Full Stack .NET Projects Proposals

Project 1: TechXpress - E-commerce Platform with ASP.NET Core

Objective:

Build a scalable and maintainable e-commerce web application for selling electronics, leveraging design patterns like NTier Architecture, Repository Pattern, and Unit of Work for efficient development and separation of concerns.

Description:

TechXpress is a full-featured e-commerce platform that allows users to browse electronics (laptops, mobiles, cameras), add them to a shopping cart, and complete purchases using integrated payment gateways. The platform includes an admin panel for managing products, categories, and orders. It will follow the NTier architecture to separate concerns (UI, Business Logic, and Data Access), improving maintainability and scalability.

Design Patterns to Use:

- NTier Architecture: Separation of the application into Presentation (UI), Business Logic (Service Layer), and Data Access (Repository/Entity Framework) layers.
- Repository Pattern: A layer that abstracts database interactions, providing a clean API for querying and persisting data.
- Unit of Work Pattern: Ensures that multiple related changes are grouped together in a single transaction, reducing the risk of data inconsistencies.
- Dependency Injection: Inject services (repositories, business logic) into controllers to promote testability and loose coupling.

Technologies to Use:

ASP.NET Core MVC, Entity Framework Core, ASP.NET Identity (for authentication), Stripe (for payment integration), JQuery, DataTables, Toaster JS, Microsoft Azure (for deployment).

Week 1: Initial Setup and Product Listings

NTier Architecture Setup:

- Presentation Layer (TechXpress.Web): The user-facing ASP.NET MVC application where views, controllers, and client-side logic (JQuery, DataTables) are implemented.
- Business Logic Layer (TechXpress.Services): Handles the business rules, operations like adding/removing products, calculating totals in the cart, and managing orders.
- Data Access Layer (TechXpress.Data): This layer uses the Repository and Unit of Work patterns to interact with the database, ensuring clean separation from business logic.

Database Design:

Use Entity Framework Core to define the schema for tables like Products, Categories, Orders, OrderDetails, and Users. The DbContext will serve as the Unit of Work, and repositories will abstract CRUD operations.

Repository Pattern Implementation:

Create generic repositories for common data operations (CRUD) and specific repositories for ProductRepository and CategoryRepository.

User Authentication Setup:

Set up ASP.NET Identity for user registration, login, and role management (customer and admin).

Deliverables:

- Full NTier Architecture setup with proper separation of Presentation, Business Logic, and Data Access.
- Basic product listing page integrated with search and filters.
- Database schema designed using Entity Framework Core, with repositories and the Unit of Work pattern implemented.
- User authentication system integrated using ASP.NET Identity.

Week 2: Shopping Cart, Role-Based Access Control, and Admin Panel

Shopping Cart Functionality:

Implement the shopping cart using Session State to store products added by users, allowing customers to review items before checkout. Utilize the Business Logic Layer for calculating cart totals and handling promotions or discounts.

Role-Based Access Control (RBAC):

Define roles (Customer, Admin) using ASP.NET Identity. Restrict access to the Admin Panel and product management features based on roles.

Admin Dashboard:

Create an admin interface using DataTables for managing products, categories, and orders. Use the Repository Pattern to abstract data access and ensure all product updates are handled through the Unit of Work pattern.

Deliverables:

- Fully functional shopping cart with session management.
- Role-based access control with permissions for admin and customers.
- Admin dashboard for managing product listings and orders, powered by repositories and unit of work.
- Thorough testing of shopping cart and role management.

Week 3: Order Placement, Stripe Payment Integration, and User Profiles

Order Placement:

Implement a checkout flow where customers can place orders. Use the Unit of Work to ensure the order transaction (saving the order and reducing stock) is atomic.

Stripe Payment Integration:

Integrate Stripe for handling payments. Use Stripe's API to securely process payments during checkout and store transaction details for reference.

User Profiles and Order History:

Create customer profile pages where users can view personal details, track their order history, and manage account settings. Ensure that profile updates and order views use the Repository Pattern for clean data access.

Deliverables:

- Order placement with integrated Stripe payments.
- Customer profile page with order history and account management features.
- Tested and functional order placement with Stripe payment gateway.

Week 4: Final Testing, UI Enhancements, and Deployment

UI Enhancements:

Refine the overall look and feel of the site using Bootstrap and custom CSS to create a modern, responsive design. Add JQuery for dynamic elements (e.g., live search, interactive product listings).

DataTables Integration for Admin Panel:

Use DataTables for product management in the admin dashboard, allowing features like sorting, filtering, and pagination to improve usability.

Final Testing and Deployment:

Perform comprehensive testing on the entire system, covering functionality like product browsing, cart management, order placement, and admin management. Deploy the application to Microsoft Azure or Hostinger for live use.

Deliverables:

- Fully responsive, polished UI for both front-end and admin panel.
- DataTables integrated into the admin dashboard for efficient product management.
- Live deployment of the TechXpress platform on Microsoft Azure/Hostinger.
- Completed project documentation, including architecture explanation (NTier, Repository, Unit of Work), setup, and user manual.

Project 2: MediCare - Clinic Management & Appointment System

Objective:

Build a comprehensive clinic management system that allows patients to book appointments with doctors, enables doctors to manage their schedules and patients, with an administrative dashboard for overall management.

Description:

MediCare is a full-featured healthcare platform that enables patients to search for doctors by specialization, book appointments, and access their medical history. Doctors can manage their schedules, write prescriptions, and track patient history. The system follows NTier architecture to ensure clear separation between application layers, improving maintainability and scalability.

Design Patterns to Use:

- NTier Architecture: Separation of the application into Presentation (UI), Business Logic (Service Layer), and Data Access (Repository/Entity Framework) layers.
- Repository Pattern: A layer that abstracts database interactions, providing a clean API for querying and persisting data.
- Unit of Work Pattern: Ensures that multiple related changes are grouped together in a single transaction, reducing the risk of data inconsistencies. ● Factory Pattern: For creating different types of appointments and notifications.
- Dependency Injection: Inject services (repositories, business logic) into controllers to promote testability and loose coupling.

Technologies to Use:

ASP.NET Core MVC, Entity Framework Core, ASP. NET Identity, SignalR (for real-time notifications), Twilio (for SMS), SendGrid (for email), FullCalendar. js, Chart.js, Bootstrap 5, Microsoft Azure.

Week 1: Initial Setup and User Management

NTier Architecture Setup:

- MediCare.Web (Presentation Layer): The user-facing ASP.NET MVC application where views, controllers, and client-side logic are implemented.
- MediCare.Services (Business Logic Layer): Handles business rules, operations like scheduling appointments, managing patient records, and prescription handling.
- MediCare.Data (Data Access Layer): Uses Repository and Unit of Work patterns to interact with the database, ensuring clean separation from business logic.

Database Design:

Use Entity Framework Core to define the schema for tables:

- `Doctors` (doctor information, specialization, consultation fee, profile image)
- `Patients` (patient information, medical history, allergies)
- `Appointments` (appointment details with status and notes)
- `Specializations` (medical specializations)
- `WorkingHours` (doctor availability schedules)
- `MedicalRecords` (patient medical records and history)

Repository Pattern Implementation:

Create generic repositories for common data operations (CRUD) and specific repositories:

- `IDoctorRepository`
- `IPatientRepository`
- `IAppointmentRepository`
- `IMedicalRecordRepository`

Multi-Role Authentication:

Set up ASP. NET Identity with three roles: Patient, Doctor, Admin. Implement registration flows specific to each role with appropriate validation.

Deliverables:

- Full NTier Architecture setup with proper separation of Presentation, Business Logic, and Data Access.
- Multi-role authentication and registration system.
- Database schema designed using Entity Framework Core, with repositories and Unit of Work pattern implemented.
- Doctor listing page with search and filter by specialization, location, and availability.

Week 2: Appointment System and Doctor Dashboard

Appointment Booking System:

- Interactive calendar using FullCalendar.js to display available time slots.
- Appointment booking with date and time selection based on doctor availability.
- Appointment cancellation and rescheduling functionality.
- Conflict detection to prevent double bookings.

Doctor Dashboard:

- View daily and weekly appointments in calendar and list views.

- Manage working hours and vacation days.
- Confirm or reject appointment requests.
- Quick patient information access.

Real-time Notifications:

Implement SignalR for instant notifications:

- Notify doctor when a new appointment is booked.
- Notify patient when appointment is confirmed or rejected.
- Send reminders before scheduled appointments.

Deliverables:

- Fully functional appointment booking system with interactive calendar.
- Doctor dashboard for managing appointments and schedules.
- Real-time notifications using SignalR.
- Comprehensive testing of booking system and conflict handling.

Week 3: Medical Records, Prescriptions, and External Notifications

Medical Records Management:

- Create medical records for each patient visit.
- Record diagnosis, symptoms, and treatment notes.
- Upload test results, X-rays, and medical documents.
- View complete patient medical history.

Prescription System:

- Create digital prescriptions with medication details.
- Medication list with dosage and frequency instructions.
- Print-friendly prescription format.
- Prescription history for patients.

External Notifications:

- Twilio integration for SMS appointment reminders.
- SendGrid integration for email notifications.
- Automated reminder scheduling (24 hours before appointment).
- Customizable notification preferences.

Patient Profile:

- View appointment history and upcoming appointments.
- Access medical records and prescriptions.
- Update personal information and medical history.
- Download medical documents.

Deliverables:

- Complete medical records management system.
- Digital prescription system with print capability.
- SMS and email notification integration.
- Patient profile page with comprehensive medical history.

Week 4: Admin Dashboard, Analytics, and Deployment

Admin Dashboard:

- Manage doctors (approve registrations, update profiles).
- Manage patients and their access.
- Manage specializations and clinic settings.
- View and manage all appointments.

Analytics & Reports:

Using Chart.js to display:

- Monthly appointment statistics.
- Most popular doctors and specializations.
- Patient demographics and trends.
- Revenue reports and financial summaries.
- Export reports to PDF/Excel format.

UI Enhancements:

- Responsive design using Bootstrap 5.
- Improved user experience with intuitive navigation.
- Dark/Light mode toggle.
- Accessibility improvements.

Deployment:

- Deploy application to Microsoft Azure App Service.
- Configure Azure SQL Database.
- Set up Azure SignalR Service for real-time features.
- Implement CI/CD pipeline using Azure DevOps or GitHub Actions.

Deliverables:

- Comprehensive admin dashboard with full management capabilities.
- Analytics and reporting with visual charts and export options.
- Polished, responsive UI with accessibility features.
- Live deployment on Azure with complete documentation.
- Project documentation including architecture explanation, setup guide, and user manual.

Project 3: EduHub - Online Learning Platform

Objective:

Build a comprehensive e-learning platform that enables instructors to create and publish courses, and allows students to enroll, learn, track progress, and earn certificates.

Description:

EduHub is a Udemy-like platform that allows instructors to upload video courses organized into sections and lessons. Students can enroll in courses, watch videos, complete quizzes, and earn certificates upon completion. The system includes a payment gateway for paid courses and an automated certificate generation system.

Design Patterns to Use:

- NTier Architecture: Clear separation between the three application layers.
- Repository Pattern: Abstracts database access operations.
- Unit of Work Pattern: Manages multiple transactions as a single unit.
- Strategy Pattern: For calculating different pricing and discount strategies.
- Observer Pattern: For tracking student progress and triggering notifications.
- Dependency Injection: For improved testability and maintainability.

Technologies to Use:

ASP.NET Core MVC, Entity Framework Core, ASP.NET Identity, Azure Blob Storage (for videos), Stripe (for payments), iTextSharp (for PDF certificates), Video.js, Progress. js, Bootstrap 5, Microsoft Azure.

Week 1: Initial Setup and Course Management

NTier Architecture Setup:

- EduHub.Web (Presentation Layer): User interface and interaction handling.
- EduHub. Services (Business Logic Layer): Business logic for course management, enrollments, and progress tracking.
- EduHub.Data (Data Access Layer): Data access and storage using repositories.

Database Design:

Design tables using Entity Framework Core:

- `Courses` (title, description, price, level, thumbnail image, status)
- `Sections` (course sections/modules)
- `Lessons` (individual lessons with video URL and duration)
- `Categories` (course categories and subcategories)
- `Instructors` (instructor profiles and earnings)
- `Students` (student profiles and preferences)
- `Enrollments` (student course enrollments)
- `Progress` (lesson completion tracking)
- `Reviews` (course ratings and reviews)

Repository Implementation:

- `ICourseRepository`
- `ILessonRepository`
- `IEnrollmentRepository`
- `IProgressRepository`
- `IReviewRepository`

Multi-Role System:

Three roles with specific permissions: Student, Instructor, Admin.

Deliverables:

- Complete NTier Architecture implementation.
- Database with Repository and Unit of Work patterns.
- Multi-role authentication system.
- Course browsing page with filtering, search, and category navigation.

Week 2: Instructor Dashboard and Content Upload

Instructor Dashboard:

- Create new courses with detailed descriptions and thumbnails.
- Organize content into sections and lessons.
- Set pricing, discounts, and course level.
- View enrollment statistics and revenue.

Video Upload System:

- Upload videos to Azure Blob Storage.
- Video processing and compression.
- Automatic thumbnail generation.
- Secure video streaming with signed URLs.

Course Builder:

- Drag-and-drop interface for organizing sections and lessons.
- Preview course before publishing.
- Save as draft or publish immediately.
- Bulk upload capabilities.

Course Management:

- Edit and delete courses and content.
- View detailed enrollment statistics.
- Respond to student questions in Q&A section.
- Manage course announcements.

Deliverables:

- Comprehensive instructor dashboard.
- Video upload system with Azure Blob Storage.
- Interactive course builder.
- Complete content management capabilities.

Week 3: Student Experience and Payments

Student Learning Experience:

- Custom video player using Video.js with playback controls.
- Automatic progress tracking (mark lessons as completed).
- Progress bar for each course showing completion percentage.
- Note-taking feature while watching videos.
- Bookmarking lessons for later review.

Enrollment & Payment:

- Shopping cart for multiple course purchases.
- Stripe integration for secure payment processing.
- Coupon codes and promotional discounts (Strategy Pattern).
- Support for both free and paid courses.
- Purchase history and receipts.

Quiz System:

- Create quizzes at the end of sections or courses.
- Multiple question types (multiple choice, true/false, fill-in-the-blank). ●
Instant result display with explanations.
- Minimum passing score requirements.

Reviews & Ratings:

- Course rating system (1-5 stars).
- Written review submission.
- Display average rating and review count.
- Helpful vote system for reviews.

Deliverables:

- Complete learning experience with progress tracking.
- Payment system integrated with Stripe.
- Quiz system with multiple question types.
- Review and rating system.

Week 4: Certificates, Admin Panel, and Deployment

Certificate Generation:

- PDF certificate generation using iTextSharp.
- Personalized certificates with student name, course title, and completion date.
- Unique verification code for each certificate.
- Public certificate verification page.
- LinkedIn sharing integration.

Admin Dashboard:

- User management (students and instructors).
- Course approval workflow for quality control.
- Category and tag management.
- Sales reports and revenue analytics.
- Platform settings and configuration.

Analytics:

Using Chart.js to visualize:

- Best-selling courses.
- Course completion rates.
- Instructor earnings.
- Student engagement metrics.
- Revenue trends over time.

Deployment:

- Deploy to Azure App Service.
- Configure Azure CDN for video delivery.
- Set up Azure Blob Storage for media files.
- Performance testing and optimization.
- Implement caching strategies.

Deliverables:

- Automated certificate generation system.
- Comprehensive admin dashboard.
- Advanced analytics and reporting.
- Full deployment on Azure with documentation.
- Complete project documentation including architecture, setup, and user guides.

Project 4: TaskFlow - Project & Task Management System

Objective:

Build a comprehensive project and task management system that enables teams to collaborate and track workflow using Kanban boards and Gantt charts.

Description:

TaskFlow is a Trello/Jira-like system that allows teams to create projects, break them down into tasks, and track work progress through interactive Kanban boards. The system supports real-time collaboration among team members with notifications and discussions.

Design Patterns to Use:

- NTier Architecture: Layer separation for easy maintenance.
- Repository Pattern: Abstract data operations.
- Unit of Work Pattern: Ensure data integrity across transactions.
- State Pattern: For managing different task states and transitions.
- Command Pattern: For undo/redo operations.
- Dependency Injection: For improved architecture and testing.

Technologies to Use:

ASP.NET Core MVC, Entity Framework Core, ASP.NET Identity, SignalR (for real-time updates), Dragula. js (for drag-and-drop), Frappe Gantt (for Gantt charts), Chart.js, Bootstrap 5, Microsoft Azure.

Week 1: Setup, Projects, and Team Management

NTier Architecture Setup:

- TaskFlow.Web (Presentation Layer): Interactive user interface.
- TaskFlow.Services (Business Logic Layer): Project and task management logic.
- TaskFlow.Data (Data Access Layer): Data layer with Repository pattern.

Database Design:

- `Workspaces` (organizational containers for projects)
- `Projects` (individual projects with settings)
- `Boards` (Kanban boards within projects)
- `Columns` (board columns: To Do, In Progress, Review, Done, etc.)
- `Tasks` (individual task items)
- `Subtasks` (checklist items within tasks)
- `Teams` (team definitions)
- `TeamMembers` (team membership with roles)
- `Comments` (task comments and discussions)
- `Attachments` (file attachments)
- `ActivityLogs` (audit trail of all actions)
- `Labels` (colored tags for categorization)

Team Management:

- Create workspaces and teams.
- Invite members via email with customizable permissions.
- Role hierarchy: Owner, Admin, Member, Viewer.
- Team member activity tracking.

Project Setup:

- Create projects within workspaces.
- Customize boards and columns.
- Pre-built project templates (Agile, Scrum, Basic).
- Project settings and access control.

Deliverables:

- Complete NTier Architecture implementation.
- Team and workspace management system.
- Project creation and customization.
- Multi-role authentication with granular permissions.

Week 2: Kanban Boards and Task Management

Kanban Board:

- Interactive board with drag-and-drop using Dragula.js.
- Move tasks between columns with visual feedback.
- Real-time updates for all team members using SignalR.
- Column limits (WIP limits) for workflow optimization.
- Swimlanes for task categorization.

Task Management:

Create tasks with comprehensive details:

- Title and rich text description.
- Due date and time estimates.
- Priority levels (Low, Medium, High, Critical).
- Colored labels for categorization.
- Assignee selection (single or multiple).
- Subtask checklists with progress tracking.
- Custom fields for additional data.

Task operations:

- Edit, delete, and archive tasks.
- Copy and move tasks between projects.
- Task templates for recurring work.

Real-time Collaboration:

- Instant updates when tasks are modified.
- "Currently viewing" indicators showing active users.
- Real-time notifications for task changes.
- Collaborative editing conflict resolution.

Comments & Attachments:

- Comment threads on tasks.
- File attachments (images, documents, etc.).
- @mention team members for notifications.
- Rich text formatting in comments.

Deliverables:

- Interactive Kanban board with drag-and-drop.
- Comprehensive task management system.
- Real-time collaboration using SignalR.
- Comments and attachments system.

Week 3: Tracking, Reporting, and Notifications

Gantt Chart View:

- Project visualization using Frappe Gantt.
- Define task dependencies (finish-to-start, etc.).
- Drag to adjust dates and durations.
- Critical path highlighting.
- Milestone markers.

Time Tracking:

- Log time spent on each task.
- Timer functionality for active work.
- Team productivity reports.
- Estimated vs. actual time comparison.
- Timesheet exports.

Dashboard & Analytics:

Using Chart.js for visualizations:

- Tasks by status (pie chart).
- Project progress (progress bars).
- Team member workload (bar chart).
- Overdue tasks highlighting.
- Velocity charts for sprints.
- Burndown/burnup charts.

Notification System:

- In-app notification center.
- Email notifications (customizable preferences).
- Due date reminders.
- Assignment notifications.
- Digest emails (daily/weekly summaries).

Activity Log:

- Complete audit trail of all actions.
- Track who changed what and when.
- Filter by user, task, or action type.
- Export activity logs.

Deliverables:

- Interactive Gantt chart view.
- Time tracking system.
- Analytics dashboard with visual charts.
- Comprehensive notification system.
- Activity logging and audit trail.

Week 4: Advanced Features and Deployment

Advanced Features:

- Global search across all tasks and projects.
- Advanced filtering (by status, priority, assignee, date, labels).
- Saved filters and custom views.
- Reusable task templates.
- Project archiving and restoration.
- Keyboard shortcuts for power users.

Automation Rules:

Configurable automation triggers:

- Auto-move task when all subtasks complete.
- Send notification when due date approaches.
- Auto-assign based on labels or columns.
- Custom webhook integrations.

Admin Panel:

- User and team management.
- Usage statistics and analytics.
- System settings and configuration.
- Subscription and billing management.
- Data export and backup options.

UI/UX Enhancements:

- Dark mode toggle.
- Responsive design for mobile devices.
- Keyboard shortcuts reference.
- Customizable dashboard widgets.
- Accessibility compliance (WCAG 2.1).

Deployment:

- Deploy to Azure App Service.
- Configure Azure SQL Database.
- Set up Azure SignalR Service.
- Implement Azure Application Insights for monitoring.
- Configure CI/CD with GitHub Actions.
- Load testing and performance optimization.

Deliverables:

- Advanced search, filtering, and templates.
- Automation rules engine.
- Comprehensive admin panel.
- Polished UI with dark mode and responsive design.
- Full deployment on Azure with monitoring.
- Complete documentation including architecture, API reference, setup guide, and user manual.

Project 5: SmartSupport AI - Agentic AI Customer Support Platform

Objective:

Build an intelligent customer support platform powered by Agentic AI that can autonomously handle customer inquiries, escalate complex issues to human agents,

and learn from interactions to improve over time.

Description:

SmartSupport AI is a next-generation customer support system that leverages Large Language Models (LLMs) and AI agents to provide 24/7 automated customer assistance. The platform can understand customer intent, access knowledge bases, perform actions (like checking order status, processing refunds), and seamlessly hand off to human agents when needed. It includes analytics to track AI performance and customer satisfaction.

Design Patterns to Use:

- NTier Architecture: Separation of Presentation, Business Logic, AI Services, and Data Access layers.
- Repository Pattern: Abstract database interactions for customers, conversations, and knowledge base.
- Unit of Work Pattern: Manage complex transactions involving multiple entities.
- Chain of Responsibility Pattern: For routing queries through different AI agents and handlers.
- Strategy Pattern: For selecting appropriate AI response strategies based on query type.
- Observer Pattern: For real-time updates and notifications.
- Dependency Injection: For flexible AI service integration and testing.

Technologies to Use:

ASP.NET Core MVC, Entity Framework Core, ASP.NET Identity, Azure OpenAI Service / OpenAI API, Semantic Kernel (. NET AI SDK), SignalR (for real-time chat), Azure Cognitive Search (for knowledge base), Redis (for caching), Azure Blob Storage, Chart.js, Bootstrap 5, Microsoft Azure.

Week 1: Foundation and AI Integration Setup

NTier Architecture Setup:

- SmartSupport. Web (Presentation Layer): Customer chat widget, agent dashboard, admin panel.
- SmartSupport.Services (Business Logic Layer): Conversation management, ticket handling, escalation logic. ● SmartSupport.AI (AI Services Layer): AI agent orchestration, LLM integration, intent recognition.
- SmartSupport.Data (Data Access Layer): Repository pattern for all data operations.

Database Design:

- `Customers` (customer profiles and preferences)
- `Conversations` (chat sessions with metadata)
- `Messages` (individual messages with sender type: AI/Customer/Agent)
- `Tickets` (support tickets for escalated issues)
- `KnowledgeArticles` (FAQ and help documentation)
- `AIResponses` (cached AI responses for common queries)
- `Feedback` (customer satisfaction ratings)
- `Agents` (human support agents)
- `ActionLogs` (AI actions and decisions audit trail)

AI Integration:

- Set up Azure OpenAI Service or OpenAI API integration.
- Implement Semantic Kernel for AI orchestration.
- Create base AI agent with conversation capabilities.
- Implement prompt engineering for customer support context.

Knowledge Base Setup:

- Create knowledge base structure for FAQs and documentation.
- Implement Azure Cognitive Search for semantic search.
- Enable AI to retrieve relevant articles during conversations.

Deliverables:

- Complete NTier Architecture with AI Services layer.
- Basic AI chat functionality with LLM integration.
- Knowledge base with semantic search capabilities.
- Database schema with repositories implemented.

Week 2: Agentic AI Capabilities and Customer Interface

AI Agent Functions (Tools):

Implement AI agent tools/functions:

- `CheckOrderStatus(orderId)` - Retrieve order information.
- `ProcessRefundRequest(orderId, reason)` - Initiate refund workflow.

- `UpdateCustomerInfo(customerId, updates)` - Modify customer details.
- `ScheduleCallback(customerId, preferredTime)` - Schedule human callback.
- `CreateSupportTicket(issue, priority)` - Escalate to human agent.
- `SearchKnowledgeBase(query)` - Find relevant help articles.

Intent Recognition:

- Implement intent classification for common support scenarios.
- Route conversations based on detected intent.
- Handle multi-turn conversations with context retention.

Customer Chat Widget:

- Embeddable chat widget using JavaScript/SignalR.
- Real-time message streaming (typing indicators).
- File attachment support for screenshots/documents.
- Chat history persistence.
- Mobile-responsive design.

Conversation Management:

- Session management with context window.
- Conversation summarization for long chats.
- Sentiment analysis during conversation.
- Automatic language detection and response.

Deliverables:

- AI agent with functional tools for common support tasks.
- Intent recognition and routing system.
- Customer-facing chat widget with real-time messaging.
- Context-aware multi-turn conversation handling.

Week 3: Human Agent Dashboard and Escalation

Escalation System:

- Automatic escalation triggers (sentiment, complexity, customer request).
- Smooth handoff from AI to human agent.
- AI-generated conversation summary for agent context.
- Priority queue management.

Human Agent Dashboard:

- Real-time queue of escalated conversations.
- Customer history and previous interactions view.
- AI-suggested responses for agents.
- Canned responses library.
- Internal notes and collaboration.

Hybrid AI-Human Mode:

- AI co-pilot assisting human agents.
- Real-time AI suggestions during human conversations.
- One-click AI response approval.
- Agent can request AI to draft responses.

Ticket Management:

- Create tickets from conversations.
- Ticket status workflow (Open, In Progress, Resolved, Closed).
- SLA tracking and alerts.
- Ticket assignment and routing.

Deliverables:

- Intelligent escalation system with smooth AI-to-human handoff. ● Comprehensive agent dashboard with AI assistance.
- Ticket management system with SLA tracking.
- Hybrid AI-human collaboration features.

Week 4: Analytics, Learning, and Deployment

AI Performance Analytics:

Using Chart.js to display:

- Resolution rate (AI vs. human).
- Average response time.
- Customer satisfaction scores (CSAT).
- Common query categories.
- Escalation rate and reasons.
- AI accuracy metrics.

Continuous Learning:

- Feedback loop from agent corrections.
- Knowledge base auto-update suggestions.
- Response quality rating system.
- A/B testing for AI responses.

Admin Panel:

- AI configuration and prompt management.
- Knowledge base content management.
- Agent performance metrics.
- System settings and integrations.
- API key management.

Security & Compliance:

- PII detection and handling.
- Conversation encryption.
- Data retention policies.
- Audit logging for all AI actions.

Deployment:

- Deploy to Azure App Service.
- Configure Azure OpenAI Service.
- Set up Azure Cognitive Search.
- Implement Redis caching for performance.
- Configure Azure Application Insights.

Deliverables:

- Comprehensive analytics dashboard.
- Continuous learning and improvement system.
- Admin panel for AI and system management.
- Security and compliance features.
- Full deployment on Azure with documentation.
- Complete project documentation including AI architecture, prompt engineering guide, and user manual.

Project 6: TechAssist AI - Agentic AI Technical Support System

Objective:

Build an intelligent technical support system powered by Agentic AI that can diagnose technical issues, guide users through troubleshooting steps, execute diagnostic commands, and resolve common IT problems autonomously.

Description:

TechAssist AI is an advanced technical support platform designed for IT helpdesk scenarios. It uses AI agents that can understand technical problems, access system documentation, run diagnostic checks, provide step-by-step troubleshooting guidance, and even execute automated fixes. The system learns from resolved tickets to improve future diagnostics and includes a comprehensive dashboard for IT administrators.

Design Patterns to Use:

- NTier Architecture: Separation of UI, Business Logic, AI Engine, and Data layers.
- Repository Pattern: Abstract data access for tickets, solutions, and configurations.
- Unit of Work Pattern: Manage complex diagnostic and resolution transactions.
- State Machine Pattern: For managing ticket and diagnostic workflow states.
- Chain of Responsibility Pattern: For hierarchical troubleshooting steps.
- Command Pattern: For executable diagnostic and fix commands.
- Decorator Pattern: For adding logging and validation to AI operations.
- Dependency Injection: For modular AI and diagnostic service integration.

Technologies to Use:

ASP.NET Core MVC, Entity Framework Core, ASP.NET Identity, Azure OpenAI Service / OpenAI API, Semantic Kernel, LangChain. NET, SignalR, PowerShell/Azure Automation (for remote diagnostics), Azure Cognitive Search, Azure Blob Storage, Redis, Mermaid. js (for flowcharts), Chart.js, Bootstrap 5, Microsoft Azure.

Week 1: Foundation and Technical Knowledge Base

NTier Architecture Setup:

- TechAssist. Web (Presentation Layer): User support portal, technician dashboard, admin console.
- TechAssist.Services (Business Logic Layer): Ticket management, workflow orchestration, escalation logic.
- TechAssist.AI (AI Engine Layer): Diagnostic AI agents, LLM integration, reasoning engine.
- TechAssist. Diagnostics (Diagnostic Services Layer): System checks, command execution, health monitoring.
- TechAssist.Data (Data Access Layer): Repository pattern implementation.

Database Design:

- `Users` (employees/customers with device information)
- `Devices` (registered devices and configurations)
- `Tickets` (technical support tickets)
- `DiagnosticSessions` (AI diagnostic sessions)
- `DiagnosticSteps` (individual troubleshooting steps taken)
- `Solutions` (known solutions and fixes)
- `TroubleshootingGuides` (step-by-step guides)
- `SystemLogs` (collected system logs for analysis)
- `KnowledgeBase` (technical documentation)
- `Technicians` (IT support staff)
- `ResolutionHistory` (successful resolution patterns)

Technical Knowledge Base:

- Import technical documentation and manuals.
- Create structured troubleshooting decision trees.
- Implement semantic search for technical queries.
- Version control for knowledge articles.

AI Diagnostic Agent Setup:

- Configure LLM with technical support persona.
- Implement Semantic Kernel plugins for diagnostics.
- Create diagnostic reasoning chain.
- Set up error pattern recognition.

Deliverables:

- Complete NTier Architecture with Diagnostics layer.
- Technical knowledge base with semantic search.
- Base AI diagnostic agent configuration.
- Database schema with all repositories.

Week 2: AI Diagnostic Capabilities and User Interface

AI Diagnostic Functions (Tools):

Implement AI agent diagnostic tools:

- `AnalyzeErrorMessage(errorText)` - Parse and identify error type.
- `CheckSystemStatus(deviceId)` - Retrieve device health status.
- `RunDiagnosticCommand(command, deviceId)` - Execute remote diagnostic.
- `GetEventLogs(deviceId, timeRange)` - Retrieve system event logs.
- `CheckNetworkConnectivity(deviceId)` - Test network status.
- `VerifySoftwareVersion(software, deviceId)` - Check installed versions.
- `SearchSolutions(problemDescription)` - Find known solutions.
- `GenerateTroubleshootingGuide(issue)` - Create step-by-step guide.
- `EscalateToTechnician(ticketId, reason)` - Escalate complex issues.

Intelligent Problem Analysis:

- Natural language problem description parsing.
- Error code recognition and lookup.
- Screenshot/log file analysis using AI vision.
- Pattern matching with historical issues.
- Root cause analysis suggestions.

User Support Portal:

- Submit technical support request.
- AI-guided problem description (structured intake).
- Real-time chat with AI diagnostic agent.
- View troubleshooting progress and steps.
- Access to knowledge base articles.
- Ticket history and status tracking.

Interactive Troubleshooting:

- Step-by-step guided troubleshooting.
- Visual flowcharts using Mermaid.js.
- User confirmation at each step.
- Rollback capability for failed steps.
- Progress saving for multi-session troubleshooting.

Deliverables:

- Comprehensive AI diagnostic toolkit.
- Intelligent problem analysis system.
- User support portal with AI chat.
- Interactive troubleshooting interface.

Week 3: Technician Dashboard and Automated Remediation

Automated Remediation:

Safe automated fixes for common issues:

- Clear temporary files and cache.
- Restart services.
- Reset network configuration.
- Apply registry fixes.
- Install missing updates.
- Permission fixes.

Safety controls:

- User consent before automated actions.
- Rollback snapshots before changes.
- Action logging and audit trail.
- Sandbox testing for scripts.

Technician Dashboard:

- Queue of escalated and complex tickets.
- AI-generated diagnostic summary.
- Device information and history.
- Remote diagnostic tools integration.
- Knowledge base search and article creation.
- Resolution documentation.

Collaborative Diagnosis:

- AI suggests diagnostic steps to technician.
- Technician can guide AI to try different approaches.
- Real-time collaboration on complex issues.
- AI learns from technician's solutions.

Solution Library:

- Catalog of verified solutions.
- Success rate tracking per solution.
- Environment-specific solution variants.
- Community-contributed solutions.
- AI-suggested solution improvements.

Deliverables:

- Automated remediation with safety controls.
- Comprehensive technician dashboard.
- Collaborative AI-technician diagnosis.
- Searchable solution library.

Week 4: Analytics, Learning System, and Deployment

Analytics Dashboard:

Using Chart.js to visualize:

- Ticket volume and trends.
- AI resolution rate vs. human resolution.
- Mean time to resolution (MTTR).
- Common issue categories.
- Device/software problem patterns.
- Technician performance metrics.
- Customer satisfaction scores.

Machine Learning Integration:

- Pattern recognition for recurring issues.
- Predictive issue detection.
- Solution recommendation ranking.
- Anomaly detection in system logs.

Continuous Improvement:

- Learning from successful resolutions.
- Knowledge base gap identification.
- Automated solution documentation.
- Feedback integration from technicians.
- A/B testing for diagnostic approaches.

Admin Console:

- AI configuration and tuning.
- Diagnostic command management.
- Knowledge base administration.
- User and device management.
- Integration settings (AD, SCCM, etc.).
- Security and access controls.

Integration Capabilities:

- Active Directory integration.
- SCCM/Intune integration.
- ServiceNow/Jira ticketing integration.
- Email notification system.
- Webhook support for external systems.

Deployment:

- Deploy to Azure App Service.
- Configure Azure OpenAI Service.
- Set up Azure Automation for remote commands.
- Implement Azure Key Vault for secrets.
- Configure Application Insights monitoring.
- Set up CI/CD pipeline.

Deliverables:

- Comprehensive analytics and reporting.
- Machine learning-enhanced diagnostics.
- Continuous learning and improvement system.
- Full admin console with all configurations.
- Integration with enterprise systems.
- Complete deployment on Azure.
- Documentation including AI training guide, diagnostic command reference, and administration manual.

Projects Summary

#	Project	Domain	Key Features	Distinctive Technologies
1	TechXpress	E-commerce	Product catalog, Shopping cart, Payments	Stripe, DataTables, JQuery
2	MediCare	Healthcare	Appointments, Medical records, Prescriptions	SignalR, Twilio, FullCalendar. js
3	EduHub	Education	Video courses, Quizzes, Certificates	Azure Blob, Video.js, iTextSharp

#	Project Name	Domain	Key Features	Technologies
5	SmartSupport AI	AI Customer Support	AI Chat, Intent Recognition, Escalation	Azure OpenAI, Semantic Kernel, Cognitive Search
6	TechAssist AI	AI Technical Support	AI Diagnostics, Auto-remediation, Knowledge Base	Azure OpenAI, Semantic Kernel, Azure Automation

---

Common Technical Requirements for All Projects

Architecture Requirements:

- Implement NTier Architecture with clear separation of concerns
- Use Repository Pattern for data access abstraction
- Implement Unit of Work for transaction management
- Apply Dependency Injection throughout the application

Code Quality:

- Follow SOLID principles
- Write clean, readable, and maintainable code
- Include XML documentation for public APIs
- Implement proper error handling and logging

Testing:

- Unit tests for business logic layer
- Integration tests for repository layer
- Basic UI testing for critical paths

Security:

- Implement ASP.NET Identity for authentication
- Role-based authorization
- Input validation and sanitization
- Protection against common vulnerabilities (XSS, CSRF, SQL Injection)

Documentation:

- Architecture documentation with diagrams
- API documentation
- Setup and installation guide
- User manual

These projects are designed to give students hands-on experience with real-world full-stack . NET development, covering frontend, backend, database design, cloud deployment, and cutting-edge AI integration.