Project Title: Smart Feedback Portal

Scenario:

A company wants to build an internal portal where employees can submit feedback about various departments, services, or events. The system should allow users to log in, submit feedback, view past feedback, and for admins to analyze sentiment trends using Azure AI.

Key Features:

User Features:

User registration and login (via ASP.NET Identity)

Submit feedback with category and optional image

View personal feedback history

Sentiment analysis of submitted feedback

Admin Features:

Dashboard with feedback analytics

View feedback by category, sentiment, and date

Export feedback reports

Image moderation (optional via Azure AI)

Technology Mapping:

Frontend:

ReactJS: For a dynamic, responsive UI

Feedback form

Dashboard with charts (using Chart.js or similar)

Authentication pages

ASP.NET MVC: For admin panel or legacy-style views (optional hybrid)

Backend:

C# with.NET Core Web API:

RESTful APIs for feedback CRUD, user management, sentiment analysis

Authentication and authorization

Integration with Azure services

I

ORM:

Entity Framework Core:

Code-first approach

Models: User, Feedback. SentimentResult

Migrations and seeding

Database:

Azure SQL Database / SQLite:

Store user data, feedback, sentiment scores

Use stored procedures or views for reporting (optional)

Azure Cloud Integration:

Azure App Services: Host frontend and backend

Azure SQL: Main database

Azure Cognitive Services (Text Analytics API):

Analyze sentiment of feedback text

Azure Blob Storage (optional): Store uploaded images

Azure Functions (optional): Trigger sentiment analysis or image moderation

Azure Monitor / Application Insights: Logging and diagnostics

Practices:

Unit Testing: xUnit or NUnit for API logic (>=80% Code Coverage)

Debugging: Visual Studio and browser dev tools

CI/CD: GitHub Actions or Azure DevOps pipelines (optional)

I

Sample Workflow:

1. User logs in ReactJS frontend calls Web API.

2. User submits feedback → API stores it in Azure SQL

3. API triggers Azure Cognitive Services Sentiment score is returned and stored.

3. API triggers Azure Cognitive Services → Sentiment score is returned and stored.

4. Admin views dashboard → ReactJS dashboard fetches aggregated data from API.

5. Deployment Frontend and backend deployed to Azure App Services.

Here are the best practices for implementing and managing the Smart Feedback Portal project, tailored for freshers learning the Microsoft technology stack and Azure Cloud:

Project Planning & Architecture

1. Modular Design:

Separate concerns: UI. API, Business Logic, Data Access.

Use layered architecture (e.g.. Controller → Service → Repository).

2. Clean Code Principles:

Follow SOLID principles.

Use meaningful naming conventions and consistent formatting.

3. Agile Development (SCRUM):

Break the project into sprints (e.g.. Authentication, Feedback

Submission, Sentiment Analysis)..

Use tools like Azure Boards or GitHub Projects for task tracking.

Frontend (ReactJS + ASP.NET MVC)

1. Component-Based Architecture:

Reusable components (e.g.. Feedbackform. Feedbacklist. DashboardCard).

2. State Management:

Use React Context API or Redux for managing global state.

3. API Integration:

Use API with proper error handling and loading states.

4. Security:

Store tokens securely (e.g.. HttpOnly cookies or secure local storage).

Prevent XSS and CSRF attacks.

Backend (C#.NET Core Web API)

1. RESTful API Design:

Use proper HTTP verbs and status codes.

Version your APIs le.g/api/v1/feedback).

2. Dependency Injection:

Use built-in DI for services and repositories.

3. Validation:

Use Data Annotations and Fluent Validation for input validation.

4. Logging & Monitoring:

Integrate Serilog or Application Insights for logging.

Entity Framework Core (ORM) & Azure SQL

1. Code-First Approach:

Use migrations to manage schema changes.

Seed initial data for testing.

2. Performance:

Use As No Tracking() for read-only queries (Entity Framework).

Optimize queries with indexes and stored procedures (if needed).

3. Security:

Use Stored Procedures to prevent SQL injection.

Secure connection strings using Azure Key Vault or App Settings.

Azure Cloud Integration

1. Deployment:

Use Azure App Services for hosting frontend and backend.

Automate deployment with GitHub Actions or Azure DeyOps

Pipelines.

2. Azure Cognitive Services:

Use Text Analytics API for sentiment analysis.

Handle API rate limits and errors gracefully.

3. Azure Blob Storage (optional):

Store uploaded images securely.

Generate SAS tokens for secure access.

4. Monitoring:

Use Azure Monitor and Application Insights for diagnostics.

Testing & Debugging

1. Unit Testing:

Use xUnit or Nunit for backend logic (>=80% Code Coverage).

Mock dependencies using Mog

2. Frontend Testing:

Library for Use Jest and React Testing Library for component testing.

3. Debugging:

Use Visual Studio Debugger and Browser Dev Tools

Log meaningful errors and stack traces.

3. Debugging:

Use Visual Studio Debugger and Browser Dey Tools

Log meaningful errors and stack traces.

Security Best Practices

Use HTTPS for all communications.

Implement role-based access control (RBAC).

Sanitize all user inputs

Regularly update dependencies to patch vulnerabilities.

CI/CD & DevOps

Automate builds, tests, and deployments

Use Infrastructure as Code (e.g.. Bicep or ARM templates) for provisioning Azure resources.