

Patient Management System – README

Author – Pratik Wasnik

A mini Patient Management System built with .NET 6 Web API and SQL Server. The system supports CRUD operations, reporting, and optimized querying with indexes and stored procedures.

Table of Contents

1. - Features
2. - GitHub Content
3. - Tech Stack
4. - Project Structure
5. - Database Schema
6. - Database Optimization
7. - Advanced Requirement Implemented
8. - Getting Started
9. - API Endpoints
10. - Error Handling
11. - Exception Handling
12. - Future Enhancements
13. - Sitecore Task

Features

- - Manage Patients (Create, Read, Update, Delete)
- - Manage Conditions and Patient Conditions
- - Reporting: Recent patients (last 6 months), Top 3 cities with maximum patients, Patients with more than 2 conditions, Average age by condition
- - Dynamic search with filters (age range, condition, city) using stored procedures
- - Optimized querying with clustered and non-clustered indexes
- - JWT Authentication & Authorization
- - Swagger UI for API testing
- - Centralized Exception Handling Middleware

Github Content

Link : <https://github.com/pratikwasnik/patient-management-system>

- Project Repo with Solution File
- SQL Query Files
- DB Backup
- README FILE

Tech Stack

- - .NET 6/8 Web API
- - SQL Server
- - Dapper (Micro ORM)
- - FluentValidation (DTO validation)
- - Swagger / Swashbuckle (API documentation)
- - xUnit + Moq (Unit Testing)

Project Structure

PatientManagement.API/

- ├ Controllers/ # API endpoints
- ├ DTOs/ # Data Transfer Objects
- ├ Middleware/ # Global exception handling
- ├ Models/ # Entity models
- ├ Repositories/ # Data access using Dapper
- ├ Services/ # Business logic
- ├ Validators/ # FluentValidation rules
- ├ Program.cs # Entry point
- ├ appsettings.json # Config (DB, JWT, etc.)

PatientManagement.Tests/

- ├ Services/ # Unit tests for services







































Database Schema

- Patients (Id, FirstName, LastName, DOB, Gender, City, Email, Phone)

- Conditions (Id, Name, Description)

- PatientConditions (PatientId, ConditionId, DiagnosedDate)

Note : Id will be autoincremented

- [-]  PatientManagementSystem
 - [+]  Database Diagrams
 - [-]  Tables
 - [+]  System Tables
 - [+]  FileTables
 - [+]  External Tables
 - [+]  Graph Tables
 - [-]  dbo.Conditions
 - [-]  Columns
 -  Id (PK, int, not null)
 -  Name (varchar(100), not null)
 -  Description (varchar(1000), null)
 - [+]  Keys
 - [+]  Constraints
 - [+]  Triggers
 - [+]  Indexes
 - [+]  Statistics
 - [-]  dbo.PatientConditions
 - [-]  Columns
 -  PatientId (PK, FK, int, not null)
 -  ConditionId (PK, FK, int, not null)
 -  DiagnosedDate (date, not null)
 - [+]  Keys
 - [+]  Constraints
 - [+]  Triggers
 - [+]  Indexes
 - [+]  Statistics
 - [-]  dbo.Patients
 - [-]  Columns
 -  Id (PK, int, not null)
 -  FirstName (varchar(100), not null)
 -  LastName (varchar(100), not null)
 -  DOB (date, not null)
 -  Gender (char(1), null)
 -  City (varchar(100), null)
 -  Email (varchar(100), not null)
 -  Phone (varchar(15), null)
 - [+]  Keys

Database Optimization

For Faster Search Non Clustered Index have been created for “Name” Column in Conditions Table , “City” column in Patients table and “Email” column Patients column.

Tables Patients and Conditions have Id as primary key , hence by default a clustered index is created.

The SQL query is provided in github and in code using dapper it is used.

Stored Procedure is also created with exception handling.

Advanced Requirement Implemented

- **Dependency injection.**
- **Repository + Service pattern.**
- **Global Exception Handling** middleware.
- **Fluent Validation**
- **Region For Code-Readability**
- **Dapper for Better performance and control over SQL Queries**
- **Unit Tests** (xUnit + Moq).
- **API Security with JWT Authentication.**

Getting Started

Prerequisites

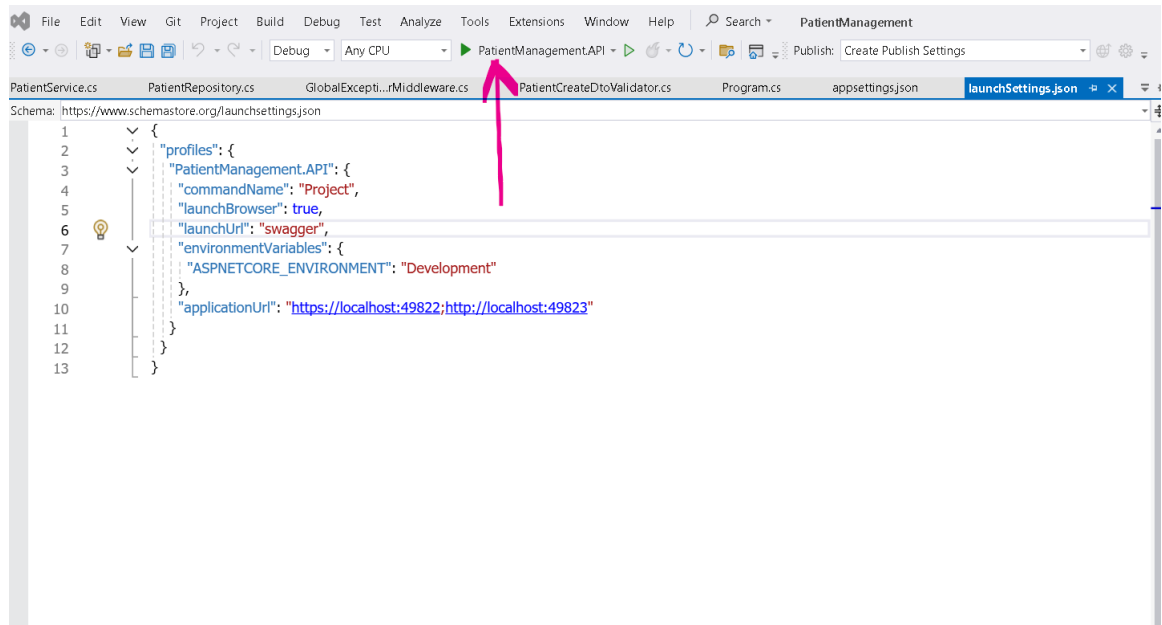
- - .NET 6 SDK
- - SQL Server (local or remote)
- - Swagger for testing

Setup Instructions

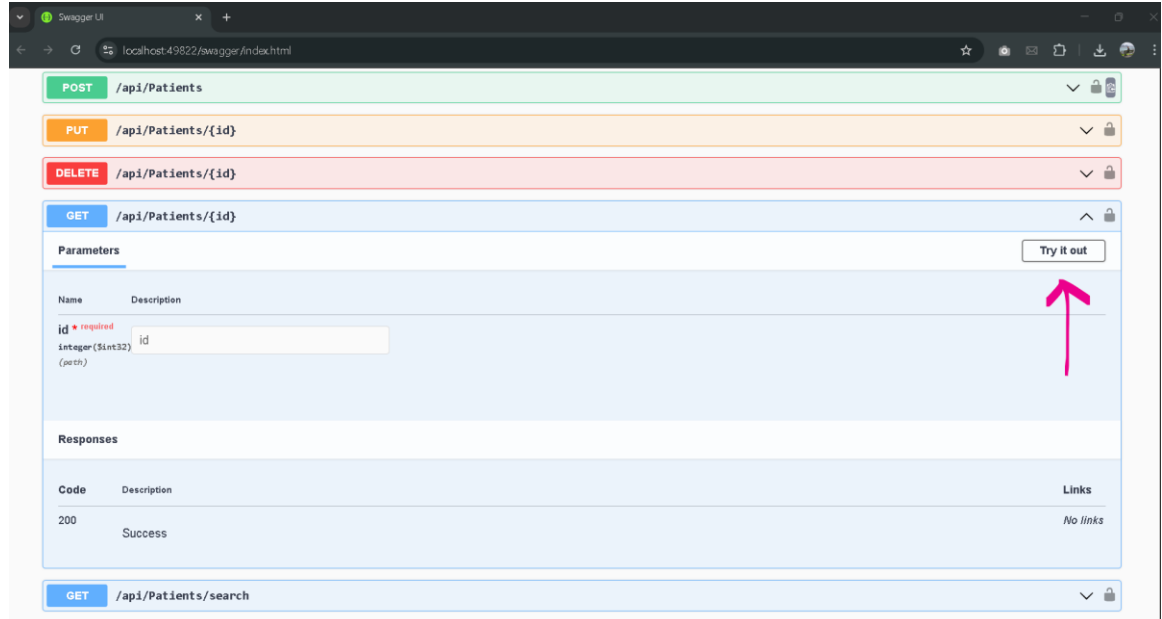
1. - Clone the repository: git clone <https://github.com/pratikwasnik/patient-management-system.git/>
2. - Create the database from Backup files provided
3. - Open the solution file in Visual Studio 2022
4. - Update appsettings.json with your SQL Server connection string and JWT settings
5. - In Visual Studio 2022 run the project
6. - Swagger is configured to run automatically once project is run
7. - In Swagger select any of the different endpoint and test the API

8. - E.G. to get patient by id follow the steps below :

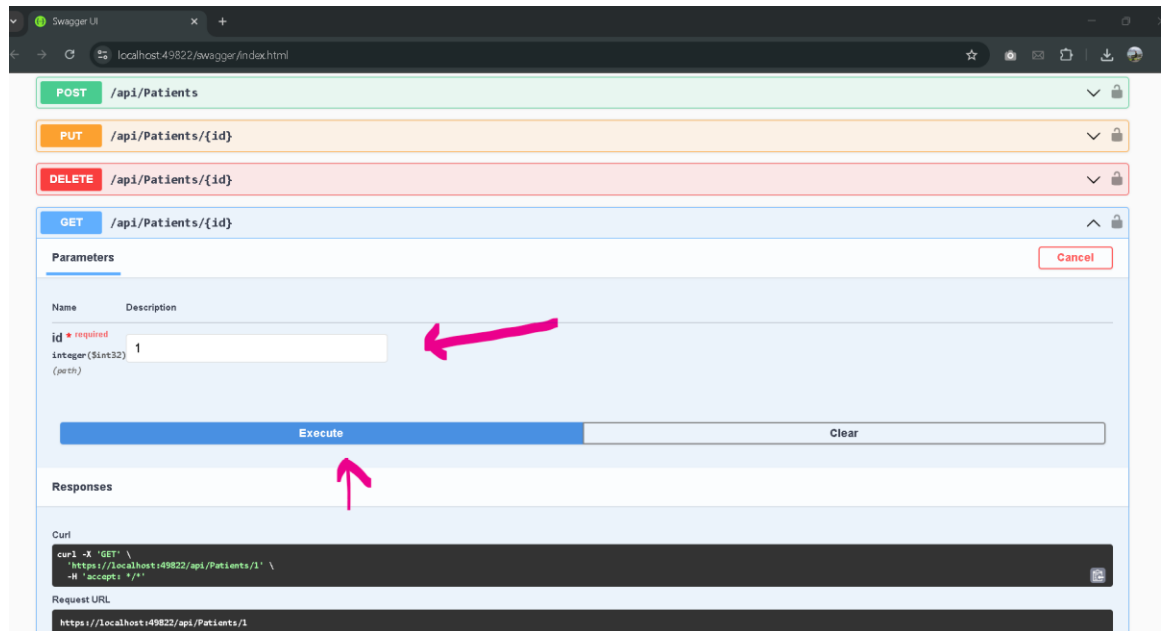
a) Click on the Dark green button in Visual Studio -



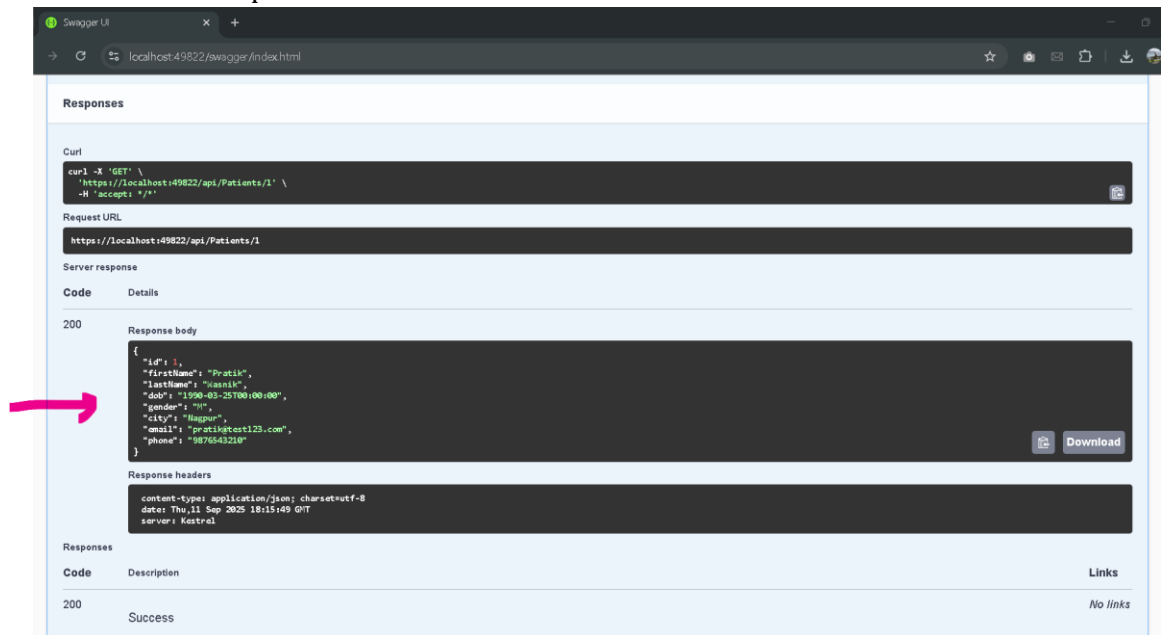
b) Once the Swagger page is automatically opened , select any of the API endpoint and Click on Try Now



c) E.g. to get Patient by Id , enter 1 in Id textbox and click on Execute



d) You will see the response as below :



9. Note : when creating a Patient by API , you can use following json body , you do not need to give Id as it is auto incremented in SQL

```
{
  "firstName": "Pratik",
  "lastName": "Wasnik",
  "dob": "1990-03-25T00:00:00",
  "gender": "M",
  "city": "Nagpur",
  "email": "pratik@test123.com",
  "phone": "9876543210"
}
```

API Endpoints

Patients

- - POST /api/patients → Add new patient
- - GET /api/patients/GetById/{id} → Get patient by Id
- - PUT /api/patients/UpdateById/{id} → Update patient
- - DELETE /api/patients/DeleteById/{id} → Delete patient
- - GET /api/patients/search → Search with filters

Reports

- - GET /api/patients/recent
- - GET /api/patients/top-cities
- - GET /api/patients/multi-condition
- - GET /api/patients/avg-age-by-condition

Error Handling

- - 400 Bad Request → Invalid input (e.g., wrong email format)
- - 404 Not Found → Patient/Condition not found
- - 409 Conflict → Duplicate email
- - 500 Internal Server Error → Unhandled exceptions

Exception Handling

- Future DOB is not allowed



- Only Patient with Unique Email can be added



Sitecore Task

- Note : Since I don't have Sitecore license , following are my suggestions for the Patients Management Project.

Information Architecture :

Depending on whether project ecosystem has Headless or SXA implementation , the Page and Components should be designed and created.

IF SXA is implemented, check whether existing component can be used or not.

If SXA is not implemented or existing SXA components are not used then creation of custom Templates and Components can be done.

📄 **Page type:** *Patient Management Page*

- **Components:**
 - Static content area (editor-managed, rich text, banners, descriptions).
 - Patient management component (connects to backend API for CRUD).
 - In SXA , Theme can be created right from Site definition item and including required Themes from Settings Items.
For Non SXA Project, Theme selector (for applying different UI themes).
- **Create Templates**
 1. **Patient Management Page Template**
 - Fields:
 - PageTitle (Single-Line Text)
 - PageDescription (Rich Text)
 - Theme (Droplist → As per Business Requirement)
 2. **Patient Component Template** (for API-based rendering)
 - Fields:
 - ApiEndpoint (Single-Line Text, default = /api/patients/)
 - AllowAdminOperations (Checkbox)
- **Create Renderings**
 1. **Static Content Rendering**
 - Datasource: Patient Management Page Template
 - Displays title, description, editor-managed content.
 - Supports *dynamic placeholders* for flexibility.
 2. **Patient CRUD Rendering**
 - Datasource: Patient Component Template.
 - Renders UI for **View / Add / Edit / Delete Patients**.
 - Uses AJAX/Fetch to call PatientController API (/api/patients/...).
 - Example:
 - GET → list patients
 - POST → add patient
 - PUT → update patient
 - DELETE → remove patient
- **. Dynamic Placeholder Setup**
 - In Patient Page layout:
 - Define a **dynamic placeholder key** (e.g., patient-content-{GUID}).
 - This allows content authors to add multiple renderings (e.g., banners, lists, forms) into the page without being locked to a fixed placeholder.
- ---
- **5. Support Multiple Themes**
 - Add a Theme field at page level (Droplist).
 - In your rendering code:
 - Apply a CSS class or load a theme-specific stylesheet dynamically.

- Example: if Theme = Dark, rendering wraps output with `<div class="theme-dark">...</div>`.

-

- **6. Datasource Restrictions**

- At rendering level:
 - Configure datasource templates.
 - Example: **Patient CRUD Rendering** should only allow Patient Component Template as its datasource.
 - Prevents content authors from accidentally attaching wrong datasources (like a Banner item).

-

- **7. Admin CRUD Operations via API**

- Use the **PatientsController API**
- The rendering (e.g., React or vanilla JS inside Sitecore) can:
 - Show **patient list** from GET `/api/patients/Search`.
 - Provide a form for **Add Patient** (POST `/api/patients/Add`).
 - Support **Edit Patient** (PUT `/api/patients/UpdateById/{id}`).
 - Allow **Delete Patient** (DELETE `/api/patients/DeleteById/{id}`).