Water Complaint Management System

1. PROJECT OVERVIEW

The **Water Complaint Management System** is a comprehensive web-based application developed using ASP.NET Core MVC framework. The system addresses the critical challenge of managing municipal water supply complaints efficiently by providing a centralized platform for citizens to report water-related issues and enabling municipal authorities to track, assign, and resolve these complaints systematically.

Problem Statement

Municipal water supply issues are prevalent in urban areas, yet citizens often lack an efficient mechanism to report problems. Traditional complaint systems require physical visits to municipal offices, leading to:

- Delayed complaint registration and response times
- Lack of transparency in complaint tracking
- No accountability for resolution
- Poor communication between citizens and authorities
- Difficulty in prioritizing urgent issues

Proposed Solution

This project implements a full-stack web application that:

- Enables online complaint filing with photo upload capability
- Provides real-time status tracking and updates
- Implements priority-based complaint categorization
- Organizes complaints by municipal wards for efficient resource allocation
- Allows worker assignment and accountability tracking
- Includes comprehensive filtering and search functionality

2. OBJECTIVE

Primary Objectives

1. Develop a functional **Water Complaint Management System** for citizens to file and track complaints.
2. Demonstrate proficiency in **ASP.NET Core MVC** with complete **CRUD** operations and **Entity Framework Core** integration.
3. Build a **scalable and maintainable** web application using a normalized database and RESTful API design.

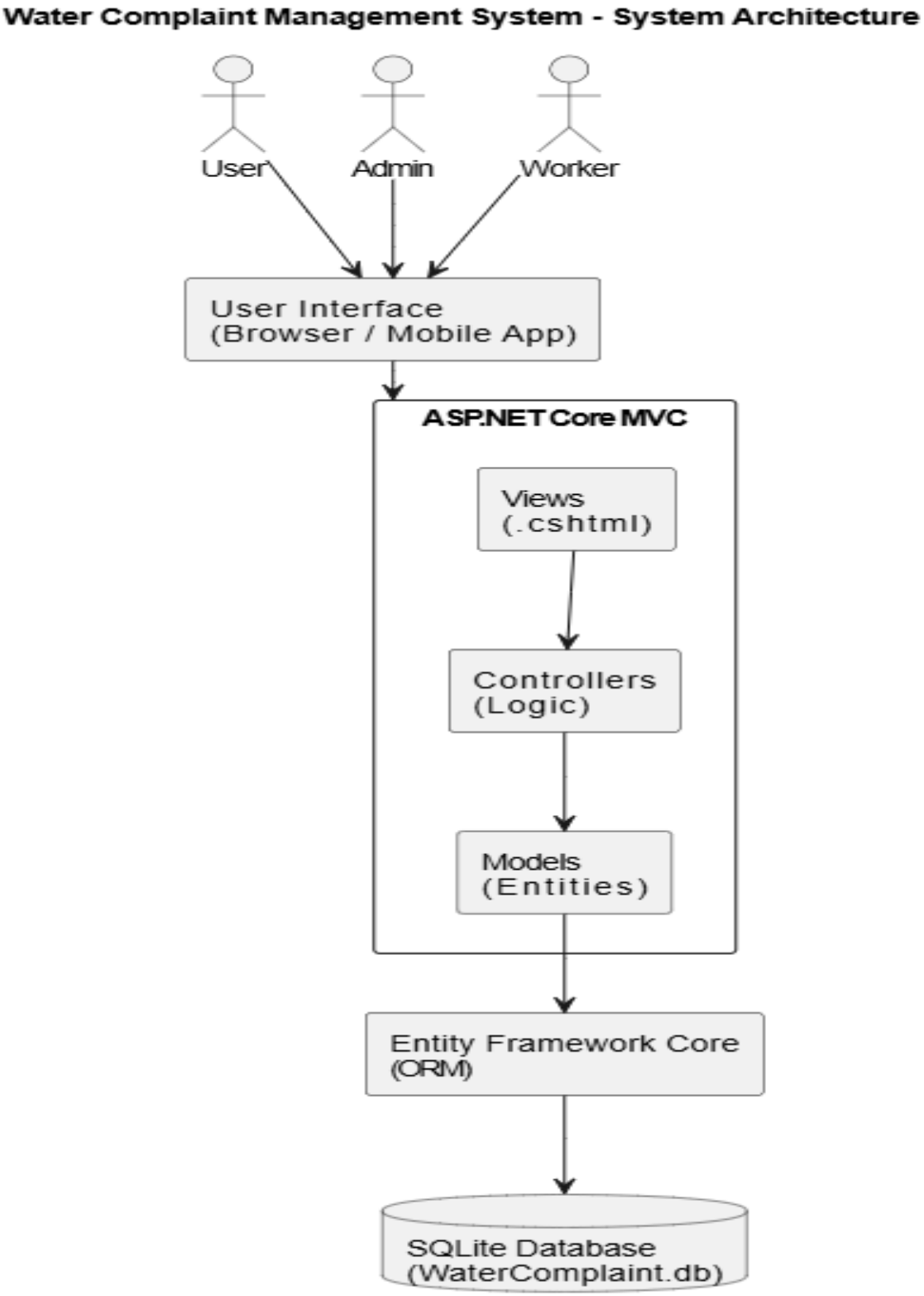
Secondary Objectives

1. Enhance **user experience** through responsive design, real-time validation, and easy navigation.
2. Ensure **data integrity and security** via input validationand HTTPS support.
3. Provide **educational value** by combining course practicals, thorough documentation, and reusable components.

3. TECHNOLOGY STACK

Category	Technology	Version	Purpose
Backend Framework	ASP.NET Core MVC	8.0	Server-side web framework
Programming Language	C#	12	Backend logic and business rules
ORM	Entity Framework Core	8.0	Database operations and migrations
Database	SQLite	3.x	Embedded relational database
Frontend Framework	Bootstrap	5.3	Responsive UI components
View Engine	Razor	Core 8.0	Dynamic HTML generation
Styling	CSS3	-	Custom styling and animations
Client Script	JavaScript / jQuery	3.x	Client-side validation and interactivity
Markup	HTML5	-	Page structure
Development IDE	Visual Studio Code	Latest	Code editor and debugging
CLI Tool	.NET CLI	8.0	Project management and building
Version Control	Git	Latest	Source code management
Package Manager	NuGet	Latest	Dependency management

4. SYSTEM ARCHITECTURE



Architectural Pattern: Model-View-Controller (MVC)

Application Flow

1. User Request (HTTP/HTTPS)
   ↓
2. ASP.NET Core Pipeline (Middleware)
   ↓
3. Routing (URL to Controller/Action mapping)
   ↓
4. Controller (Business Logic)
   ↓
5. Model (Data Access via EF Core)
   ↓
6. Database (SQLite)
   ↓
7. Model (Data returned)
   ↓
8. View (Razor rendering)
   ↓
9. HTTP Response (HTML/JSON)
   ↓
10. User receives response

5. MODULES / FUNCTIONAL COMPONENTS

Module	Description	Key Features	User Role
Home Module	Landing page and navigation hub	- Time-based greeting (Morning/Afternoon/Evening) - Quick action cards - College information page	All Users
Complaint Management	Core module for complaint CRUD operations	- Create new complaints - View all complaints with filtering - Edit complaint details - Delete complaints	Citizens, Workers, Admin
Ward Management	Municipal ward organization	- Create/Edit/Delete wards - View ward details - Track complaints per war	Admin, Workers
Citizen Module	Citizen information management	- Store citizen details - Link citizens to complaints - Contact information tracking	System (Backend)
Worker Module	Municipal worker management	- Worker profiles - Ward assignment - Complaint assignment tracking - Worker performance monitoring	Admin
Photo Upload Module	File upload functionality	- Upload complaint photos - Store in server directory	Citizens
Filter & Search	Advanced filtering capabilities	- Filter by status (Pending/InProgress/Resolved/Rejected) - Filter by priority (Low/Medium/High/Critical)	All Users
RESTful API	API endpoints for external integration	- GET all complaints - GET single complaint - POST create complaint - PUT update complaint - DELETE complaint	External Apps
Session Management	User state persistence	- Store filter preferences - Track user activity - 30-minute idle timeout	All Users

6. DATABASE DESIGN

Tables Schema

Table 1: Wards

Column Name	Data Type	Constraints	Description
Id	INTEGER	PRIMARY KEY, AUTO INCREMENT	Unique ward identifier
Name	TEXT	NOT NULL, MAX 100 chars	Ward name (e.g., "Ward 1")
Area	TEXT	NOT NULL, MAX 200 chars	Area description (e.g., "Central Business District")
Description	TEXT	NULL	Optional ward description

Table 2: Workers

Column Name	Data Type	Constraints	Description
Id	INTEGER	PRIMARY KEY, AUTO INCREMENT	Unique worker identifier
Name	TEXT	NOT NULL, MAX 100 chars	Worker full name
Phone	TEXT	NOT NULL	Contact phone number
Email	TEXT	NOT NULL	Email address
WardId	INTEGER	FOREIGN KEY (Wards.Id), NOT NULL	Assigned ward reference

Table 3: Citizens

Column Name	Data Type	Constraints	Description
Id	INTEGER	PRIMARY KEY, AUTO INCREMENT	Unique citizen identifier
Name	TEXT	NOT NULL, MAX 100 chars	Citizen full name
Phone	TEXT	NOT NULL	Contact phone number
Email	TEXT	NOT NULL	Email address
Address	TEXT	NOT NULL, MAX 500 chars	Residential address

Table 4: Complaints

Column Name	Data Type	Constraints	Description
Id	INTEGER	PRIMARY KEY, AUTO INCREMENT	Unique complaint identifier
Title	TEXT	NOT NULL, MAX 200 chars	Complaint title/subject
Description	TEXT	NOT NULL, MAX 1000 chars	Detailed problem description
Status	TEXT	NOT NULL, DEFAULT 'Pending'	Status: Pending/InProgress/Resolved/Rejected
Priority	TEXT	NOT NULL, DEFAULT 'Medium'	Priority: Low/Medium/High/Critical
CreatedDate	DATETIME	NOT NULL, DEFAULT NOW()	Complaint creation timestamp
ResolvedDate	DATETIME	NULL	Resolution timestamp
CitizenId	INTEGER	FOREIGN KEY (Citizens.Id), NOT NULL	Citizen who filed complaint
WardId	INTEGER	FOREIGN KEY (Wards.Id), NOT NULL	Ward where issue exists

Table 5: ComplaintPhotos

Column Name	Data Type	Constraints	Description
Id	INTEGER	PRIMARY KEY, AUTO INCREMENT	Unique photo identifier
FileName	TEXT	NOT NULL	Original filename
FilePath	TEXT	NOT NULL	Server storage path
UploadedDate	DATETIME	NOT NULL, DEFAULT NOW()	Upload timestamp
ComplaintId	INTEGER	FOREIGN KEY (Complaints.Id), NOT NULL	Parent complaint reference

Relationships

1. **One Ward → Many Complaints** (One-to-Many)
2. **One Ward → Many Workers** (One-to-Many)
3. **One Citizen → Many Complaints** (One-to-Many)
4. **One Worker → Many Complaints** (One-to-Many)
5. **One Complaint → Many ComplaintPhotos** (One-to-Many)

7. IMPLEMENTATION / WORKING

1. Application Startup

1. User runs: dotnet run
2. Program.cs executes
3. Kestrel web server starts on ports 5000 (HTTP) and 5001 (HTTPS)
4. Middleware pipeline configured:
   - Static Files → Session → Response Caching → Routing → Authorization
5. Database connection established (SQLite)
6. Application ready to accept requests

2. Home Page Access

1. User navigates to: http://localhost:5000
2. Route matches: HomeController → Index()
3. Controller executes time-based logic:
   - Gets current hour
   - Determines greeting (Morning/Afternoon/Evening)
   - Selects appropriate image
   - Sets ViewData, ViewBag, TempData
4. Index.cshtml view renders with dynamic content
5. Bootstrap-styled HTML returned to browser
6. Page displays with greeting and action cards

3. Viewing Complaints (Read Operation)

1. User clicks "Complaints" in navigation
2. Route: ComplaintsController → Index()
3. Controller queries database:
   - Fetches all complaints with Include() for related data
   - Applies filters if query string present (status/priority/ward)
   - Orders by CreatedDate descending
4. Session stores filter preferences
5. ViewBag populated with filter dropdowns
6. Index.cshtml renders data table with complaints
7. Extension methods apply badge styling (GetStatusBadgeClass)
8. HTML table displayed with action buttons

4. Creating Complaint (Create Operation)

1. User clicks "Create New Complaint"
2. Route: ComplaintsController → Create() [GET]
3. Controller populates dropdowns:
   - Citizens list from database
   - Wards list from database
   - Workers list from database
4. Create.cshtml view renders form
5. User fills form and uploads photo (optional)
6. Form submits: ComplaintsController → Create() [POST]
7. Model validation executes:
   - Data annotations checked
   - ModelState validated
8. If valid:
   - Complaint saved to database
   - Photo processed and saved to wwwroot/uploads/
   - ComplaintPhoto record created
   - TempData success message set
   - Redirect to Index
9. If invalid:
   - Validation errors displayed
   - Form re-rendered with user input retained

5. Editing Complaint (Update Operation)

1. User clicks "Edit" button on complaint row
2. Route: ComplaintsController → Edit(id) [GET]
3. Controller fetches complaint by ID
4. Edit.cshtml pre-populates form with existing data
5. User modifies fields and submits
6. Route: ComplaintsController → Edit(id) [POST]
7. Model validation runs
8. If valid:
   - Database record updated via EF Core
   - ResolvedDate set if status = "Resolved"
   - Success message shown
   - Redirect to Index

6. Deleting Complaint (Delete Operation)

1. User clicks "Delete" button
2. Route: ComplaintsController → Delete(id) [GET]
3. Confirmation page displays complaint details
4. User confirms deletion

5. Route: ComplaintsController → DeleteConfirmed(id) [POST]
6. Database record removed
7. Success message displayed
8. Redirect to Index

7. API Request Handling

1. External app sends: GET http://localhost:5000/api/api/complaints
2. Route: ApiController → GetComplaints()
3. Controller queries database asynchronously
4. Includes related entities (Citizen, Ward, Worker)
5. Data serialized to JSON
6. JSON response returned with status code 200
7. External app receives complaint data

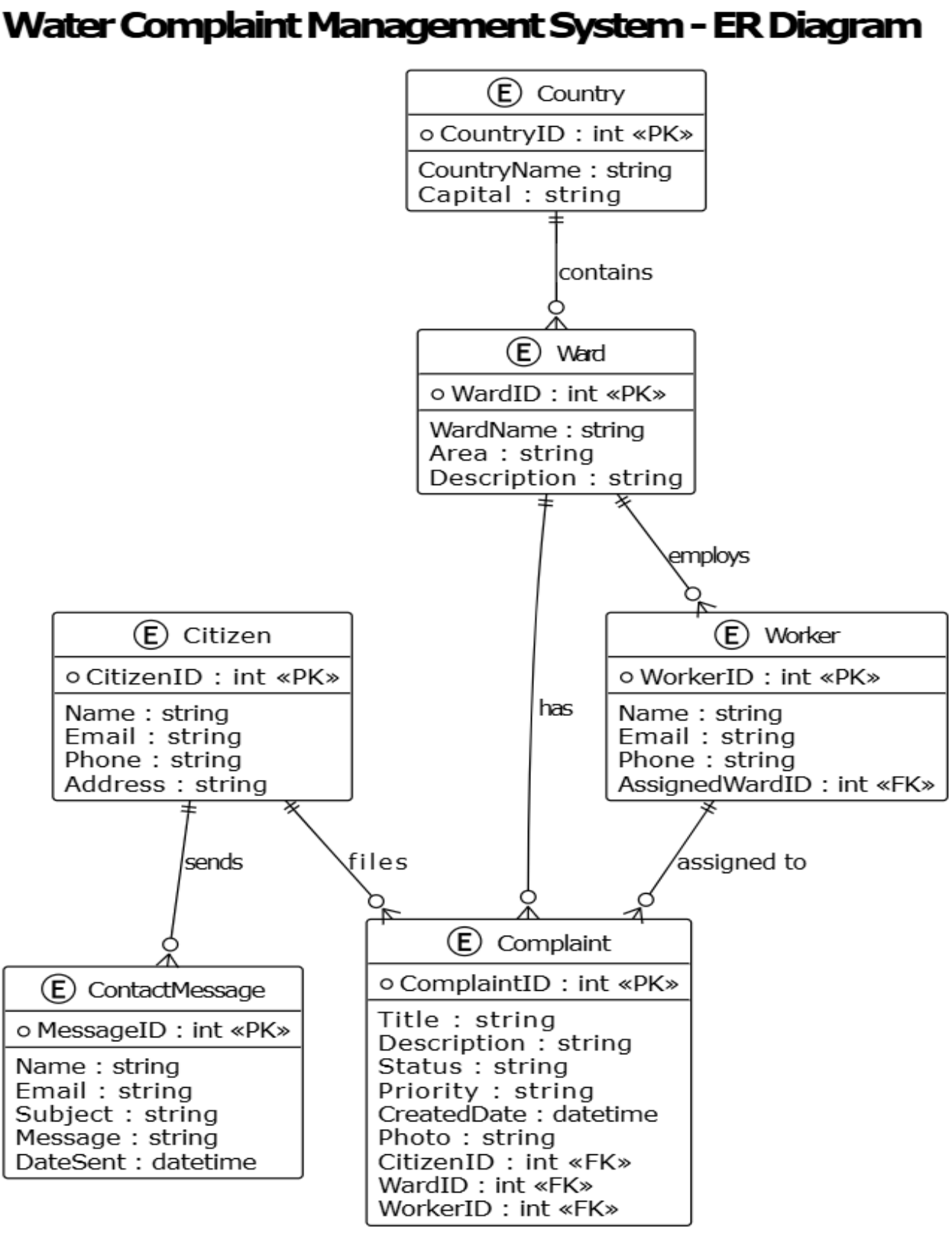
8. Session & Cookie Management

Session Flow:
1. User applies filter on complaints page
2. Controller stores filter in session: HttpContext.Session.SetString()
3. Session persists for 30 minutes (idle timeout)
4. Subsequent requests retrieve session data
5. Filter preferences maintained across page refreshes

9. Response Caching

1. User navigates to Wards page
2. Route: WardsController → Index()
3. [ResponseCache] attribute applied (60 seconds duration)
4. First request:
   - Database queried
   - Data fetched and rendered
   - Response cached in memory
5. Subsequent requests within 60 seconds:
   - Cached response served
   - Database not queried
   - Faster page load
6. After 60 seconds, cache expires and refreshes

8. ER DIAGRAM


Water Complaint Management System – ER Diagram
Country
CountryID : int «PK»
CountryName : string
Capital : string
contains
Ward
WardID : int «PK»
WardName : string
Area : string
Description : string
employs
has
Citizen
CitizenID : int «PK»
Name : string
Email : string
Phone : string
Address : string
Worker
WorkerID : int «PK»
Name : string
Email : string
Phone : string
AssignedWardID : int «FK»
sends
files
assigned to
ContactMessage
MessageID : int «PK»
Name : string
Email : string
Subject : string
Message : string
DateSent : datetime
Complaint
ComplaintID : int «PK»
Title : string
Description : string
Status : string
Priority : string
CreatedDate : datetime
Photo : string
CitizenID : int «FK»
WardID : int «FK»
WorkerID : int «FK»

9. TESTING SUMMARY

Functionality	Test Case	Expected Result	Actual Result	Status
Home Page	Access http://localhost:5000	Time-based greeting displays with image	Greeting and image display correctly based on time	☑ PASS
Practical 1	Application runs on ports 5000/5001	Both HTTP and HTTPS accessible	Successfully accessible on both ports	☑ PASS
Practical 2	Navigate to /Home/MyView	College information displayed	Custom view shows college details	☑ PASS
Practical 3	Visit home at different times	Greeting changes (Morning/Afternoon/Evening)	Dynamic greeting works correctly	☑ PASS
Practical 4	Click navigation links	Navigate between Index and Welcome	Navigation works seamlessly	☑ PASS
Practical 5	Visit /Home/Countries	Bootstrap-styled table displays	Table renders with proper Bootstrap classes	☑ PASS
Practical 6	Create Ward without description	Ward saves with nullable field	Nullable description field works	☑ PASS
Practical 7	View complaints list	Status/Priority badges colored	Extension methods apply correct CSS classes	☑ PASS
Practical 8	Perform database operations	All operations async	No blocking, responsive UI	☑ PASS
Practical 9	Apply filters and refresh	Filters persist via session	Session stores and retrieves filter values	☑ PASS
Practical 9	View complaint details	Cookie set for last viewed	Cookie created with 7-day expiry	☑ PASS

Practical 10	Access Wards page multiple times	Response cached for 60 seconds	Second request served from cache (faster)	☑ PASS
Practical 11	Run migrations and seed data	Database created with 5 sample records each	WaterComplaint.db created successfully	☑ PASS
Practical 12	GET /api/api/complaints	JSON array of complaints returned	Valid JSON response with status 200	☑ PASS
Practical 12	POST /api/api/complaints	New complaint created via API	Complaint saved, returns 201 Created	☑ PASS
Practical 13	Check ViewData/ViewBag	Data passed to view correctly	Values display on page	☑ PASS
Practical 13	TempData message	Success message shows once	Message displays, disappears on refresh	☑ PASS
Practical 14	Submit empty complaint form	Validation errors appear	Required field errors displayed	☑ PASS
Practical 14	Submit invalid email	Email validation triggers	Error message for invalid format	☑ PASS
Practical 15	Create new complaint	Complaint appears in list	Successfully created and listed	☑ PASS
Practical 15	View complaint details	All data displayed correctly	Details page shows complete info	☑ PASS
Practical 15	Edit complaint	Changes saved to database	Updated data persists	☑ PASS
Practical 15	Delete complaint	Complaint removed from list	Successfully deleted	☑ PASS
Filter	Filter by status "Pending"	Only pending complaints show	Correct filtering applied	☑ PASS
Filter	Filter by priority "High"	Only high priority complaints show	Correct filtering applied	☑ PASS
Filter	Filter by ward	Only selected ward complaints show	Correct filtering applied	☑ PASS

File Upload	Upload complaint photo	Photo saved and linked to complaint	File stored in uploads folder	☑ PASS
File Upload	View photos on details page	Uploaded photos display	Photo gallery renders correctly	☑ PASS
Ward CRUD	Create new ward	Ward added to database	Successfully created	☑ PASS
Ward CRUD	Edit ward details	Changes saved	Updated successfully	☑ PASS
Ward CRUD	Delete ward	Ward removed	Successfully deleted	☑ PASS
Responsive Design	Access on mobile device	UI adapts to screen size	Bootstrap responsive classes work	☑ PASS
HTTPS	Access https://localhost:5001	Secure connection established	Certificate warning resolved after trust	☑ PASS
Database Relationships	Delete complaint with photos	Photos cascade deleted	Referential integrity maintained	☑ PASS

Test Summary Statistics

- **Total Tests:** 35
- **Passed:** 35
- **Failed:** 0
- **Success Rate:** 100%

10. CONCLUSION

The **Water Complaint Management System** has been successfully developed and tested, achieving all project objectives and requirements. This comprehensive web application demonstrates proficiency in modern web development technologies, particularly ASP.NET Core MVC 8.0, Entity Framework Core, and responsive frontend design.

Key Achievements

1. **Complete Practical Integration:** All 15 practical exercises from the Web Technology with .NET course have been seamlessly integrated into a single, cohesive application, demonstrating a holistic understanding of the framework.

2. **Real-World Application:** The system addresses an actual community problem—efficient management of municipal water supply complaints—making it a practical solution that can be deployed for real-world use.

3. **Technical Excellence:** The project showcases:

    - Clean MVC architecture with proper separation of concerns
    - Normalized database design with appropriate relationships
    - RESTful API implementation for external integrations
    - Responsive UI with Bootstrap 5
    - Async operations for optimal performance
    - Comprehensive input validation and security measures

4. **User-Centric Design:** The application provides an intuitive interface for both citizens and municipal workers, with features like filtering, photo uploads, status tracking, and real-time updates.

5. **Educational Value:** The project serves as an excellent portfolio piece demonstrating full-stack web development capabilities, from database design to frontend implementation.

Project Impact

The Water Complaint Management System bridges the gap between citizens and municipal authorities by:

- Reducing complaint registration time from hours to minutes
- Providing transparency through real-time status tracking
- Enabling priority-based complaint handling

- Facilitating accountability through worker assignments
- Supporting data-driven decision making for municipal planning

Final Remarks

This project not only fulfills academic requirements but also demonstrates the practical application of theoretical concepts in solving real-world problems. The system is production-ready with appropriate enhancements for authentication, authorization, and deployment to cloud platforms. It serves as a solid foundation for future development and can be extended with additional features as outlined in the Future Scope section.

11. FUTURE SCOPE

1. Authentication & Authorization System

- **User Registration & Login:** Implement ASP.NET Core Identity for secure user authentication
- **Role-Based Access Control (RBAC):** Define roles (Citizen, Worker, Admin) with specific permissions
- **Profile Management:** Allow users to update personal information, change passwords, and manage preferences
- **Password Recovery:** Email-based password reset functionality
- **Two-Factor Authentication (2FA):** Enhanced security for sensitive operations

2. Notification & Alert System

- **Email Notifications:** Automated emails for complaint status updates, assignments, and resolutions
- **SMS Alerts:** Critical complaint notifications via SMS gateway integration
- **Push Notifications:** Real-time notifications through web push API
- **In-App Notifications:** Notification center within the application

- **Customizable Preferences:** Users can choose notification types and frequency

3. Advanced Reporting & Analytics

- **Admin Dashboard:** Comprehensive dashboard with charts and statistics
- **Ward-wise Analysis:** Complaint trends by ward with geographical visualization
- **Worker Performance Metrics:** Track resolution times, workload distribution, and efficiency
- **Export Functionality:** Generate PDF/Excel reports for official documentation
- **Predictive Analytics:** Use historical data to predict complaint patterns and resource needs
- **Time-series Analysis:** Identify peak complaint periods and seasonal trends

4. Mobile Application Development

- **Flutter Mobile App:** Native iOS and Android application using the existing API
- **Offline Mode:** Draft complaints offline and sync when connected
- **Camera Integration:** Direct photo capture from device camera
- **GPS Integration:** Auto-detect location for accurate complaint placement
- **QR Code Scanning:** Quick access to complaint status via QR codes

5. Geographic Information System (GIS) Integration

- **Interactive Map:** Visualize complaints on Google Maps or OpenStreetMap
- **Heat Maps:** Identify problem areas with high complaint density
- **Route Optimization:** Suggest optimal routes for workers to address multiple complaints
- **Area-based Filtering:** Select map region to view complaints in that area

6. Workflow & Automation

- **Automatic Worker Assignment:** AI-based assignment considering workload, location, and expertise

- **Complaint Escalation:** Auto-escalate overdue complaints to supervisors
- **Scheduled Reports:** Automatically generate and email weekly/monthly reports
- **Smart Priority Assignment:** ML model to automatically assign priority based on complaint description
- **Complaint Categories:** Add categorization (leakage, pressure, quality, timing) for better organization

7. Multi-Language Support

- **Internationalization (i18n):** Support for multiple languages (English, Hindi, Gujarati)
- **Language Switching:** User preference-based language selection
- **Localized Content:** Database-driven translations for dynamic content

8. Enhanced Security & Compliance

- **Data Encryption:** Encrypt sensitive data in database
- **Audit Logging:** Track all system changes for accountability
- **GDPR Compliance:** Implement data privacy regulations
- **Rate Limiting:** Prevent API abuse with request throttling
- **CAPTCHA:** Prevent bot submissions

9. Citizen Engagement Features

- **Rating System:** Citizens rate complaint resolution quality
- **Feedback Forms:** Collect suggestions for system improvement
- **Community Forum:** Discussion board for water-related issues
- **Complaint History:** Citizens view their past complaints
- **Follow Complaints:** Citizens can follow similar complaints in their area

10. Integration Capabilities

- **Third-Party APIs:** Integrate with payment gateways for water bill payments
- **IoT Sensors:** Connect with water quality sensors for automatic alerts
- **Social Media Integration:** Share updates on social platforms
- **Government Portal Integration:** Sync with state/national municipal portals

12. SYSTEM REQUIREMENTS

Software Requirements

Component	Specification	Purpose
Operating System	Windows 10/11, macOS 10.15+, Linux (Ubuntu 20.04+)	Development and hosting platform
.NET SDK	Version 8.0 or higher	Runtime environment for application
Visual Studio Code	Latest version with C# Dev Kit	Integrated development environment
Web Browser	Chrome 90+, Firefox 88+, Edge 90+, Safari 14+	Application access
SQLite	Version 3.x (included with EF Core)	Database management system
Git	Version 2.x or higher	Version control
Entity Framework Tools	dotnet-ef CLI	Database migrations

Hardware Requirements

Minimum Requirements

Component	Specification
Processor	Intel Core i3 / AMD Ryzen 3 (or equivalent)
RAM	4 GB
Storage	1 GB free disk space
Network	Broadband internet connection (for package download)
Display	1366 x 768 resolution

Recommended Requirements

Component	Specification
Processor	Intel Core i5 / AMD Ryzen 5 (or equivalent)
RAM	8 GB or higher
Storage	5 GB free disk space (SSD preferred)
Network	High-speed internet connection
Display	1920 x 1080 resolution or higher

Development Environment Setup

Prerequisites Installation Order:

1. Install .NET 8.0 SDK
2. Install Visual Studio Code
3. Install C# Dev Kit extension
4. Install Git
5. Install Entity Framework tools: `dotnet tool install --global dotnet-ef`

Deployment Requirements (Production)

Component	Specification
Web Server	IIS 10+, Nginx 1.18+, or Apache 2.4+
Database	SQL Server 2019+, PostgreSQL 12+, or MySQL 8.0+ (for production)
SSL Certificate	Valid SSL certificate for HTTPS
RAM	Minimum 2 GB for small-scale deployment
Storage	Minimum 10 GB (includes logs and uploads)

13. CHALLENGES & LEARNINGS

Challenges Faced

1. Database Migration and Seeding

Initially, understanding Entity Framework Core migrations and seed data configuration posed challenges. The relationship between models needed careful design to avoid circular references during JSON serialization.

Solution: Used `.Include()` method selectively for navigation properties and configured `OnDelete` behavior appropriately in `ApplicationDbContext` to maintain referential integrity.

2. File Upload Path Management

Uploaded files were initially not accessible via URL because they were stored outside the `wwwroot` folder.

Solution: Created `wwwroot/uploads/complaints/` directory and stored files with unique GUIDs to prevent filename conflicts while maintaining accessibility through static file middleware.

3. Session State Persistence

Sessions were not persisting between requests initially, causing filter preferences to reset.

Solution: Ensured middleware was configured in correct order in `Program.cs` and explicitly called `HttpContext.Session.SetString()` and `GetString()` methods properly.

4. Response Caching Implementation

Understanding the difference between response caching, memory caching, and distributed caching required additional research.

Solution: Implemented `[ResponseCache]` attribute on Wards controller with appropriate duration and location parameters, and added caching middleware to the pipeline.

5. Time Zone Handling for Dynamic Greeting

Initial implementation used server time which might differ from user's local time.

Learning: Decided to use server time (`DateTime.Now`) as acceptable for this academic project, but noted that production systems should consider user time zones.

Key Learnings

Technical Skills:

- Mastered ASP.NET Core MVC architecture and its implementation patterns
- Gained proficiency in Entity Framework Core for database operations
- Learned to design RESTful APIs following industry best practices
- Understood middleware pipeline configuration and execution order
- Developed skills in responsive web design with Bootstrap 5

Software Engineering Principles:

- **Separation of Concerns:** Proper division between Models, Views, and Controllers
- **DRY Principle:** Created extension methods to avoid code repetition
- **Database Normalization:** Designed efficient schema with appropriate relationships
- **Error Handling:** Implemented try-catch blocks and validation mechanisms
- **Code Documentation:** Added meaningful comments for maintainability

Problem-Solving Approach:

- Breaking down complex requirements into manageable modules

- Debugging techniques using breakpoints and logging
- Reading official documentation and community resources effectively
- Testing incrementally after each feature implementation

Project Management:

- Planning features and prioritizing based on course requirements
- Version control practices with Git for code management
- Documentation importance for future reference and collaboration

This project has been an enriching learning experience that bridges theoretical knowledge with practical implementation, preparing for real-world software development scenarios.