 openAgentAssignmentDialog(policyId: string): void {
  const dialogRef = this.dialog.open(AgentAssignmentDialogComponent, {
    width: '450px',
    disableClose: true,
    data: { policyId: policyId }
  });

  dialogRef.afterClosed().subscribe(result => {
    console.log('The agent assignment dialog was closed. Result:', result);
    if (result) {
    }
  });
 }
}
```

The below are the screens where the customer can handle his policies and manage the claims.



Customer-registered-Policies:

```
.my-policies-container {
  width: 100%;
  max-width: 1000px;
  margin: 40px auto;
  background-color: #ffffff;
  border-radius: 12px;
  box-shadow: 0 10px 30px rgba(0, 0, 0, 0.08);
  overflow: hidden;
  border: 1px solid #e5e5e5;
}

.mat-mdc-header-cell {
  padding: 0;
}
.mat-mdc-cell {
  padding: 0;
```

```
}
/* Header Section */
.policies-header {
  display: flex;
  justify-content: space-between;
  align-items: center;
  padding: 25px 15px;
  background: linear-gradient(to right, #f8f9fa, #eef1f6);
  border-bottom: 1px solid #e0e0e0;
  margin-bottom: 20px;
}

.policies-title {
  font-size: 2em;
  font-weight: 700;
  color: #673ab7;
  margin: 0;
  letter-spacing: 0.5px;
}
/* Search Field */
.search-field {
  width: 300px;
}

.search-field ::ng-deep .mat-mdc-form-field-subscript-wrapper {
  display: none !important;
}
```

```
.search-field ::ng-deep .mat-mdc-form-field-bottom-align::before {
  display: none !important;
}

.search-field ::ng-deep .mat-mdc-text-field-wrapper {
  height: 50px !important;
}

.search-field ::ng-deep .mat-mdc-form-field-infix {
  min-height: 20px !important;
  padding-top: 15px !important;
  padding-bottom: 15px !important;
}

.search-field ::ng-deep .mat-mdc-form-field-hint-wrapper,
.search-field ::ng-deep .mat-mdc-form-field-error-wrapper {
  display: none !important;
}

.search-field ::ng-deep .mdc-notched-outline__leading,
.search-field ::ng-deep .mdc-notched-outline__notch,
.search-field ::ng-deep .mdc-notched-outline__trailing {
  height: 50px !important;
}

.policy-search-field {
  width: 280px;
}
```

```
.mat-mdc-form-field-appearance-outline .mat-mdc-form-field-flex {
  padding: 0.5em 1em !important;
}

.mat-mdc-form-field-appearance-outline .mat-mdc-form-field-infix {
  padding-top: 0.8em !important;
  padding-bottom: 0.8em !important;
}

/* Table Wrapper */
.policies-table-wrapper {
  padding: 25px 35px;
  overflow-x: auto;
}

table {
  width: 100%;
  border-collapse: separate;
  border-spacing: 0;
  border-radius: 8px;
  overflow: hidden;
  table-layout: fixed;
}

/* Table Headers */
th.mat-header-cell {
  background-color: #42a5f5;
```

```
  color: #ffffff;
  font-weight: 600;
  font-size: 1em;
  padding: 18px 15px;
  text-align: left;
  border-bottom: none;
  position: sticky;
  top: 0;
  z-index: 1;
  white-space: nowrap;
  overflow: hidden;
  text-overflow: ellipsis;
}

th.mat-header-cell:first-child {
  border-top-left-radius: 8px;
}
th.mat-header-cell:last-child {
  border-top-right-radius: 8px;
}

/* Table Data Cells */
td.mat-cell {
  padding: 15px;
  color: #444;
  font-size: 0.95em;
  border-bottom: 1px solid #f0f0f0;
  word-wrap: break-word;
```

```
  overflow: hidden;
  text-overflow: ellipsis;
}

/* Alternating row background */
tr.mat-row:nth-child(even) {
  background-color: #fcfcfc;
}
tr.mat-row:hover {
  background-color: #e3f2fd;
  transition: background-color 0.2s ease-in-out;
}

tr.mat-row:last-child td.mat-cell {
  border-bottom: none;
}

/* --- COLUMN WIDTH DEFINITIONS (Adjusted for removal of 'sno') --- */
/* You might want to slightly increase other widths to fill the space */

/* REMOVED: Styles for th.mat-header-cell.mat-column-sno, td.mat-cell.mat-column-sno */

th.mat-header-cell.mat-column-policyId,
td.mat-cell.mat-column-policyId {
  width: 10%; /* Adjusted from 90px, using percentage for more flexibility */
}
```

```
th.mat-header-cell.mat-column-availablePolicyId,
td.mat-cell.mat-column-availablePolicyId {
 width: 15%;
}

th.mat-header-cell.mat-column-issuedDate,
td.mat-cell.mat-column-issuedDate,
th.mat-header-cell.mat-column-expiryDate,
td.mat-cell.mat-column-expiryDate {
 width: 12%; /* Giving more space to dates */
}

th.mat-header-cell.mat-column-agentId,
td.mat-cell.mat-column-agentId {
 width: 10%;
}

th.mat-header-cell.mat-column-agentContact,
td.mat-cell.mat-column-agentContact {
 width: 15%;
}

th.mat-header-cell.mat-column-premiumAmount,
td.mat-cell.mat-column-premiumAmount {
 width: 13%; /* Adjusted for currency, leaving some flexibility */
 text-align: right;
}
```

```
th.mat-header-cell.mat-column-actions,
td.mat-cell.mat-column-actions {
  width: 13%; /* Ensure the button fits */
  text-align: center;
}

/* Styling for the File a Claim button */
.file-claim-button {
  background-color: #007bff;
  color: white;
  padding: 8px 15px;
  border-radius: 5px;
  font-size: 0.85em;
  font-weight: 500;
  transition: background-color 0.3s ease;
  white-space: nowrap;
  min-width: 90px;
}

.file-claim-button:hover:not([disabled]) {
  background-color: #0056b3;
}

/* No data row */
.mat-no-data-row {
  text-align: center;
  padding: 30px;
  color: #888;
```

```
  font-style: italic;
}

/* Responsive Adjustments */
@media (max-width: 1024px) {
  .my-policies-container {
    max-width: 95%;
    margin: 20px auto;
  }

  .policies-header {
    padding: 20px 25px;
  }

  .policies-title {
    font-size: 1.8em;
  }

  .policies-table-wrapper {
    padding: 15px 25px;
  }
}

@media (max-width: 768px) {
  .policies-header {
    flex-direction: column;
    align-items: flex-start;
    gap: 15px;
```

```
  padding: 20px;
 }

 .policies-title {
  font-size: 1.5em;
 }

 .policy-search-field {
  width: 100%;
 }

 .policies-table-wrapper {
  overflow-x: auto;
  padding: 15px 20px;
 }

 table {
  width: 100%;
 }

 th.mat-header-cell,
 td.mat-cell {
  padding: 12px 10px;
  font-size: 0.85em;
 }

 .file-claim-button {
  padding: 6px 12px;
```

```
  font-size: 0.8em;
 }
}

@media (max-width: 480px) {
 .policies-title {
  font-size: 1.3em;
 }

 .file-claim-button {
  padding: 5px 10px;
  font-size: 0.75em;
 }
}
```

Customer-Registered-Policies.html:

```
<div class="my-policies-container">
 <!-- Header section for policies including title and search field -->
 <div class="policies-header">
  <h1 class="policies-title">Your Registered Policies</h1>
  <!-- Material Form Field for policy search input -->
  <mat-form-field
   appearance="outline"
   class="search-field"
   subscriptSizing="dynamic"
  >
   <mat-label>Search Policies</mat-label>
   <!-- Input field for filtering policies, triggers applyFilter on keyup -->
   <input
```

```
      matInput
      (keyup)="applyFilter($event)"
      placeholder="e.g. Life Secure"
      #input
     />
    <!-- Search icon as a suffix for the input field -->
    <mat-icon matSuffix>search</mat-icon>
   </mat-form-field>
  </div>

  <!-- Wrapper for the policies table with Material elevation styling -->
  <div class="policies-table-wrapper mat-elevation-z2">
   <!-- Material Table definition -->
   <table mat-table [dataSource]="dataSource" matSort>
    <!-- Policy ID Column Definition -->
    <ng-container matColumnDef="policyId">
     <th mat-header-cell *matHeaderCellDef mat-sort-header>Policy ID</th>
     <td mat-cell *matCellDef="let policy">{{ policy.policyId }}</td>
    </ng-container>

    <!-- Available Policy ID Column Definition -->
    <ng-container matColumnDef="availablePolicyName">
     <th mat-header-cell *matHeaderCellDef mat-sort-header>
      Available Policy Name
     </th>
     <td mat-cell *matCellDef="let policy">
      {{ policy.availablePolicyName }}
     </td>
```

```
    </ng-container>

    <!-- Issued Date Column Definition with date pipe -->
    <ng-container matColumnDef="issuedDate">
     <th mat-header-cell *matHeaderCellDef mat-sort-header>Issued Date</th>
     <td mat-cell *matCellDef="let policy">
      {{ policy.issuedDate | date : "shortDate" }}
     </td>
    </ng-container>

    <!-- Expiry Date Column Definition with date pipe -->
    <ng-container matColumnDef="expiryDate">
     <th mat-header-cell *matHeaderCellDef mat-sort-header>Expiry Date</th>
     <td mat-cell *matCellDef="let policy">
      {{ policy.expiryDate | date : "shortDate" }}
     </td>
    </ng-container>

    <!-- Agent ID Column Definition -->
    <ng-container matColumnDef="agentName">
     <th mat-header-cell *matHeaderCellDef mat-sort-header>Agent Name</th>
     <td mat-cell *matCellDef="let policy">{{ policy.agentName }}</td>
    </ng-container>

    <!-- Agent Contact Column Definition -->
    <ng-container matColumnDef="agentContact">
     <th mat-header-cell *matHeaderCellDef mat-sort-header>Agent Contact</th>
     <td mat-cell *matCellDef="let policy">{{ policy.agentContact }}</td>
```

```
    </ng-container>

    Premium Amount Column Definition with number pipe for currency formatting
    <ng-container matColumnDef="premiumAmount">
     <th mat-header-cell *matHeaderCellDef mat-sort-header>
      Premium Amount
     </th>
     <td mat-cell *matCellDef="let policy">
      ₹{{ policy.premiumAmount | number : "1.2-2" }}
     </td>
    </ng-container>

    <!-- Actions Column Definition for the "File a Claim" button -->
    <ng-container matColumnDef="actions">
     <th mat-header-cell *matHeaderCellDef>Actions</th>
     <td mat-cell *matCellDef="let policy">
      <button
       mat-flat-button
       color="primary"
       (click)="fileAClaim(policy)"
       class="file-claim-button"
      >
       File a Claim
      </button>
     </td>
    </ng-container>

    <!-- Table header row definition -->
```

```
    <tr mat-header-row *matHeaderRowDef="displayedColumns"></tr>
    <!-- Table data row definition -->
    <tr mat-row *matRowDef="let row; columns: displayedColumns"></tr>

    <!-- Row displayed when no matching data is found -->
    <tr class="mat-row" *matNoDataRow>
     <td class="mat-cell" [attr.colspan]="displayedColumns.length">
       No policies matching "{{ input.value }}"
     </td>
    </tr>
   </table>
   <!-- Material Paginator for controlling table pagination -->
   <!-- [pageSizeOptions] defines available page sizes, [showFirstLastButtons] adds
navigation buttons -->
   <mat-paginator
    [pageSizeOptions]="[5, 10, 20]"
    showFirstLastButtons
    aria-label="Select page of policies"
   ></mat-paginator>
  </div>
</div>
```

Customer-Registered-Policies.ts:

```
import { AfterViewInit, Component, OnInit, ViewChild } from '@angular/core';
import { CommonModule } from '@angular/common';
import { MatTableDataSource, MatTableModule } from '@angular/material/table';
import { MatInputModule } from '@angular/material/input';
import { MatFormFieldModule } from '@angular/material/form-field';
import { MatIconModule } from '@angular/material/icon';
```

```
import { MatSort, MatSortModule } from '@angular/material/sort';
import { MatButtonModule } from '@angular/material/button';
import { Router, RouterModule } from '@angular/router';
import { MatDialog } from '@angular/material/dialog';
// Assuming these DTOs are defined in your project
import { CustomerRegisterResponseDto } from '../dtos/customerRegisterResponse.dto';
import { CustomerPoliciesResponseDto } from '../dtos/customerPoliciesResponse.dto';
import { MatPaginator, MatPaginatorModule } from '@angular/material/paginator'; // Import MatPaginatorModule
import { CustomerService } from '../customer.service';
import { ApiResponse } from '../../../shared/api-response.interface';

@Component({
 selector: 'app-customer-registered-policies',
 standalone: true,
 imports: [
  CommonModule,
  MatTableModule,
  MatInputModule,
  MatFormFieldModule,
  MatIconModule,
  MatSortModule,
  MatButtonModule,
  RouterModule,
  MatPaginatorModule, // Added MatPaginatorModule to imports
  // MatDialogModule // Uncomment and add MatDialogModule if you are using MatDialog
 ],
 templateUrl: './customer-registered-policies.component.html',
```

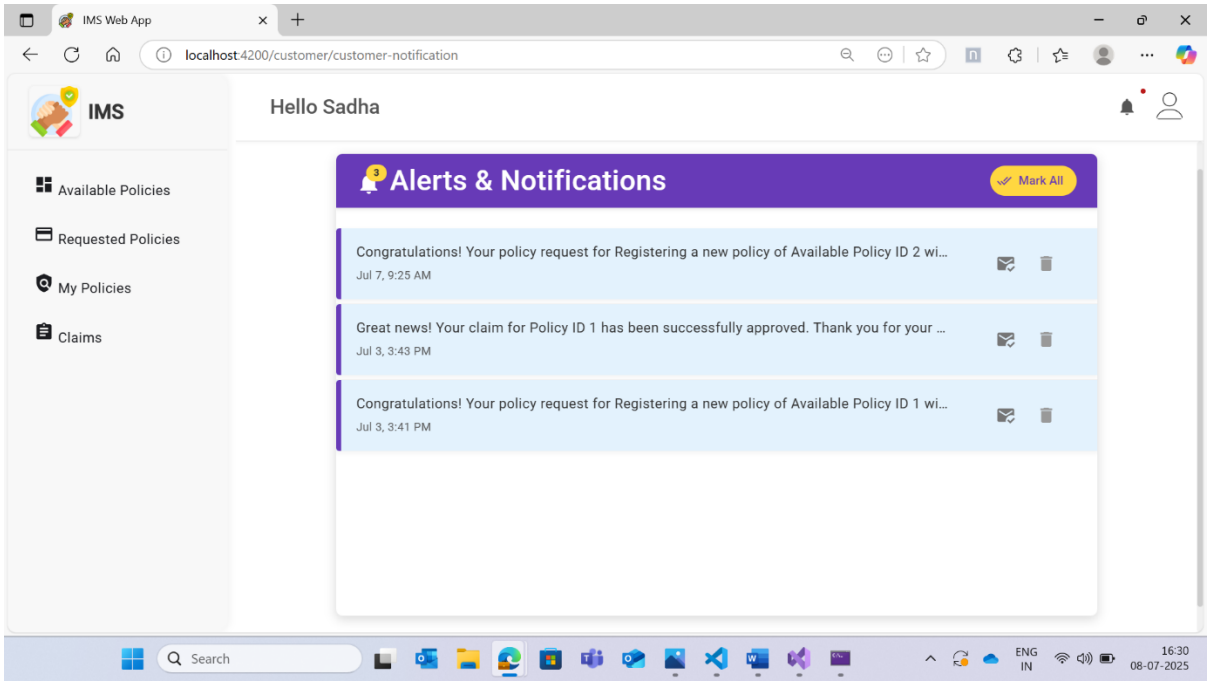
```
  styleUrls: ['./customer-registered-policies.component.css']
})
export class CustomerRegisteredPoliciesComponent implements OnInit, AfterViewInit {
  searchText: string = '';
  totalPolicies: number = 0; // Total number of policies for pagination display (if needed in template)
  registerdPolicies: CustomerPoliciesResponseDto[] = []; // Raw data from the API, assuming Policy is defined elsewhere

  // Defines the columns to be displayed in the table
  displayedColumns: string[] = [
    'policyId',
    'availablePolicyName',
    'issuedDate',
    'expiryDate',
    'agentName',
    'agentContact',
    'premiumAmount',
    'actions'
  ];
  // DataSource for the Material Table, initialized with an empty array
  dataSource = new MatTableDataSource<CustomerPoliciesResponseDto>([]);

  // ViewChild to get a reference to the MatPaginator component in the template
  @ViewChild(MatPaginator) paginator!: MatPaginator; // Used for pagination functionality
  // ViewChild to get a reference to the MatSort component in the template
  @ViewChild(MatSort) sort!: MatSort;
  loggedInCustomerId:number|null=null;
```

```
 // Constructor to inject services
 constructor(private router: Router, private customerService: CustomerService) { }

 // Lifecycle hook: Called after Angular has initialized all data-bound properties of a directive.
 ngOnInit() {
  this.customerService.getCustomerProfile$().subscribe((profile) => {
   if (profile) {

    // Ensure customerId is always a number
    this.loggedInCustomerId =
     typeof profile.customerId === 'string'
      ? Number(profile.customerId)
      : profile.customerId;
   } else {
    this.customerService.fetchAndStoreCustomerProfile().subscribe();
   }
  });
  this.getRegisteredPolicies(); // Fetch policies when the component initializes
 }

 // Lifecycle hook: Called after Angular has fully initialized a component's view.
 ngAfterViewInit(): void {
  // Assign the paginator and sort instances to the dataSource after
  //  the view has initialized
  this.dataSource.paginator = this.paginator;
  this.dataSource.sort = this.sort;
```

```
 }

 // Fetches registered policies from the customer service
 getRegisteredPolicies(): void {
  this.customerService.getCustomerPolicies().subscribe({
   next: (response: ApiResponse<CustomerPoliciesResponseDto[]>) => {
    if (response.isSuccess && response.data) {
     this.registerdPolicies = response.data || []; // Ensure data is an array
     this.dataSource.data = this.registerdPolicies; // Assign data to
MatTableDataSource
     this.totalPolicies = this.registerdPolicies.length; // Update total policies count
    } else {
     console.error('Failed to fetch registered policies:', response.message);
    }
   },
   error: (error: any) => {
    console.error('Error fetching registered policies:', error);
   }
  });
 }

 // Applies filter to the table data based on user input
 applyFilter(event: Event) {
  const filterValue = (event.target as HTMLInputElement).value;
  this.dataSource.filter = filterValue.trim().toLowerCase();

  // If a paginator is available, reset to the first page after filtering
  if (this.dataSource.paginator) {
```

```
    this.dataSource.paginator.firstPage();
  }
 }

 // Checks if claiming is enabled for a given policy (currently always true)
 isClaimEnabled(policy: { status: string }): boolean {
  return true;
 }

 // Handles the "File a Claim" action for a policy
 fileAClaim(policy: CustomerPoliciesResponseDto): void {
  console.log(`Attempting to file a claim for Policy ID: ${policy.policyId}`);
 // this.router.navigate(['/customer/file-claim'], {
 //   queryParams: {
 //     customerId: this.loggedInCustomerId,
 //     policyId: policy.policyId
 //   }
 this.router.navigate(['/customer/file-claim'], {
  queryParams: {
    customerId: this.loggedInCustomerId,
    policyName: policy.availablePolicyName // ✅ use this instead of policyId
  }
 }); // Navigates to the file claim route

 console.log('Claim Data:', {
  customerId: this.loggedInCustomerId,
  policyId: policy.policyId
 });
```

```
 }
}
```



Customer-Notification.css:

```
.notification-dashboard {
 max-width: 900px;
 margin: 32px auto;
 border-radius: 8px;
 overflow: hidden;
 background-color: #ffffff;
 box-shadow: 0 4px 12px rgba(0, 0, 0, 0.15);
 font-family: 'Roboto', sans-serif;
 display: flex;
 flex-direction: column;
}
```

```
.notification-toolbar {
  height: 64px;
  background-color: #673ab7;
  color: white;
  padding: 0 24px;
  box-shadow: 0 2px 4px rgba(0, 0, 0, 0.1);
  display: flex;
  align-items: center;
}

.notification-toolbar .header-icon {
  font-size: 30px;
  margin-right: 16px;
  position: relative;
}

.notification-toolbar .header-icon ::ng-deep .mat-badge-content {
  background-color: #ffd740 !important;
  color: #333 !important;
  font-weight: 600 !important;
  font-size: 11px !important;
  height: 20px !important;
  min-width: 20px !important;
  line-height: 20px !important;
  top: 5px !important;
  right: 5px !important;
  border-radius: 10px !important;
```

```
  padding: 0 4px;
}

.notification-toolbar .header-title {
  font-size: 1.5em;
  font-weight: 500;
  letter-spacing: 0.5px;
  margin: 0;
}

.notification-toolbar .toolbar-spacer {
  flex: 1 1 auto;
}

.notification-toolbar .mark-all-read-button {
  background-color: #ffd740;
  color: #673ab7;
  font-weight: 500;
  padding: 0 16px;
  height: 36px;
  border-radius: 20px;
  transition: background-color 0.2s ease-in-out;
}

.notification-toolbar .mark-all-read-button:hover:not([disabled]) {
  background-color: #ffe680;
}
```

```
.notification-toolbar .mark-all-read-button[disabled] {
  opacity: 0.7;
  cursor: not-allowed;
  background-color: rgba(0, 0, 0, 0.12);
  color: rgba(0, 0, 0, 0.38);
}

.notification-toolbar .mark-all-read-button mat-icon {
  margin-right: 8px;
}

.notification-list-container {
  padding: 16px 0;
  flex-grow: 1;
  min-height: 450px;
}

.notification-item {
  display: flex;
  align-items: flex-start;
  padding: 16px 24px;
  transition: background-color 0.2s ease-in-out, box-shadow 0.2s ease-in-out, height
0.3s ease-out;
  cursor: default;
  position: relative;
  overflow: hidden;
  margin-bottom: 5px;
  height: auto;
```

```
  min-height: 64px;
}

.notification-item:hover {
  background-color: #f5f5f5;
}

.notification-item.unread {
  background-color: #e3f2fd;
  font-weight: 500;
  box-shadow: 0 2px 5px rgba(0, 0, 0, 0.05);
}

.notification-item.unread::before {
  content: '';
  position: absolute;
  left: 0;
  top: 0;
  bottom: 0;
  width: 6px;
  background-color: #673ab7;
  border-top-left-radius: 4px;
  border-bottom-left-radius: 4px;
}

.notification-item.unread .notification-content .notification-message {
  color: #333;
}
```

```
.notification-item.unread .notification-content .notification-timestamp {
  color: #555;
}

.notification-item.read {
  background-color: #ffffff;
  color: #666;
}

.notification-item.read .notification-content .notification-message {
  color: #666;
  font-weight: 400;
}

.notification-item.read .notification-content .notification-timestamp {
  color: #999;
}

.notification-item.read .read-toggle-button {
  color: #999;
}

.notification-item .notiffication-type-indicator {
  min-width: 12px;
  height: 12px;
  border-radius: 50%;
  margin-right: 18px;
```

```
  flex-shrink: 0;
}

.notification-item .notification-type-indicator.type-success { background-color: #4caf50; }
.notification-item .notification-type-indicator.type-warning { background-color: #ff9800; }
.notification-item .notification-type-indicator.type-info { background-color: #2196f3; }
.notification-item .notification-type-indicator.type-error { background-color: #f44336; }
.notification-item .notification-type-indicator.type-new { background-color: #00bcd4; }

.notification-item .notification-content {
  flex-grow: 1;
  display: flex;
  flex-direction: column;
  margin-right: 16px;
  min-height: 0;
  overflow: visible;
  flex-shrink: 1;
}

.notification-item .notification-content .notification-message {
  font-size: 16px;
  line-height: 1.5;
  margin-bottom: 4px;
  white-space: nowrap;
  overflow: hidden;
  text-overflow: ellipsis;
  max-width: 100%;
```

```
  display: block;
  max-height: 1.5em;
  -webkit-user-select: none;
  -moz-user-select: none;
  -ms-user-select: none;
  user-select: none;
  cursor: pointer;
  transition: max-height 0.3s ease-out, white-space 0.3s ease-out, text-overflow 0.3s ease-out;
}

.notification-item:has(.expand-toggle-checkbox:checked) .notification-content .notification-message {
  white-space: normal;
  overflow: visible;
  text-overflow: clip;
  max-height: 500px;
  cursor: default;
}

.expand-toggle-checkbox {
  display: none;
}

.notification-item .expand-toggle-label {
  position: absolute;
  top: 0;
  left: 0;
  right: 0;
```

```
  bottom: 0;
  z-index: 1;
  cursor: pointer;
}

.notification-item .notification-actions {
  display: flex;
  flex-shrink: 0;
  gap: 8px;
  align-items: center;
  position: relative;
  z-index: 2;
}

.notification-item .notification-content .notification-timestamp {
  font-size: 13px;
  color: #888;
}

.notification-item .notification-content .message-read {
  font-weight: 400;
}

.notification-item .notification-actions .read-toggle-button,
.notification-item .notification-actions .delete-button {
  color: #757575;
  transition: color 0.2s ease-in-out;
}
```

```
.notification-item .notification-actions .read-toggle-button:hover,
.notification-item .notification-actions .delete-button:hover {
  color: #333;
}

.notification-item .notification-actions .delete-button {
  color: #999;
}

.notification-item .notification-actions .delete-button:hover {
  color: #f44336;
}

.mat-divider {
  border-top-color: #eee;
}

.empty-state {
  text-align: center;
  padding: 40px 20px;
  color: #777;
  display: flex;
  flex-direction: column;
  align-items: center;
  justify-content: center;
  height: 100%;
}
```

```
.empty-state .empty-icon {
  font-size: 80px;
  width: 80px;
  height: 80px;
  color: #bdbdbd;
  margin-bottom: 24px;
}

.empty-state .empty-message {
  font-size: 1.3em;
  font-weight: 500;
  margin-bottom: 8px;
}

.empty-state .empty-sub-message {
  font-size: 1em;
  margin-bottom: 24px;
  color: #999;
}

.empty-state .refresh-button {
  background-color: #2196f3;
  color: white;
  font-weight: 500;
  padding: 0 24px;
  height: 44px;
  border-radius: 4px;
```

```
  transition: background-color 0.2s ease-in-out;
}

.empty-state .refresh-button:hover {
  background-color: #64b5f6;
}

.empty-state .refresh-button mat-icon {
  margin-right: 8px;
}

body ::ng-deep .mat-badge-content {
  font-family: 'Roboto', sans-serif;
}
```

Cutomer-Notification.html:

```
<div class="notification-dashboard mat-elevation-z2">
  <mat-toolbar color="primary" class="notification-toolbar">
    <mat-icon
      class="header-icon"
      matBadge="{{ unreadNotificationCount }}"
      matBadgeColor="accent"
      [attr.aria-hidden]="false"
      [matBadgeHidden]="unreadNotificationCount === 0">
      notifications
    </mat-icon>
    <h2 class="header-title">Alerts & Notifications</h2>
    <span class="toolbar-spacer"></span>
```

```
  <button mat-flat-button color="accent" class="mark-all-read-button"
(click)="markAllAsRead()"
    [disabled]="!hasUnreadNotifications" matTooltip="Mark all notifications as read">
    <mat-icon>done_all</mat-icon> Mark All
  </button>
 </mat-toolbar>

 <div class="notification-list-container">
  <ng-container *ngIf="notifications.length > 0; else noNotifications">
   <mat-list>
    <ng-container *ngFor="let notification of notifications; let last = last; let i = index">
     <mat-list-item class="notification-item" [ngClass]="{ 'unread': !notification.read }">
      <input type="checkbox" id="expand-toggle-{{notification.notificationId || i}}"
class="expand-toggle-checkbox">
      <label for="expand-toggle-{{notification.notificationId || i}}" class="expand-toggle-
label"></label>

      <div matListItemTitle class="notification-content">
       <span class="notification-message" [class.message-read]="notification.read">{{
notification.message }}</span>
       <span class="notification-timestamp">{{ notification.createdAt | date:'MMM d,
h:mm a' }}</span>
      </div>

      <div matListItemMeta class="notification-actions">
       <button mat-icon-button class="read-toggle-button"
        (click)="$event.stopPropagation(); toggleReadStatus(notification)"
        matTooltip="{{ notification.read ? 'Mark as Unread' : 'Mark as Read' }}">
        <mat-icon>{{ notification.read ? 'mail_outline' : 'mark_email_read' }}</mat-icon>
```

```
          </button>

<button mat-icon-button class="delete-button"
(click)="$event.stopPropagation(); dismissNotification(notification.notificationId)"
matTooltip="Delete Notification">
<mat-icon>delete</mat-icon>
</button>

         </div>
        </mat-list-item>
        <mat-divider *ngIf="!last"></mat-divider>
       </ng-container>
     </mat-list>
   </ng-container>

   <ng-template #noNotifications>
     <div class="empty-state">
       <mat-icon class="empty-icon">inbox</mat-icon>
       <p class="empty-message">No notifications found.</p>
       <p class="empty-sub-message">Your inbox is wonderfully empty.</p>
       <button mat-flat-button color="accent" class="refresh-button">
         <mat-icon>refresh</mat-icon> Fetch Notifications
       </button>
     </div>
   </ng-template>
  </div>
</div>
```

Customer-Notification.Ts:

```
import { Component, OnInit } from '@angular/core';
import { CommonModule } from '@angular/common';
import { MatCardModule } from '@angular/material/card';
import { MatIconModule } from '@angular/material/icon';
import { MatButtonModule } from '@angular/material/button';
import { MatDividerModule } from '@angular/material/divider';
import { MatListModule } from '@angular/material/list';
import { MatToolbarModule } from '@angular/material/toolbar';
import { MatTooltipModule } from '@angular/material/tooltip';
import { MatBadgeModule } from '@angular/material/badge';
import { NotificationResponseDto } from '../../auth/dtos/notificationResponseDto.dto';
import { CustomerService } from '../customer.service';
import { ApiResponse } from '../../../shared/api-response.interface';

@Component({
 selector: 'app-customer-notifications',
 standalone: true,
 imports: [
  CommonModule,
  MatCardModule,
  MatIconModule,
  MatButtonModule,
  MatDividerModule,
  MatListModule,
  MatToolbarModule,
  MatTooltipModule,
  MatBadgeModule
```

```
 ],
 templateUrl: './customer-notifications.component.html',
 styleUrl: './customer-notifications.component.css'
})
export class CustomerNotificationsComponent implements OnInit {
 totalNotifications: number = 0; // Total number of notifications for display
 notifications: NotificationResponseDto[] = [];

 constructor(private customerService: CustomerService) {}

 ngOnInit(): void {
  this.getNotifications();
 }

 getNotifications(): void {
  console.log('Fetching notifications...');
  this.customerService.getNotifications().subscribe({
   next: (response: ApiResponse<NotificationResponseDto[]>) => {
    if (response.isSuccess) {
     this.notifications = response.data.map(notification => {
      // Convert UTC time to IST
      const utcDate = new Date(notification.createdAt);
      notification.createdAt = this.convertUtcToIst(utcDate);
      return notification;
     });

     // Sort notifications by IST time
     this.notifications.sort((a, b) => {
```

```
        const dateA = new Date(a.createdAt).getTime();
        const dateB = new Date(b.createdAt).getTime();
        return dateB - dateA; // Sort in descending order (latest first)
      });

      this.totalNotifications = this.notifications.length;
      console.log(`Total notifications fetched: ${this.totalNotifications}`);
    } else {
      console.error('Failed to fetch notifications:', response.message);
    }
  },
  error: (error) => {
    console.error('Error fetching notifications:', error);
  }
 });
}

 convertUtcToIst(utcDate: Date): string {
  // Convert UTC date to IST (Indian Standard Time)
  const istOffset = 5.5 * 60 * 60 * 1000; // IST is UTC+5:30
  const istDate = new Date(utcDate.getTime() + istOffset);
  return istDate.toLocaleString('en-US', { timeZone: 'Asia/Kolkata' });
}

 get hasUnreadNotifications(): boolean {
  return this.notifications.some(n => !n.read);
}
```

```
 get unreadNotificationCount(): number {
  return this.notifications.filter(n => !n.read).length;
 }

 toggleReadStatus(notification: NotificationResponseDto): void {
  notification.read = !notification.read;
  console.log(` Notification ${notification.notificationId} marked as ${notification.read ?
'read' : 'unread'}. `);
 }

 dismissNotification(notificationId: string | undefined): void {
  if (!notificationId) {
   console.warn("Dismiss called with undefined ID — aborting.");
   return;
  }

  this.notifications = [...this.notifications.filter(n => n.notificationId !== notificationId)];
  console.log(` Notification ${notificationId} dismissed (deleted). `);
 }

 debugLog(notification: NotificationResponseDto): void {
  console.log("Clicked notification:", JSON.stringify(notification, null, 2));
  this.dismissNotification(notification.notificationId);
 }

 markAllAsRead(): void {
  this.notifications.forEach(n => n.read = true);
  console.log('All notifications marked as read.');
```

```
 }
}
```

Web Part Of Customer :

Customer Controller:

Customer controller:

```
using System.Threading.Tasks;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using IMSIntrim.Domain.Models;
using IMSIntrim.Infrastructure.Persistance;
using IMSIntrim.Application.Interfaces;
using IMSIntrim.Applications.DTOs;
namespace IMSIntrim.Controllers
{
  [Route("api/customers")]
  [ApiController]
  public class CustomersController : ControllerBase
  {
    private readonly InsuranceDbContext _context;
    private readonly ICustomerServices _customerService;

    public CustomersController(InsuranceDbContext context, ICustomerServices
customerServices)
    {
      _context = context;
      _customerService = customerServices;
```

```
    }

    [HttpGet]
    [Authorize(Roles = Roles.Admin)]
    public async Task<ActionResult<PagedResult<CustomerProfileResponseDto>>>
GetAllCustomers(int page = 1, int size = 10)
    {
      var result = await _customerService.GetAllCustomersAsync(page, size);
      if (!result.IsSuccess) return NotFound(result);
      return Ok(result);
    }

    [HttpPost("register")]
    //[Authorize(Roles = Roles.Admin)]
    public async Task<IActionResult> AddCustomer([FromBody]
CustomerRegisterRequestDto dto)
    {
      var result = await _customerService.AddCustomerAsync(dto);
      if (!result.IsSuccess) return BadRequest(result);
      return Ok(result);
    }

    [HttpGet("{customerId}")]
    [Authorize(Roles = Roles.Admin)]
    public async Task<ActionResult<CustomerProfileResponseDto>>
GetCustomerByIdAsync(int customerId)
    {
      var result = await _customerService.GetCustomerByIdAsync(customerId);
      if (result == null) return NotFound(result);
```

```
        return Ok(result);
    }

    [HttpGet("profile")]
    public async Task<IActionResult> GetProfile()
    {
        var profile = await _customerService.GetProfileAsync();
        if (profile == null) return NotFound("Customer profile not found.");
        return Ok(profile);
    }

    [HttpPut("profile")]
    public async Task<IActionResult> UpdateProfile([FromBody]
CustomerProfileUpdateRequestDto dto)
    {
        var result = await _customerService.UpdateProfileAsync(dto);
        if (result == null) return NotFound("Customer not found");
        return Ok(result);
    }

    [HttpDelete("delete-account")]
    [Authorize(Roles = $"{Roles.Admin},{Roles.Customer}")]
    public async Task<IActionResult> DeleteAccountAsync()
    {
        var result = await _customerService.DeleteAccountAsync();

        if (!result.IsSuccess)
            return BadRequest(result.Message);
```

```
            return Ok(result);
        }
    }
}
```

Notification Controller:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using Microsoft.AspNetCore.Authorization;
using IMSIntrim.Domain.Models;
using IMSIntrim.Infrastructure.Persistance;
using IMSIntrim.Application.Interfaces;
using IMSIntrim.Applications.DTOs;

namespace IMSIntrim.Controllers
{
    [Route("api/notifications")]
    [ApiController]
    public class NotificationsController : ControllerBase
    {
        private readonly InsuranceDbContext _context;
        private readonly INotificationServices _notificationService;
```

```
        public NotificationsController(InsuranceDbContext context, INotificationServices
notificationService)
        {
          _context = context;
          _notificationService = notificationService;
        }

        [HttpGet("user")]
      [Authorize(Roles = $"{Roles.Customer},{Roles.Agent}")]
      public async Task<IActionResult> GetMyNotifications()
        {
          var notes = await _notificationService.GetMyNotificationsAsync();
          if (!notes.IsSuccess) return NotFound(notes);
          return Ok(notes);
        }
      }
  }
```

Services :

CustomerService.cs:

```
using AutoMapper;
using IMSIntrim.Application.Interfaces;
using IMSIntrim.Applications.DTOs;
using IMSIntrim.Applications.Interfaces;
using IMSIntrim.Domain.Interfaces;
using IMSIntrim.Domain.Models;
using IMSIntrim.Infrastructure.Persistance.Repositories;
using IMSIntrim.Shared.Common;
```

```
using Microsoft.AspNetCore.Identity;

namespace IMSIntrim.Application.Services
{
   public class CustomerServices : ICustomerServices
   {
      private readonly ITokenService _tokenService;

      private readonly ICustomerRepository _customerRepo;
      private readonly IAgentRepository _agentRepo;
      private readonly IAvailablePolicyRepository _availablePolicyRepo;
      private readonly IPolicyRequestRepository _policyRequestRepo;
      private readonly IClaimRepository _claimRepo;
      private readonly IPolicyRepository _policyRepo;
      private readonly IUserRepository _userRepo;
      private readonly IRoleRepository _roleRepo;
      private readonly IUserRoleRepository _userRoleRepo;
      private readonly IPasswordHasherService _passwordHasherService;
      private readonly IAuthServices _authService;
      private readonly IMapper _mapper;

      //private readonly ILogger<AdminService> _logger;
      private readonly INotificationRepository _notificationRepo;

      public CustomerServices
        (
         ICustomerRepository customerRepo,
IAgentRepository agentRepo,
```

```
IAvailablePolicyRepository availablePolicyRepo,
IPolicyRequestRepository policyRequestRepo,
IClaimRepository claimRepo,
IPolicyRepository policyRepo,
IAuthServices authService,
IUserRepository userRepository,
IPasswordHasherService passwordHasher,
IRoleRepository roleRepository,
IUserRoleRepository userRoleRepository,
IAgentRepository agentRepository,
INotificationRepository notificationRepository,
ITokenService tokenService,
    //ILogger<AdminService> logger,

    IMapper mapper
      )
    {
      _customerRepo = customerRepo;
      _agentRepo = agentRepo;
      _availablePolicyRepo = availablePolicyRepo;
      _policyRequestRepo = policyRequestRepo;
      _claimRepo = claimRepo;
      _policyRepo = policyRepo;
      _authService = authService;
      _userRepo = userRepository;
      _passwordHasherService = passwordHasher;
      _roleRepo = roleRepository;
      _userRoleRepo = userRoleRepository;
```

```
        _agentRepo = agentRepository;
        _notificationRepo = notificationRepository;
        _mapper = mapper;
        _tokenService = tokenService;
       //  _logger = logger;
    }
    public async Task<OperationResult<CustomerRegisterResponseDto>>
AddCustomerAsync(CustomerRegisterRequestDto dto)
    {
        var result = await _authService.RegisterAsync(dto);
        return result.IsSuccess
            ? OperationResult<CustomerRegisterResponseDto>.Success("Customer
added.")
            : OperationResult<CustomerRegisterResponseDto>.Failure(result.Message);
    }

    public async Task<OperationResult<bool>> DeleteAccountAsync()
    {
        Guid? userId = _tokenService.GetUserIdFromCurrentRequest();
        if (userId == null)
        {
            return OperationResult<bool>.Failure("Unable to Get the UserId from the Jwt
Token");
        }
        var result = await _userRepo.SoftDeleteUserAsync((Guid)userId);
        return result.IsSuccess
            ? OperationResult<bool>.Success(true, "Customer account deleted")
            : OperationResult<bool>.Failure(result.Message);
    }
```

```
    public async
Task<OperationResult<PagedResult<CustomerProfileResponseDto>>>
GetAllCustomersAsync(int page, int size)
    {
      var allcustomer = await _customerRepo.GetAllAsync();
      if (!allcustomer.IsSuccess)
      {
        return
OperationResult<PagedResult<CustomerProfileResponseDto>>.Failure("Failed To fetch
the details of all  Customer");
      }
      var allCustomersList = allcustomer.Data?.ToList();
      if (allCustomersList == null)
      {
        return
OperationResult<PagedResult<CustomerProfileResponseDto>>.Failure("There is no
data found in the List");
      }
      var paged = allCustomersList.Skip((page - 1) * size).Take(size).ToList();

      if (paged != null && paged.Any())
      {
        var result = new PagedResult<CustomerProfileResponseDto>
        {
          Items = paged.Select(c => _mapper.Map<CustomerProfileResponseDto>(c)
          ).ToList(),
          PageNumber = page,
          PageSize = size,
          TotalCount = allCustomersList.Count
        };
```

```
            return
OperationResult<PagedResult<CustomerProfileResponseDto>>.Success(result);
        }
        return
OperationResult<PagedResult<CustomerProfileResponseDto>>.Failure("There is no
customers in the specific page");
    }

    public async Task<OperationResult<CustomerProfileResponseDto>>
GetCustomerByIdAsync(int customerId)
    {
        var customer = await _customerRepo.GetByIdAsync(customerId);
        if (!customer.IsSuccess)
        {
            return OperationResult<CustomerProfileResponseDto>.Failure($"  Failed To
fetch the Customer of Id{customerId}");

        }
        var customerResult = customer.Data;
        var result = _mapper.Map<CustomerProfileResponseDto>(customerResult);
        return OperationResult<CustomerProfileResponseDto>.Success(result);
    }

    public async Task<OperationResult<CustomerProfileResponseDto>>
GetProfileAsync()
    {
        int? customerId = _tokenService.GetCustomerIdFromCurrentRequest();
        if (customerId == null)
        {
```

```
            return OperationResult<CustomerProfileResponseDto>.Failure("Unable to Get the CustomerId from the Jwt Token");
        }
        var customer = await _customerRepo.GetByUserIdAsync((int)customerId);
        if (customer == null)
        {
            return OperationResult<CustomerProfileResponseDto>.Failure("Customer profile not found for the specified user ID.");
        }
        var profile = customer.Data;



        var result = _mapper.Map<CustomerProfileResponseDto>(profile);
         return OperationResult<CustomerProfileResponseDto>.Success(result);
    }
     public async Task<OperationResult<bool>> UpdateProfileAsync(CustomerProfileUpdateRequestDto updateCustomerProfile)
    {
        int? customerId = _tokenService.GetCustomerIdFromCurrentRequest();


        if (customerId == null)
        {
            return OperationResult<bool>.Failure("Failed To Fetch the CustomerId from the Jwt Token");
         }
         var customer = _mapper.Map<Customer>(updateCustomerProfile);
         await _customerRepo.UpdateAsync(customer, (int)customerId);
         return OperationResult<bool>.Success(true, "Profile updated successfully");
    }
```

```
  }
}
```

Notification.cs:

```
using AutoMapper;
using IMSIntrim.Application.Interfaces;
using IMSIntrim.Applications.DTOs;
using IMSIntrim.Applications.Interfaces;
using IMSIntrim.Domain.Interfaces;
using IMSIntrim.Domain.Models;
using IMSIntrim.Infrastructure.Persistance.Repositories;
using IMSIntrim.Shared.Common;
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Mvc;

namespace IMSIntrim.Application.Services
{
  public class NotificationServices : INotificationServices
  {
    private readonly ITokenService _tokenService;
    private readonly IMapper _mapper;
    private readonly INotificationRepository _notificationRepo;
    public NotificationServices(
INotificationRepository notificationRepository,
ITokenService tokenService,


    IMapper mapper
```

```
        )
    {
        _notificationRepo = notificationRepository;
        _mapper = mapper;
        _tokenService = tokenService;
     //   _logger = logger;
    }

    public async Task<OperationResult<IEnumerable<NotificationResponseDto>>>
GetMyNotificationsAsync()
    {
        int? agentId = _tokenService.GetAgentIdFromCurrentRequest();
        int? customerId = _tokenService.GetCustomerIdFromCurrentRequest();

        if (customerId == null && agentId == null)
        {
            return
OperationResult<IEnumerable<NotificationResponseDto>>.Failure("Unable to retrieve
notifications; no valid user ID found.");
        }
        IEnumerable<Notification> notifications;
        if (customerId.HasValue)
        {
            // Fetch notifications related to the customer
            var notes = await _notificationRepo.GetByCustomerIdAsync(customerId.Value);
            notifications = notes.Data;
        }
        else if (agentId.HasValue)
        {
```

```
            // Fetch notifications related to the agent
            var notes = await _notificationRepo.GetByAgentIdAsync(agentId.Value);
            notifications = notes.Data;
        }
        else
        {
            return OperationResult<IEnumerable<NotificationResponseDto>>.Failure("No
notifications found.");
        }
        if (notifications == null || !notifications.Any())
        {
            return OperationResult<IEnumerable<NotificationResponseDto>>.Failure("No
notifications found for the specified user.");
        }

        var result = notifications.Select(n =>
_mapper.Map<NotificationResponseDto>(n));
        return
OperationResult<IEnumerable<NotificationResponseDto>>.Success(result);

        //if (customerId == null || agentId==null)
        //{
        //    return
OperationResult<IEnumerable<NotificationResponseDto>>.Failure("Unable to Get the
AgentId or CustomerId from the Jwt Token");
        //}
        //var notes = await _notificationRepo.GetByAgentIdAsync((int)customerId);
        //if (notes.Data?.Any() != true)
        //{
```

```
        //    return OperationResult<IEnumerable<NotificationResponseDto>>.Failure("No
notifications found for the specified agent.");
        //}


        //var notifications = notes.Data;
        //if (notifications?.Any() != true)
        //{
        //    return OperationResult<IEnumerable<NotificationResponseDto>>.Failure("
No notifications Found");
        //}



        //var result = notifications.Select(n =>
_mapper.Map<NotificationResponseDto>(n));


        //return
OperationResult<IEnumerable<NotificationResponseDto>>.Success(result);
      }



   }
}
```

ICustomerRepository.cs:

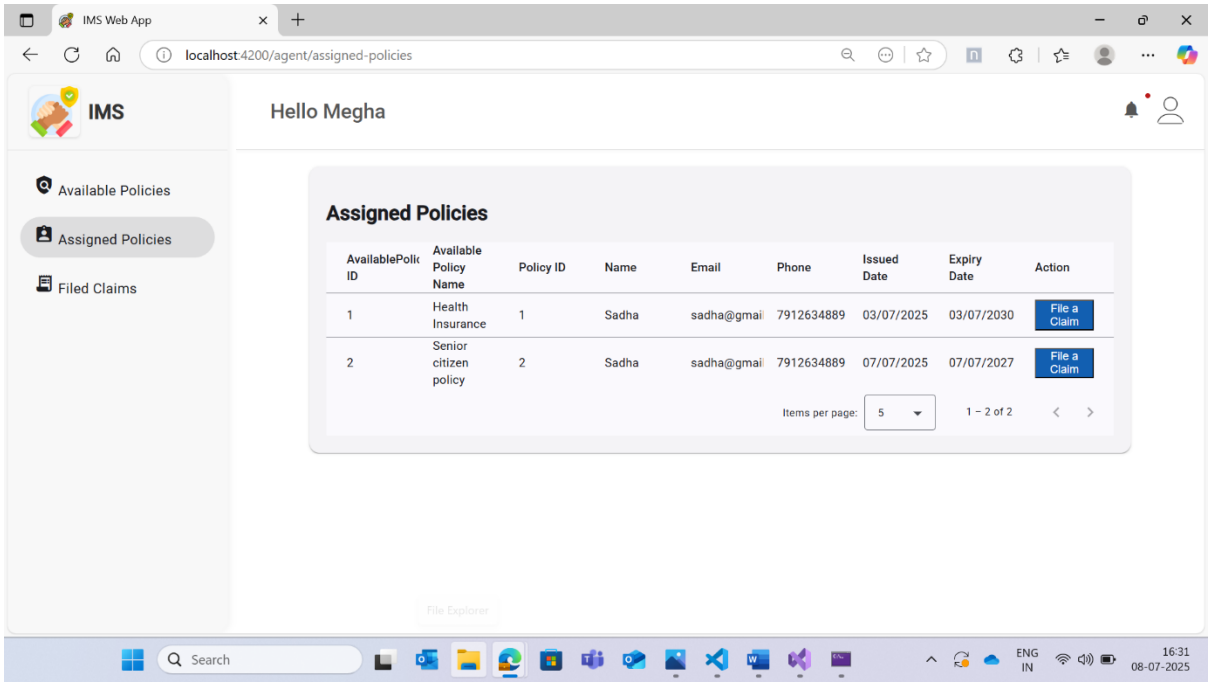
```
using IMSIntrim.Domain.Models;
using IMSIntrim.Shared.Common;
 namespace IMSIntrim.Domain.Interfaces
{
   public interface ICustomerRepository
   {
```

```
    public Task<OperationResult<IEnumerable<Customer>>> GetAllAsync();

    public Task<OperationResult<Customer?>> GetByIdAsync(int id);

    public Task<OperationResult<Customer?>> GetByUserIdAsync(int customerId);

    public Task<OperationResult<bool>> AddAsync(Customer customer);

    public Task<OperationResult<bool>> UpdateAsync(Customer customer, int
customerId);

    public Task<OperationResult<int?>> GetCustomerIdFromUserIdAsync(Guid?
userId);

  }

}
```

INotificationRepository.cs:

```
using IMSIntrim.Domain.Models;

using IMSIntrim.Shared.Common;


namespace IMSIntrim.Domain.Interfaces

{

  public interface INotificationRepository

  {

    public Task<OperationResult<IEnumerable<Notification>>> GetAllAsync();

    public Task<OperationResult<IEnumerable<Notification>>>
GetByCustomerIdAsync(int customerId);

    public Task<OperationResult<IEnumerable<Notification>>> GetByAgentIdAsync(int
agentId);

    public Task<OperationResult<bool>> AddAsync(Notification notification);

  }

}
```

The below are the files which are about the operations performed by the AGENT:



Agent-assigned-policies.css:

```
.assigned-policies{
   padding: 20px;
}
.file-claim-button{
   background-color: rgb(18, 95, 177);
   color:white;
}
```

Agent-assigned-policies.html:

```
<mat-card class="assigned-policies">
 <h2>Assigned Policies</h2>
 <mat-table [dataSource]="dataSource" class="mat-elevation-z8">
   <!-- Available Policy ID -->
   <ng-container matColumnDef="availablePolicyId">
    <mat-header-cell *matHeaderCellDef> AvailablePolicy ID </mat-header-cell>
    <mat-cell *matCellDef="let policy">
     {{ policy.availablePolicyId }}
```

```
    </mat-cell>
  </ng-container>
  <!-- Policy ID -->
  <ng-container matColumnDef="policyId">
   <mat-header-cell *matHeaderCellDef> Policy ID </mat-header-cell>
   <mat-cell *matCellDef="let policy"> {{ policy.policyId }} </mat-cell>
  </ng-container>

  <ng-container matColumnDef="availablePolicyName">
   <mat-header-cell *matHeaderCellDef>
    Available Policy Name
   </mat-header-cell>
   <mat-cell *matCellDef="let policy">
    {{ policy.availablePolicyName }}
   </mat-cell>
  </ng-container>



  <!-- Name -->
  <ng-container matColumnDef="name">
   <mat-header-cell *matHeaderCellDef> Name </mat-header-cell>
   <mat-cell *matCellDef="let policy"> {{ policy.customerName }} </mat-cell>
  </ng-container>
  <!-- Email -->
  <ng-container matColumnDef="customerEmail">
   <mat-header-cell *matHeaderCellDef> Email </mat-header-cell>
   <mat-cell *matCellDef="let policy"> {{ policy.customerEmail }} </mat-cell>
  </ng-container>
```

```
  <!-- Phone -->
  <ng-container matColumnDef="phone">
   <mat-header-cell *matHeaderCellDef> Phone </mat-header-cell>
   <mat-cell *matCellDef="let policy"> {{ policy.phone }} </mat-cell>
  </ng-container>
  <!-- Issued Date -->
  <ng-container matColumnDef="issuedDate">
   <mat-header-cell *matHeaderCellDef> Issued Date </mat-header-cell>
   <mat-cell *matCellDef="let policy">
    {{ policy.issuedDate | date : "dd/MM/yyyy" }}
   </mat-cell>
  </ng-container>
  <!-- Expiry Date -->
  <ng-container matColumnDef="expiryDate">
   <mat-header-cell *matHeaderCellDef> Expiry Date </mat-header-cell>
   <mat-cell *matCellDef="let policy">
    {{ policy.expiryDate | date : "dd/MM/yyyy" }}
   </mat-cell>
  </ng-container>
  <!-- Remove redundant mat-cell block -->
  <ng-container matColumnDef="action">
   <mat-header-cell *matHeaderCellDef> Action </mat-header-cell>
   <mat-cell *matCellDef="let policy">
    <button
     mat-raised-button
     class="file-claim-button"
     (click)="fileClaim(policy)"
    >
```

```
        File a Claim
      </button>
    </mat-cell>
  </ng-container>
  <mat-header-row *matHeaderRowDef="displayedColumns"></mat-header-row>
  <mat-row *matRowDef="let row; columns: displayedColumns"></mat-row>
</mat-table>
  <mat-paginator [length]="totalPolicies"
                 [pageSize]="pageSize"
                 [pageSizeOptions]="pageSizeOptions"
                 (page)="onPageChange($event)"
                 aria-label="Select">
  </mat-paginator>
</mat-card>
```

Agent-Assigned-Policies.ts:

```
import { Component, ViewChild, AfterViewInit, OnInit } from '@angular/core';
import { MatTableModule } from '@angular/material/table';
import { MatCardModule } from '@angular/material/card';
import { CommonModule } from '@angular/common';
import { Router, RouterLink } from '@angular/router';
import { AgentAssignedPolicyResponseDto } from '../dtos/agentAssignedPolicy.dto';
import { MatTableDataSource } from '@angular/material/table';
import { MatPaginator, PageEvent } from '@angular/material/paginator';
import { MatSort, MatSortModule } from '@angular/material/sort';

import { MatInputModule } from '@angular/material/input'; // Import MatInputModule
for search input
import { MatFormFieldModule } from '@angular/material/form-field'; // Import
MatFormFieldModule
```

```
import { AgentService } from '../agent.service';


@Component({
  standalone: true,
  selector: 'app-assigned-policies',
  templateUrl: './assigned-policies.component.html',
  styleUrls: ['./assigned-policies.component.css'],
  imports: [
    MatTableModule,
    MatCardModule,
    CommonModule,
    MatSortModule,
    MatInputModule, // Needed for search input if you use matInput
    MatFormFieldModule, // Needed for mat-form-field
    MatPaginator
  ],
})
export class AssignedPoliciesComponent implements OnInit, AfterViewInit {
  displayedColumns: string[] =
['availablePolicyId','availablePolicyName','policyId',  'name', 'customerEmail', 'phone',
'issuedDate', 'expiryDate', 'action'];
  dataSource = new MatTableDataSource<AgentAssignedPolicyResponseDto>([]);


  @ViewChild(MatPaginator) paginator!: MatPaginator;
  @ViewChild(MatSort) sort!: MatSort;
  loggedInAgentId:number|null=null;


  searchText: string = '';
  totalPolicies: number = 0;
```

```
 pageSize = 5;
 pageSizeOptions: number[] = [5, 10, 20];

 constructor(private agentService: AgentService, private router: Router) {}

 ngOnInit(): void {
  this.agentService.getAgentProfile$().subscribe((profile) => {
   if (profile) {

    this.loggedInAgentId =
     typeof profile.agentId === 'string'
      ? Number(profile.agentId)
      : profile.agentId;
   } else {
    this.agentService.fetchAndStoreAgentProfile().subscribe();
   }
  });

  this.getAssignedPolicies();
 }

 ngAfterViewInit(): void {
  this.dataSource.paginator = this.paginator;
  this.dataSource.sort = this.sort;

  // Apply filter predicate for search functionality
  this.dataSource.filterPredicate = (data: AgentAssignedPolicyResponseDto, filter:
string) => {
```

```
    const dataStr = JSON.stringify(data).toLowerCase();
    return dataStr.indexOf(filter) !== -1;
  };
 }

 getAssignedPolicies(): void {
  this.agentService.getAllAssignedPolicies().subscribe({
   // ...
next: (response: any) => {
 if (response.isSuccess && response.data) {
  this.dataSource.data = response.data;
  console.log('Assigned policies loaded successfully', this.dataSource.data);
 }

 else if (response.isSuccess && response.data && response.data.items) {
  this.dataSource.data = response.data.items;
  this.totalPolicies = response.data.totalCount;
  console.log('Assigned policies loaded successfully', this.dataSource.data);
 }
 else {
  console.error('Failed to fetch assigned policies:', response.message || 'Unknown
error');
  this.dataSource.data = [];
  this.totalPolicies = 0;
 }
},
// ...,
    error: (error: any) => {
```

```
      console.error('Error fetching assigned policies:', error);
      this.dataSource.data = [];
      this.totalPolicies = 0;
    },
  });
}
 onPageChange(event: PageEvent): void {

}
 applyFilter(): void{
  this.dataSource.filter = this.searchText.trim().toLowerCase();
  if (this.dataSource.paginator) {
    this.dataSource.paginator.firstPage();
  }
}


 fileClaim(policy: AgentAssignedPolicyResponseDto): void {
    console.log(` Filing a claim for policy ID: ${policy.policyId}` );
    this.router.navigate(['/agent/file-a-claim'], {
      queryParams: {
        policyId: policy.policyId,
        agentId: this.loggedInAgentId,
        customerId: policy.customerId
      }
    });
  }
```

}

AgentController.cs:

```
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using IMSIntrim.Application.Interfaces;
using IMSIntrim.Applications.DTOs;
using IMSIntrim.Infrastructure.Persistance;
using System.Threading.Tasks;
using IMSIntrim.Applications.MiddleWares;

using IMSIntrim.Domain.Models;



namespace IMSIntrim.Controllers
{
  [Route("api/[controller]")]
  [ApiController]
  public class AgentController : ControllerBase
  {
    private readonly InsuranceDbContext _context;
    private readonly IAgentServices _agentService;

    public AgentController(InsuranceDbContext context, IAgentServices agentService)
    {
      _context = context;
      _agentService = agentService;
    }
```

```
    [HttpGet("admin/agents")]
    [Authorize(Roles = Roles.Admin)]
    public async Task<IActionResult> GetAllAgents(int page = 1, int size = 10)
    {
      var result = await _agentService.GetAllAgentsAsync(page, size);
      if (!result.IsSuccess) return NotFound(result);
      return Ok(result);
    }

    [HttpPost("admin/add")]
    [Authorize(Roles = Roles.Admin)]
    public async Task<IActionResult> AddAgent([FromBody] AgentRegisterRequestDto
dto)
    {
      var result = await _agentService.AddAgentAsync(dto);
      if (!result.IsSuccess) return BadRequest(result);
      return Ok(result);
    }

    [HttpGet("admin/{agentId}")]
    [Authorize(Roles = Roles.Admin)]
    public async Task<IActionResult> GetAgentById(int agentId)
    {
      var result = await _agentService.GetAgentByIdAsync(agentId);
      if (result == null) return NotFound();
      return Ok(result);
    }
```

```
    [HttpGet("profile")]
    [Authorize(Roles = Roles.Agent)]
    public async Task<IActionResult> GetProfile()
    {
      var profile = await _agentService.GetProfileAsync();
      if (profile == null) return NotFound("Agent profile not found");
      return Ok(profile);
    }

    [HttpPut("profile")]
    [Authorize(Roles = Roles.Agent)]
    public async Task<IActionResult> UpdateProfile([FromBody]
AgentProfileUpdateRequestDto dto)
    {
      var result = await _agentService.UpdateProfileAsync(dto);
      if (result == null) return NotFound("Agent not found");
      return Ok(result);
    }
  }
}
```

AgentServices.cs:

```
using AutoMapper;
using IMSIntrim.Application.Interfaces;
using IMSIntrim.Applications.DTOs;
using IMSIntrim.Domain.Interfaces;
using IMSIntrim.Domain.Models;
using IMSIntrim.Shared.Common;
using Microsoft.AspNetCore.Identity;
```

```
using IMSIntrim.Applications.Services;
using Microsoft.AspNetCore.Mvc;
using IMSIntrim.Applications.Interfaces;

namespace IMSIntrim.Application.Services
{
  public class AgentServices : IAgentServices
  {
    private readonly ICustomerRepository _customerRepo;
    private readonly IAgentRepository _agentRepo;
    private readonly IAvailablePolicyRepository _availablePolicyRepo;
    private readonly IPolicyRequestRepository _policyRequestRepo;
    private readonly IClaimRepository _claimRepo;
    private readonly IPolicyRepository _policyRepo;
    private readonly IUserRepository _userRepo;
    private readonly IRoleRepository _roleRepo;
    private readonly IUserRoleRepository _userRoleRepo;
    private readonly IPasswordHasherService _passwordHasher;
    private readonly IAuthServices _authService;
    private readonly IMapper _mapper;
    //private readonly ILogger<AgentServices> _logger;
    private readonly INotificationRepository _notificationRepo;
    private readonly ITokenService _tokenService;

    public AgentServices(
      ICustomerRepository customerRepo,
      IAgentRepository agentRepo,
      IAvailablePolicyRepository availablePolicyRepo,
```

```
        IPolicyRequestRepository policyRequestRepo,
        IClaimRepository claimRepo,
        IPolicyRepository policyRepo,
        IAuthServices authService,
        IUserRepository userRepository,
        IPasswordHasherService passwordHasher,
        IRoleRepository roleRepository,
        IUserRoleRepository userRoleRepository,
        IAgentRepository agentRepository,
       INotificationRepository notificationRepository,
       ITokenService tokenService,
    //    ITokenService tokenService,
      // ILogger<AdminService> logger,

       IMapper mapper
       )
    {
       _customerRepo = customerRepo;
       _agentRepo = agentRepo;
       _availablePolicyRepo = availablePolicyRepo;
       _policyRequestRepo = policyRequestRepo;
       _claimRepo = claimRepo;
       _policyRepo = policyRepo;
       _authService = authService;
       _userRepo = userRepository;
       _passwordHasher = passwordHasher;
       _roleRepo = roleRepository;
       _userRoleRepo = userRoleRepository;
```

```
        _agentRepo = agentRepository;
        _notificationRepo = notificationRepository;
        _mapper = mapper;
        _tokenService = tokenService;
        //_logger = logger;

    }
    public async Task<OperationResult<AgentRegisterResponseDto>>
AddAgentAsync(AgentRegisterRequestDto dto)
    {
        var existingUser = await _userRepo.GetByUsernameAsync(dto.UserName);
        if (!existingUser.IsSuccess)
        {
            return
OperationResult<AgentRegisterResponseDto>.Failure(existingUser.Message);
        }

        var user = _mapper.Map<User>(dto);
        user.Id = Guid.NewGuid();
        user.IsDeleted = false;
        user.PasswordHash = _passwordHasher.HashPassword(dto.Password);

        var userAddingResult = await _userRepo.AddAsync(user);

        if (!userAddingResult.IsSuccess)
        {
            return
OperationResult<AgentRegisterResponseDto>.Failure(userAddingResult.Message);
        }
```

```
        var role = await _roleRepo.GetByNameAsync("Agent");
        if (role?.Data == null)
        {
            return OperationResult<AgentRegisterResponseDto>.Failure("Agent role
missing");
        }
        var userRole = _mapper.Map<UserRole>(user);
        _mapper.Map(role.Data, userRole);

        var userRoleResult = await _userRoleRepo.AddAsync(userRole);
        if (!userRoleResult.IsSuccess)
        {
            return
OperationResult<AgentRegisterResponseDto>.Failure(userRoleResult.Message);
        }

        var agent = _mapper.Map<Agent>(dto);
        _mapper.Map(user, agent);

        var agentResult = await _agentRepo.AddAsync(agent);
        if (!agentResult.IsSuccess)
        {
            return
OperationResult<AgentRegisterResponseDto>.Failure(agentResult.Message);
        }
        var notification = new Notification
        {
            CustomerId = null,
```

```
            AgentId = user.Agent.AgentId,
            Message = $"You have been registered by an admin. Welcome to the team, {dto.Name}! 🎉 We're excited to have you on board.",
            CreatedAt = DateTime.UtcNow
        };
        await _notificationRepo.AddAsync(notification);

        return OperationResult<AgentRegisterResponseDto>.Success(new AgentRegisterResponseDto { Message = "Agent added successfully" });

    }

    //public Task<IActionResult> DeleteAgent(int id)
    //{
    //    throw new NotImplementedException();
    //}

    public async Task<OperationResult<AgentProfileResponseDto>> GetAgentByIdAsync(int agentId)
    {
        var agent = await _agentRepo.GetByIdAsync(agentId);
        if (!agent.IsSuccess)
        {
            return OperationResult<AgentProfileResponseDto>.Failure($"  Failed To fetch the Agent of Id{agentId}");

        }
        var agentresult = agent.Data;
```

```
        var result = _mapper.Map<AgentProfileResponseDto>(agentresult);


        return OperationResult<AgentProfileResponseDto>.Success(result);
    }


    public async Task<OperationResult<PagedResult<AgentProfileResponseDto>>>
GetAllAgentsAsync(int page, int size)
    {
        var result = await _agentRepo.GetAllAsync();


        if (!result.IsSuccess)
        {
            return OperationResult<PagedResult<AgentProfileResponseDto>>.Failure("
Failed To fetch the Agent");
        }


        var all = result.Data?.ToList();
        if (all == null)
        {
            return OperationResult<PagedResult<AgentProfileResponseDto>>.Failure(" No
Data Avaiable in the Agent List");
        }


        var paged = all.Skip((page - 1) * size).Take(size).ToList();




        if (paged != null && paged.Any())
        {
```

```
            var agentResult = new PagedResult<AgentProfileResponseDto>
            {
                Items = paged.Select(a => _mapper.Map<AgentProfileResponseDto>(a)),
                PageNumber = page,
                PageSize = size,
                TotalCount = all.Count
            };

            return
OperationResult<PagedResult<AgentProfileResponseDto>>.Success(agentResult);
        }
        return OperationResult<PagedResult<AgentProfileResponseDto>>.Failure("There
is no Agent in the specific page");



    }

    public async Task<OperationResult<AgentProfileResponseDto>> GetProfileAsync()
    {
        int? agentId = _tokenService.GetAgentIdFromCurrentRequest();
        if (agentId == null)
        {
            return OperationResult<AgentProfileResponseDto>.Failure("Unable to Get the
AgentId from the Jwt Token");
        }
        var agent = await _agentRepo.GetByUserIdAsync((int)agentId);
        if (agent.Data == null)
        {
```

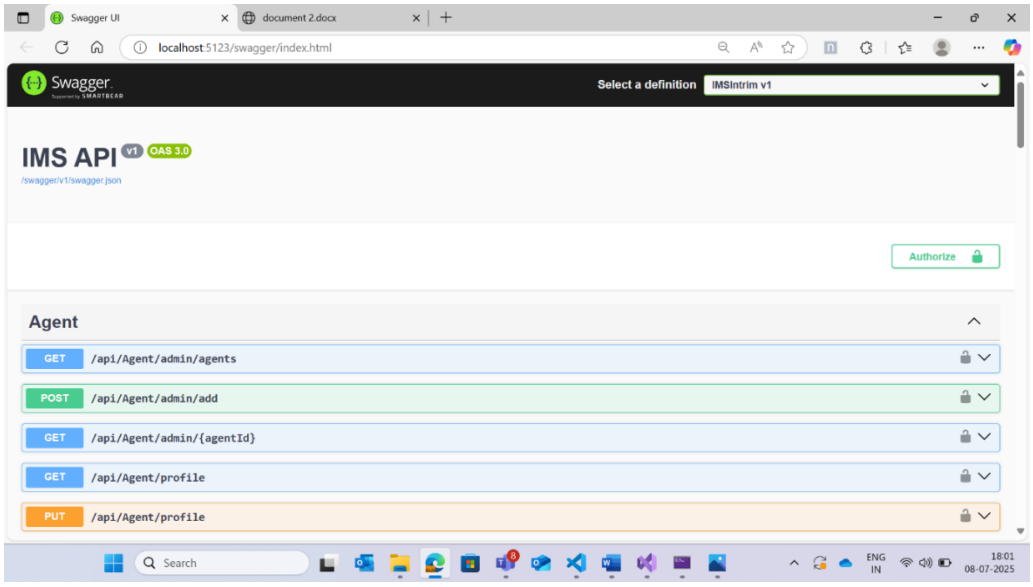
```
            return OperationResult<AgentProfileResponseDto>.Failure("Agent Profile not
found");
        }
         var result = _mapper.Map<AgentProfileResponseDto>(agent.Data);
        return OperationResult<AgentProfileResponseDto>.Success(result);
      }
       public async Task<OperationResult<bool>>
UpdateProfileAsync(AgentProfileUpdateRequestDto updateAgentProfileDto)
     {
        int? agentId = _tokenService.GetAgentIdFromCurrentRequest();
        if (agentId == null)
        {
           return OperationResult<bool>.Failure("Unable to Fetch the AgentId from The
Token ");
        }
        var agent = _mapper.Map<Agent>(updateAgentProfileDto);

        await _agentRepo.UpdateAsync(agent, (int)agentId);
        return OperationResult<bool>.Success(true, "Profile Updated Successfully");
     }
   }
}
```

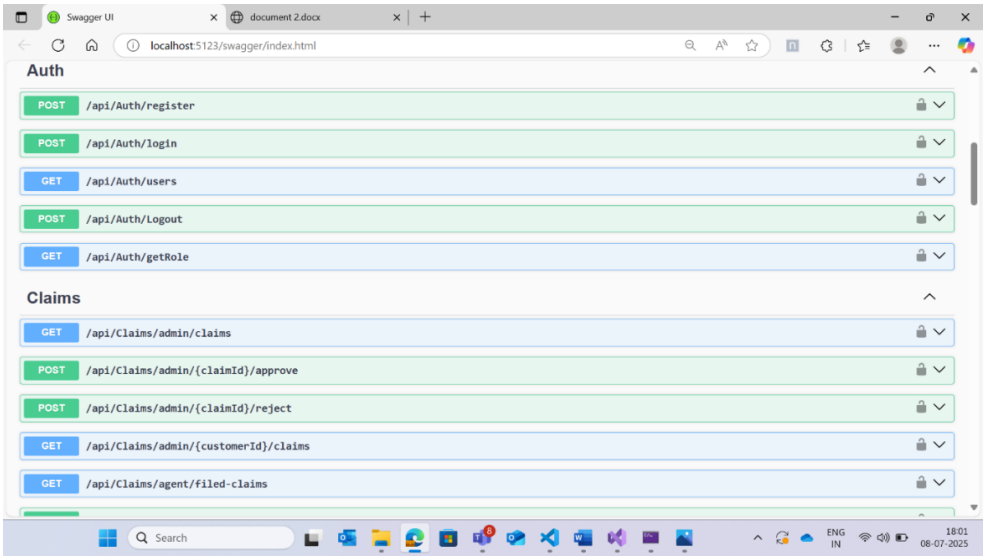
IAgentRepository.cs:

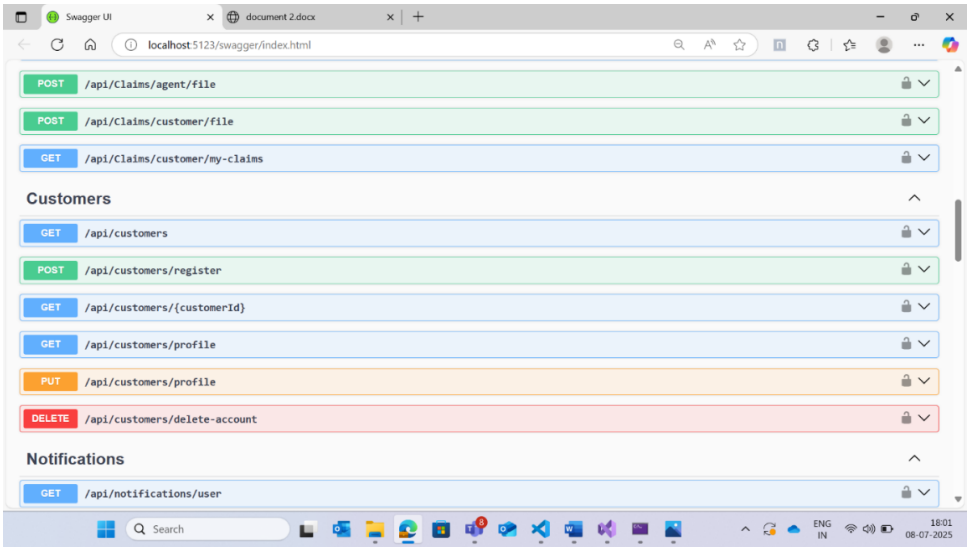
```
using IMSIntrim.Domain.Models;
using IMSIntrim.Shared.Common;

namespace IMSIntrim.Domain.Interfaces
{
```

```
  public interface IAgentRepository
  {
    public Task<OperationResult<IEnumerable<Agent>>> GetAllAsync();
    public Task<OperationResult<Agent?>> GetByIdAsync(int id);
    public Task<OperationResult<Agent>> GetByUserIdAsync(int agentId);
    public Task<OperationResult<bool>> AddAsync(Agent agent);
    public  Task<OperationResult<bool>> UpdateAsync(Agent agent, int agentId);
    public Task<OperationResult<bool>> DeleteAsync(int id);
    public  Task<OperationResult<int?>> GetAgentIdFromUserIdAsync(Guid? userId);
  }
}
```

The below are web API Part







Program.cs:

```
using IMSIntrim.Applications.DependencyInjections;
using System.Text;
using IMSIntrim.Infrastructure.Persistance;
using Microsoft.AspNetCore.Authentication.JwtBearer;
using Microsoft.EntityFrameworkCore;
using Microsoft.IdentityModel.Tokens;
using IMSIntrim.Applications.Interfaces;
using IMSIntrim.Applications.Services;
using IMSIntrim.Applications.Utils;
using IMSIntrim.Domain.Interfaces;
using IMSIntrim.Infrastructure.Persistance.Repositories;
using IMSIntrim.Application.Interfaces;
using IMSIntrim.Application.Services;
using Microsoft.OpenApi.Models;
using IMSIntrim.Applications.MiddleWares;
using Serilog;
using Newtonsoft.Json;
using System.Net;
using Serilog.Events;
using Serilog.Formatting.Json;

namespace IMSIntrim
{
  public class Program
  {
    public static void Main(string[] args)
    {
```

```
var builder = WebApplication.CreateBuilder(args);
builder.Services.AddControllers();
// Learn more about configuring Swagger/OpenAPI at
https://aka.ms/aspnetcore/swashbuckle
builder.Services.AddEndpointsApiExplorer();


builder.Services.AddSwaggerGen(c =>
{
  c.SwaggerDoc("v1", new OpenApiInfo { Title = "IMS API", Version = "v1" });
  c.AddSecurityDefinition("Bearer", new OpenApiSecurityScheme
  {
    In = ParameterLocation.Header,
    Description = "Enter 'Bearer' followed by space and JWT token",
    Name = "Authorization",
    Type = SecuritySchemeType.ApiKey,
    Scheme = "Bearer"
  });
  c.AddSecurityRequirement(new OpenApiSecurityRequirement
{
  {
  new OpenApiSecurityScheme
  {
    Reference=new OpenApiReference
    {
      Type=ReferenceType.SecurityScheme,
      Id="Bearer"
    }
  },
```

```
            new string[] { }
        }
    });
        });
        builder.Services.AddSwaggerGen();
        var xmlFile =
$"{System.Reflection.Assembly.GetExecutingAssembly().GetName().Name}.xml";
        var xmlPath = Path.Combine(AppContext.BaseDirectory, xmlFile);
        builder.Services.AddSwaggerGen(c =>
        {
            c.IncludeXmlComments(xmlPath);
        });
        builder.Services.AddDbContext<InsuranceDbContext>(options =>
options.UseSqlServer(builder.Configuration.GetConnectionString("DefaultConnection"
)));
        builder.Services.AddApplicationServices(builder.Configuration);
        builder.Services.AddAutoMapper(AppDomain.CurrentDomain.GetAssemblies());



        //builder.Services.AddInfrastructureservices(builder.Configuration);
        builder.Services.AddHttpContextAccessor();
        var jwtSettings = builder.Configuration.GetSection("JwtSettings");
        var secretKey = jwtSettings["SecretKey"];
        var key = Encoding.ASCII.GetBytes(secretKey);

        builder.Services.AddAuthentication(options =>
        {
            options.DefaultAuthenticateScheme =
JwtBearerDefaults.AuthenticationScheme;
```

```
        options.DefaultChallengeScheme = JwtBearerDefaults.AuthenticationScheme;
    })
    .AddJwtBearer(options =>
    {
        options.TokenValidationParameters = new TokenValidationParameters
        {
            ValidateIssuer = true,
            ValidateAudience = true,
            ValidateLifetime = true,
            ValidateIssuerSigningKey = true,
            ValidIssuer = jwtSettings["Issuer"],
            ValidAudience = jwtSettings["Audience"],
            IssuerSigningKey = new SymmetricSecurityKey(key)

        };
        options.Events = new JwtBearerEvents
        {
            OnMessageReceived = context =>
            {
                var token = context.Request.Cookies["jwt"];
                if (!string.IsNullOrEmpty(token))
                {
                    context.Token = token;
                }
                return Task.CompletedTask;
            }
        };
    });
```

```
builder.Services.AddAutoMapper(typeof(Program));
builder.Services.AddControllers();

var allowedOrigins =
builder.Configuration.GetValue<string>("allowedOrigins")!.Split(",");

builder.Services.AddCors(options =>
{
  options.AddPolicy("AllowFrontend", builder =>
  {
    builder.WithOrigins("http://localhost:4200")
        .AllowAnyHeader()
        .AllowAnyMethod()
        .AllowCredentials(); // Only if using cookies/sessions
  });
});
var app = builder.Build();
app.UseMiddleware<ExceptionMiddleware>();
app.UseExceptionHandler(errorApp =>
{
  errorApp.Run(async context =>
  {
    context.Response.StatusCode = (int)HttpStatusCode.InternalServerError;
    context.Response.ContentType = "application/json";

    var response = new { Message = "An unexpected error occurred." };
    await context.Response.WriteAsync(JsonConvert.SerializeObject(response));
  });
```

```
        });

        // Configure the HTTP request pipeline.
        if (app.Environment.IsDevelopment())
        {
           app.UseSwagger();
           app.UseSwaggerUI(c =>
           {
              c.SwaggerEndpoint("/swagger/v1/swagger.json", "IMSIntrim v1");
           });
        }
        app.UseDeveloperExceptionPage();
        app.UseHttpsRedirection();
        app.UseCors("AllowFrontend");
        app.UseAuthentication();
        app.UseAuthorization();
        app.MapControllers();

        app.Run();
     }
  }
}
```

This documentation serves as a foundational reference for the project lifecycle.