Auto Insurance System – Architecture and Code Walkthrough

Repository: https://github.com/Supritha-AV/Auto-Insurance

# Table of Contents

1. Overview

2. Solution Structure

3. Models

4. Services

5. Controllers (MVC)

6. Views (UI/UX)

7. API Project (Admin API)

8. Authentication, Authorization, Roles

9. Data Flow (UI → Backend → DB)

10. Endpoints and Status Codes

11. Design Patterns and Conventions

12. Diagrams

# 1. Overview

Explains architecture of MVC app + separate Admin API with EF Core, DI, Filters, and Razor Views.

# 2. Solution Structure

- Auto\_Insurance\_System (MVC)

- AutoInsuranceSystemAPI (Web API)

# 3. Models

## Users

UserId, Username, Password, Email, Role enum

## Policy

PolicyId, PolicyNumber, VehicleDetails, CoverageAmount, CoverageType, PremiumAmount, Start/End, PolicyStatus

## Claim

ClaimId, PolicyId, ClaimAmount, ClaimDate, ClaimStatus, AdjusterId

## Payment

PaymentId, PolicyId, PaymentAmount, PaymentDate, PaymentStatus

## SupportTicket

TicketId, UserId, AssignedAgentId, IssueDescription, TicketStatus, CreatedDate, ResolvedDate

# 4. Services

## UsersService

Register/Login/Verify/ChangePassword/GetAll using DbContext.Users

## PolicyService

CRUD on DbContext.Policy

## ClaimService

GetAll/GetDetails/Submit/Update on DbContext.Claims

## PaymentService

MakePayment validates against Policy.PremiumAmount; GetPayment(s)

## SupportTicketService

Create/Assign/Resolve/List with includes

# 5. Controllers (MVC)

## AuthController

Register, Login (sets Session UserName/UserRole), ForgotPassword, dashboards, Logout

## AdminController

Role filter ADMIN; user CRUD; policies/claims/payments/tickets admin flows; JSON admin endpoints

## AgentController

Role filter AGENT; limited user create; policies/claims/payments/tickets flows

## CustomerController

Role filter CUSTOMER; policy create calls Admin API; claims/payments/tickets flows

## HomeController

Landing/About/Privacy/Error/AccessDenied

# 6. Views (UI/UX)

Bootstrap layout in \_Layout.cshtml, session-aware navbar, role dashboards, forms with validation.

# 7. API Project (Admin API)

PoliciesController: /api/admin/policies with CRUD, returns 200/201/204/404; Swagger enabled.

# 8. Authentication, Authorization, Roles

Session-based login in AuthController. RoleAuthorizationFilter compares Session[UserRole] with required role and redirects to Home/AccessDenied. No ASP.NET Identity.

# 9. Data Flow (UI → Backend → DB)

Forms → Controllers → Services → EF Core DbContext → SQL. Customer policy create → Admin API via HttpClient; rest use MVC services.

# 10. Endpoints and Status Codes

|  |  |  |  |
| --- | --- | --- | --- |
| Method | Path | Desc | Responses |
| GET | /api/admin/policies | List | 200 |
| GET | /api/admin/policies/{id} | Get by id | 200,404 |
| POST | /api/admin/policies | Create | 201 |
| PUT | /api/admin/policies/{id} | Update | 204,404 |
| DELETE | /api/admin/policies/{id} | Delete | 204,404 |

# 11. Design Patterns and Conventions

- MVC separation with Services and EF Core repository via DbContext

- Dependency Injection registration in Program.cs

- Custom Filters for exceptions and role-based authorization

- Swagger in API project

# 12. Diagrams

See high-level ASCII diagram:

UI (Razor Views)  
 ↓  
Controllers (Auth/Admin/Agent/Customer/Home)  
 ↓ (DI)  
Services (Users/Policy/Claim/Payment/Ticket)  
 ↓ (EF Core)  
AutoInsuranceDbContext → SQL Server  
  
Customer Policy Create → HttpClient → Admin API (/api/admin/policies) → AutoInsuranceApiDbContext → SQL Server