ChatGPT

ASP.NET Core Web API Development Plan

Ticket 1: Project Setup and API Structure

User Story: As a developer, I want to initialize a new ASP.NET Core Web API project with a clean, layered solution structure (Controllers, Models, Services, Data Access) so that the codebase is organized and maintainable [1].
Acceptance Criteria:
- [ ] New project created (using Visual Studio template or `dotnet new webapi`) [2].
- [ ] Folders exist for Controllers, Models/DTOs, Services (business logic), and Data Access (repositories) [1].
- [ ] `Program.cs` (or `Startup.cs`) is configured for Dependency Injection (services configured).
- [ ] A default controller (e.g., `WeatherForecastController`) is present and returns an HTTP 200 response when the project runs.

Ticket 2: Implement CRUD Endpoints

User Story: As a developer, I want to implement standard CRUD operations (Create, Read, Update, Delete) for the primary resource (e.g., TodoItem) so that clients can manage data through the API [3].
Acceptance Criteria:
- [ ] A model/entity (e.g., `TodoItem`) is defined in the project.
- [ ] `GET /api/[entities]` returns a list of items (HTTP 200 with JSON array) [3].
- [ ] `GET /api/[entities]/{id}` returns a single item by ID (HTTP 200 if found, or 404 if not).
- [ ] `POST /api/[entities]` accepts JSON body and creates a new item (returns HTTP 201 Created or 200 with new item) [3].
- [ ] `PUT /api/[entities]/{id}` updates an existing item with given ID (returns HTTP 204 No Content or 200) [3].
- [ ] `DELETE /api/[entities]/{id}` deletes the item with given ID (returns HTTP 204 No Content) [3].
- [ ] Endpoints return appropriate HTTP status codes and content (e.g., 404 for missing items, 400 for validation errors).

Ticket 3: Add Custom HTTP Headers to Responses

User Story: As an API client, I want custom headers (e.g., `X-Api-Version` or `X-Response-Time`) included in responses so that the server can send extra metadata (like API version or timing) [4].
Acceptance Criteria:
- [ ] Custom header(s) are added in responses. For example, use `HttpContext.Response.Headers.Add("X-My-Header", value)` in a controller action.
- [ ] If a header should be global, add it in middleware (e.g., in `Program.cs`: `app.Use(async (context, next) => { context.Response.Headers.Add("X-My-Header", "value"); await next(); });`) [5].
- [ ] Custom header appears in Postman/browser network tab for relevant endpoints.

- [ ] CORS policy is configured to expose custom headers if needed (e.g., `.AllowAnyOrigin().WithExposedHeaders("X-My-Header")` in `AddCors`) [6].

Ticket 4: Implement JWT Authentication

User Story: As a developer, I want to secure the API with JWT Bearer authentication so that only requests with a valid token can access protected endpoints [7].
Acceptance Criteria:
- [ ] Install the `Microsoft.AspNetCore.Authentication.JwtBearer` NuGet package.
- [ ] Configure JWT in `Program.cs`: call `services.AddAuthentication(JwtBearerDefaults.AuthenticationScheme).AddJwtBearer(...)` with valid issuer, audience, and signing key parameters [8].
- [ ] (Optional) Implement a login endpoint that validates credentials and issues a JWT token.
- [ ] Protect endpoints by adding the `[Authorize]` attribute so that they require a valid JWT.
- [ ] Requests missing a valid token receive HTTP 401 Unauthorized (invalid token yields 401 as well).

Ticket 5: Implement Role-based Authorization

User Story: As a system administrator, I want to restrict certain API actions to specific user roles (e.g., Admin, Editor) so that only authorized roles can perform privileged operations [9].
Acceptance Criteria:
- [ ] Define user roles (e.g., in Identity or application configuration) and assign roles to users.
- [ ] Ensure JWT tokens include a `role` claim for each user with their roles.
- [ ] Apply `[Authorize(Roles = "...")]` on controllers or actions to restrict access to specific roles [9]. For example: `[Authorize(Roles = "Administrator")]`.
- [ ] Users without the required role receive HTTP 403 Forbidden when accessing those endpoints.

Ticket 6: Apply Security Best Practices (HTTPS, CORS, Rate Limiting)

User Story: As a developer, I want to enforce best practices (HTTPS, CORS policies, and rate limiting) so that the API is secure in transit and protected from abuse [10] [11].
Acceptance Criteria:
- [ ] **HTTPS:** Configure the app to only use HTTPS. In production, use `app.UseHttpsRedirection()` or listen on HTTPS only [10]. The API should reject or not listen to HTTP requests.
- [ ] **CORS:** Configure CORS with `services.AddCors` and `app.UseCors`. Only allow necessary origins (e.g., `AllowAnyOrigin` for local dev, restrict in prod) [6]. Specify allowed methods and headers.
- [ ] **Rate Limiting:** Add the ASP.NET Core Rate Limiting middleware (`services.AddRateLimiter`) and define policies (e.g., limit to N requests per minute per client) to throttle excessive requests [11].
- [ ] **Other Security Headers:** (Optional) Enable HSTS, set security headers (CSP, X-Content-Type-Options, etc.) as appropriate.
- [ ] Test: CORS preflight (OPTIONS) should succeed for allowed domains, and exceeding rate limits should return HTTP 429.

Ticket 7: Logging and Global Error Handling

User Story: As a developer, I want to set up structured logging and global exception handling so that errors are recorded and clients get consistent error responses [12] [13].
Acceptance Criteria:
- [ ] **Logging:** Integrate a logging framework (e.g., Serilog) or use built-in `ILogger`. Configure it in `Program.cs` to log to console/file [13].
- [ ] **Exception Handling:** In production, configure `app.UseExceptionHandler("/Error")` (or a custom handler) to catch unhandled exceptions and log them [12]. In development, use `app.UseDeveloperExceptionPage()` for detailed error pages.
- [ ] Ensure all exceptions are logged. Unhandled exceptions should result in HTTP 500 (with a generic error message returned, not stack trace).

Ticket 8: Manual API Testing with Postman

User Story: As a tester or developer, I want to use Postman to test all API endpoints so that I can verify their behavior (status codes and response data) [14] [15].
Acceptance Criteria:
- [ ] Create Postman requests for each endpoint.
- [ ] **GET endpoints:** Send GET requests and verify response status is 200 OK and the JSON matches expected data [14].
- [ ] **POST/PUT endpoints:** Send requests with JSON bodies in the Body (set type to `application/json`) and verify status (e.g., 200 OK or 201 Created) [15].
- [ ] **DELETE endpoints:** Send DELETE requests and verify the item is removed and returns 204 No Content (or 200) as expected.
- [ ] Test protected endpoints by adding an `Authorization: Bearer <token>` header. Verify that missing or invalid token yields 401.

Ticket 9: Implement API Versioning

User Story: As a developer, I want to version the API so that I can introduce non-breaking and breaking changes without breaking existing clients [16].
Acceptance Criteria:
- [ ] Add an API versioning library (e.g., `Asp.Versioning.Http` or `Microsoft.AspNetCore.Mvc.Versioning`).
- [ ] Define supported API versions (e.g., 1.0, 2.0) in configuration or via attributes.
- [ ] Choose a versioning scheme: add version to route (e.g., `Route("api/v{version:apiVersion}/[controller]")`) or use header/query versioning.
- [ ] Decorate controllers with `[ApiVersion("1.0")]`, etc., and update routes accordingly.
- [ ] Ensure Swagger UI shows versioned endpoints under separate groups.
- [ ] (Optional) Mark old versions as deprecated (e.g., `ApiVersion(..., Deprecated = true)`) and return warnings for deprecated versions.
- [ ] Document versioning strategy in README.

Ticket 10: Write Unit Tests

User Story: As a developer, I want to write automated unit tests for the API logic so that changes are verified and regressions are caught early [17] [18].
Acceptance Criteria:
- [ ] Create a test project (e.g., using xUnit) referencing the Web API project.
- [ ] Write unit tests for controllers and services, following the Arrange-Act-Assert pattern [18].
- [ ] Use `[Fact]` and `[Theory]` (xUnit attributes) for test methods [17].
- [ ] Use mocking (e.g., Moq) to replace external dependencies (database, other services) during testing.
- [ ] Tests cover key scenarios (successful calls and error cases) and all pass before code is merged.

Ticket 11: Add Swagger/OpenAPI Documentation

User Story: As a developer or API consumer, I want Swagger UI integrated so that the API is self-documented and endpoints can be tested through a web interface [19].
Acceptance Criteria:
- [ ] Install Swashbuckle.AspNetCore (Swagger) NuGet package.
- [ ] In `Program.cs`, call `builder.Services.AddEndpointsApiExplorer()` and `builder.Services.AddSwaggerGen()` [19].
- [ ] Enable middleware: in development, use `app.UseSwagger()` and `app.UseSwaggerUI()` [19].
- [ ] Verify Swagger UI is accessible (typically at `/swagger`) and lists all endpoints with schemas.
- [ ] If needed, enable XML comments in project and configure Swagger to include them for richer docs.

Ticket 12: Integrate Database with Entity Framework Core

User Story: As a developer, I want to use a real database (e.g., SQL Server with EF Core) so that data is persisted between runs and managed with migrations [20].
Acceptance Criteria:
- [ ] Add EF Core packages (`Microsoft.EntityFrameworkCore` and a provider like `Microsoft.EntityFrameworkCore.SqlServer` or `Microsoft.EntityFrameworkCore.InMemory`) [20].
- [ ] Create a `DbContext` class and register it in `Program.cs` with `services.AddDbContext<>()`, pointing to the connection string.
- [ ] Add a connection string in `appsettings.json` (e.g., for SQL Server). Use `options.UseSqlServer(Configuration.GetConnectionString("Default"))`.
- [ ] Create and apply migrations: run `dotnet ef migrations add InitialCreate` and `dotnet ef database update` to create database schema.
- [ ] Update repository or controller code to use `DbContext` instead of in-memory collections.
- [ ] Verify that data is stored and retrieved from the database (e.g., by querying persisted records after restart).

Ticket 13: Environment Configuration and Secrets

User Story: As a developer, I want to manage configuration per environment and protect secrets so that sensitive data (API keys, DB passwords) are not in source control [21] [22].

Acceptance Criteria:
- [ ] Use `appsettings.json` for default settings and `appsettings.Development.json` (or `Production.json`) for environment-specific overrides.
- [ ] Store secrets and sensitive values (e.g., JWT signing key, DB password) using the Secret Manager (for development) or environment variables (for production) [21] [22].
- [ ] Ensure `ASPNETCORE_ENVIRONMENT` is set appropriately in each environment to select the correct config file.
- [ ] Confirm configuration values (especially secrets) are loaded via the `Configuration` system and not hard-coded or checked into source.

Ticket 14: Dockerization and Deployment (Optional)

User Story: As a DevOps engineer, I want to containerize the API using Docker and deploy it so that it runs reliably in any environment [23].
Acceptance Criteria:
- [ ] Create a `Dockerfile` at the project root using multi-stage builds (build on the .NET SDK image, run on the ASP.NET Core runtime image) [23].
- [ ] Build the Docker image (e.g., `docker build -t myapi:latest .`) and run a container locally to verify the API responds correctly [24].
- [ ] Push the Docker image to a container registry (e.g., Docker Hub, Azure Container Registry).
- [ ] Deploy the container to a hosting environment (e.g., Azure App Service for Containers or Kubernetes). Verify the API functions correctly in the deployed environment.

Sources: Official documentation and tutorials for ASP.NET Core Web API, authentication, authorization, and best practices [1] [2] [3] [4] [5] [8] [9] [10] [11] [13] [12] [14] [16] [17] [19] [20] [21] [23].

---

[1] **Best Practices for Structuring ASP.NET Core Web API Projects | by Yohan Malshika | Medium**
https://malshikay.medium.com/best-practices-for-structuring-asp-net-core-web-api-projects-80880e3a5b85

[2] [3] [20] **Tutorial: Create a controller-based web API with ASP.NET Core | Microsoft Learn**
https://learn.microsoft.com/en-us/aspnet/core/tutorials/first-web-api?view=aspnetcore-9.0

[4] [5] [6] **How to Add Custom Headers in ASP.NET Core Web API**
https://code-maze.com/aspnetcore-add-custom-headers/

[7] [8] **Configure JWT bearer authentication in ASP.NET Core | Microsoft Learn**
https://learn.microsoft.com/en-us/aspnet/core/security/authentication/configure-jwt-bearer-authentication?view=aspnetcore-9.0

[9] **Role-based authorization in ASP.NET Core | Microsoft Learn**
https://learn.microsoft.com/en-us/aspnet/core/security/authorization/roles?view=aspnetcore-9.0

[10] **Enforce HTTPS in ASP.NET Core | Microsoft Learn**
https://learn.microsoft.com/en-us/aspnet/core/security/enforcing-ssl?view=aspnetcore-9.0

[11] **Rate limiting middleware in ASP.NET Core | Microsoft Learn**
https://learn.microsoft.com/en-us/aspnet/core/performance/rate-limit?view=aspnetcore-9.0

[12] **Handle errors in ASP.NET Core | Microsoft Learn**
https://learn.microsoft.com/en-us/aspnet/core/fundamentals/error-handling?view=aspnetcore-9.0

[13] **Structured Logging with Serilog in ASP.NET Core - The Complete Guide for .NET 8 Applications - codewithmukesh**
https://codewithmukesh.com/blog/structured-logging-with-serilog-in-aspnet-core/

[14] [15] **How To Use Postman With ASP.NET Core Web API Testing**
https://www.c-sharpcorner.com/article/how-to-use-postman-with-asp-net-core-web-api-testing/

[16] **API Versioning in ASP.NET Core**
https://www.milanjovanovic.tech/blog/api-versioning-in-aspnetcore

[17] [18] **Implementation Of Unit Test Using Xunit And Moq in .NET Core 6 Web API**
https://www.c-sharpcorner.com/blogs/implementation-of-unit-test-using-xunit-and-moq-in-net-core-6-web-api

[19] **ASP.NET Core web API documentation with Swagger / OpenAPI | Microsoft Learn**
https://learn.microsoft.com/en-us/aspnet/core/tutorials/web-api-help-pages-using-swagger?view=aspnetcore-8.0

[21] [22] **Safe storage of app secrets in development in ASP.NET Core | Microsoft Learn**
https://learn.microsoft.com/en-us/aspnet/core/security/app-secrets?view=aspnetcore-9.0

[23] [24] **Containerize an app with Docker tutorial - .NET | Microsoft Learn**
https://learn.microsoft.com/en-us/dotnet/core/docker/build-container