# Backend Roadmap

## 4-Day .NET Todo API Development Roadmap

### Project Goal: Simple Todo Application REST API

Build a HTTP server implementing a RESTful API to perform basic CRUD operations for a primitive todo management system. The application will demonstrate fundamental concepts of API development, database integration, and modern .NET practices.

---

### Todo Application Specification

#### Core Features

Your todo application should support the following functionality:

- Create, read, update, and delete todo items
- Organize todos by categories
- Set priority levels (Low, Medium, High)
- Track completion status
- Set due dates for todos
- Basic user management

#### Database Design Requirements

**Users Table:**

- Store user information including username, email, and password
- Each user should have a unique identifier
- Track when users are created

**Categories Table:**

- Allow users to organize their todos into categories
- Each category belongs to a specific user
- Categories should have names and optional color coding

- Track creation timestamps

**Todos Table:**

- Store todo items with title and optional description
- Track completion status and completion timestamp
- Support priority levels (1=Low, 2=Medium, 3=High)
- Allow optional due dates
- Each todo belongs to a user and optionally to a category
- Track creation and last update timestamps

**Relationships:**

- Users can have multiple todos (one-to-many)
- Users can have multiple categories (one-to-many)
- Todos can belong to one category (many-to-one, optional)
- Todos belong to one user (many-to-one, required)

### API Endpoints Structure

Plan for these RESTful endpoints:

- `GET /api/todos` - List todos with optional filtering
- `POST /api/todos` - Create new todo
- `GET /api/todos/{id}` - Get specific todo
- `PUT /api/todos/{id}` - Update todo
- `DELETE /api/todos/{id}` - Delete todo
- `GET /api/categories` - List categories
- `POST /api/categories` - Create category

---

## Day 0: SQL Fundamentals

### Learning Objectives

By the end of this day, you should understand SQL basics and be comfortable with database design principles essential for the todo application.

### Topics to Master

**Database Design Concepts:**

- Understanding relational database principles

- Primary keys, foreign keys, and constraints
- Data types selection (VARCHAR, TEXT, INTEGER, BOOLEAN, TIMESTAMP)
- Normalization basics

**Table Creation:**

- CREATE TABLE syntax and best practices
- Defining columns with appropriate data types
- Setting up constraints (NOT NULL, UNIQUE, CHECK)
- Creating indexes for performance

**Relationships:**

- One-to-many relationships using foreign keys
- CASCADE and SET NULL delete behaviors
- Junction tables for many-to-many relationships

**CRUD Operations:**

- INSERT statements for adding data
- SELECT queries with filtering, sorting, and joins
- UPDATE statements for modifying records
- DELETE statements for removing data

**Advanced Queries:**

- JOIN operations (INNER, LEFT, RIGHT)
- Aggregation functions (COUNT, SUM, AVG, MAX, MIN)
- GROUP BY and HAVING clauses
- Subqueries and conditional logic

### Practical Exercises

- Design a simple database schema on paper
- Practice writing queries for different scenarios
- Experiment with sample data insertion and manipulation
- Try complex queries involving multiple tables

---

## Day 1: Database Infrastructure Setup

### Learning Objectives

Establish a working PostgreSQL database environment using Docker and implement the complete todo application schema.

### Docker and PostgreSQL Setup

**Docker Fundamentals:**

- Understanding containerization concepts
- Installing Docker and Docker Compose
- Working with container lifecycle (start, stop, restart)
- Managing volumes for data persistence

**PostgreSQL Container Setup:**

- Create a docker-compose.yml file for PostgreSQL
- Configure environment variables (database name, user, password)
- Set up port mapping for local access
- Configure data persistence with volumes
- Learn to connect to the database container

**Database Creation:**

- Connect to PostgreSQL using command-line tools or GUI clients
- Create the todo application database
- Implement your designed schema from Day 0
- Create all required tables with proper relationships
- Set up appropriate indexes for performance
- Insert sample data for testing

**Connection Management:**

- Learn connection string formats
- Test connections from different tools
- Understand security considerations for database access

### Key Deliverables

- Working PostgreSQL container
- Complete database schema implementation
- Sample data for testing
- Documented connection procedures

---

## Day 2: .NET Database Integration

### Learning Objectives

Build a robust data access layer that connects your .NET application to PostgreSQL and handles all database operations.

### Project Structure Setup

**Initial Project Creation:**

- Create a new .NET Web API project
- Install required NuGet packages for PostgreSQL integration
- Set up project structure with appropriate folders
- Configure connection strings in appsettings.json

**Database Models and Entities:**

- Create C# classes representing your database tables
- Implement proper data annotations for validation
- Set up navigation properties for relationships

**Database Context Configuration:**

- Set up Entity Framework DbContext
- Configure database connection
- Define DbSets for your entities
- Configure entity relationships using Fluent API
- Handle database migrations

**Repository Pattern Implementation:**

- Create interfaces for data access operations
- Implement repository classes for each entity
- Focus on CRUD operations (Create, Read, Update, Delete)
- Handle async operations properly
- Implement proper error handling and logging

**Dependency Injection:**

- Register repositories in the service container
- Understand scoped vs transient vs singleton lifetimes
- Configure services in Program.cs

### Core Operations to Implement

- Retrieve all todos for a user
- Get a specific todo by ID

- Create new todos
- Update existing todos
- Delete todos
- Filter todos by completion status, priority, or category
- Handle category operations

---

## Day 3: HTTP API Implementation

### Learning Objectives

Create RESTful controllers that expose your data operations through HTTP endpoints, handle requests/responses, and implement proper API patterns.

### Controller Development

**RESTful API Design:**

- Understand REST principles and HTTP methods
- Design clean, intuitive API endpoints
- Implement proper HTTP status codes
- Handle different content types (JSON)

**Request/Response Handling:**

- Create Data Transfer Objects (DTOs) for API contracts
- Implement proper request validation
- Handle query parameters and route parameters
- Parse JSON request bodies
- Structure consistent API responses

**Controller Implementation:**

- Create API controllers inheriting from ControllerBase
- Implement dependency injection in controllers
- Handle different HTTP methods (GET, POST, PUT, DELETE)
- Implement proper error handling and logging
- Return appropriate HTTP status codes

**Validation and Error Handling:**

- Implement model validation using data annotations
- Handle validation errors gracefully

- Create consistent error response formats
- Log errors appropriately
- Handle edge cases (not found, conflicts, etc.)

**API Documentation:**

- Set up Swagger/OpenAPI documentation
- Document endpoint parameters and responses
- Test endpoints using Swagger UI

### Testing Your API

**Manual Testing:**

- Use Postman or similar tools to test endpoints
- Test all CRUD operations
- Verify proper status codes and responses
- Test error scenarios
- Validate request/response data

**Integration Testing:**

- Set up basic integration tests
- Test complete request/response cycles
- Verify database interactions work correctly

---

## Additional Requirements & Best Practices

### Git Workflow

- Initialize Git repository from Day 0
- Make meaningful commits at the end of each day
- Use descriptive commit messages
- Create branches for different features if desired
- Document your progress in README.md

### Data Transfer Objects (DTOs)

- Create separate DTOs for requests and responses
- Never expose internal entity models directly to clients
- Include only necessary data in API responses
- Validate input data using DTOs

- Consider different DTOs for create vs update operations

### Testing Strategy

- Write unit tests for repository methods
- Create integration tests for API endpoints
- Test both success and failure scenarios
- Use in-memory databases for testing when appropriate
- Aim for meaningful test coverage, not just high percentages

### Code Quality

- Follow consistent naming conventions
- Implement proper logging throughout the application
- Handle exceptions gracefully
- Use async/await patterns correctly
- Comment complex business logic
- Keep methods focused and single-purpose

---

### Bonus Challenges

- Implement filtering and sorting for todo lists
- Add pagination for large todo collections
- Create endpoint for todo statistics (counts by status, priority)
- Implement soft delete for todos
- Add data seeding for development environment
- Set up different configurations for development/production environments