ChatGPT

FInShark Project Documentation

Project Overview

FInShark is a finance-oriented web API built on ASP.NET Core with a SQL Server backend. It provides features for **user account management**, **stock data retrieval**, **portfolio tracking**, and **user comments/notes** on stocks. In essence, it lets users register/login, search stock information, maintain a personalized stock portfolio (add/remove stocks), and attach comments to stocks. Similar FinShark projects describe it as a *"financial platform where people can get real-time stock data, manage their own stock portfolios, post and read comments on stocks."* [1] [2]. The solution is organized as a Visual Studio solution (`FinShark.sln`) containing an ASP.NET Core Web API project (in an `api` folder) that uses Entity Framework Core to interface with SQL Server.

Setup and Configuration

1. **Prerequisites:** Install .NET 8 SDK and SQL Server (or SQL Server Express) on your machine. Optionally use SQL Server Management Studio (SSMS) or Azure Data Studio to manage the database.
2. **Clone Repository:**

   ```
   git clone https://github.com/Khushtaunk28/FInShark.git
   cd FInShark/api
   ```

3. **Database Creation:**
4. Create a new SQL Server database (e.g. named `FinShark`) on your local instance.
5. In the `api` project directory, add or edit `appsettings.json` to include your connection string and any secrets. For example, you might have:

   ```
   {
     "ConnectionStrings": {
       "DefaultConnection":
   "Server=YOUR_SERVER_NAME;Database=FinShark;Trusted_Connection=True;MultipleActiveResultSe
     },
     "JWT": {
       "Issuer": "http://localhost:5000",
       "SigningKey": "YOUR_SECRET_KEY"
     }
   }
   ```

   This follows the pattern shown in related FinShark projects [3] [4]. Make sure the `DefaultConnection` matches your server name and database.
6. **Apply Migrations:** The project uses EF Core code-first migrations. Run the following to create the schema in your database:

```
dotnet restore
dotnet ef migrations add Init
dotnet ef database update
```

This will create tables for the entities (e.g. Users, Stocks, Portfolio, Comments, etc.) in your SQL Server database [5] [4].

7. **Run the API:** Start the ASP.NET Core Web API (default URL is `http://localhost:5000` or `https://localhost:5001`):

```
dotnet run
```

The API should now be listening locally. (If a Swagger/OpenAPI middleware is included, you can browse to `/swagger` to explore the endpoints.)

Major Functionalities

FInShark implements the following key features (based on the code structure and analogous projects):

- **User Authentication** – Secure registration and login (likely using ASP.NET Identity or JWT). Users can create accounts and log in to manage their data [6] [7].
- **Stock Search/Insights** – Retrieve stock information (e.g. current price, fundamentals). The API likely calls an external stock data service (e.g. Financial Modeling Prep) under the hood to fetch real-time data.
- **Portfolio Management** – Each authenticated user can add or remove stocks from their personal portfolio. This involves CRUD operations on a "Portfolio" entity that links users to stock symbols [8] [7].
- **User Comments** – Users can attach comments or notes to specific stocks. The system supports creating and retrieving comments associated with stocks [2] [9].
- **Data Transfer and Validation** – The API likely uses Data Transfer Objects (DTOs) to shape request/response payloads and employs model validation attributes. Controllers return `IActionResult` responses with appropriate HTTP status codes, as seen in similar FinShark projects [10] [7].

API Endpoints

The Web API exposes RESTful endpoints under an `/api` route. While the exact routes depend on the controllers, a typical set might include:

Method	Endpoint	Request Body	Description
POST	`/api/auth/register`	`{ email, password, ... }`	Register a new user account.
POST	`/api/auth/login`	`{ email, password }`	Log in a user, returning a JWT token.
GET	`/api/stocks`	(query params: search/filter)	Search/list available stocks or quotes.

Method	Endpoint	Request Body	Description
GET	`/api/stocks/{symbol}`	–	Get details for a specific stock symbol.
GET	`/api/portfolio`	–	Get current user's portfolio entries. (Auth)
POST	`/api/portfolio`	`{ symbol, quantity }`	Add a stock to user's portfolio.
DELETE	`/api/portfolio/{id}`	–	Remove a stock entry from portfolio by ID.
GET	`/api/comments`	`?symbol={symbol}`	List comments for a given stock.
POST	`/api/comments`	`{ symbol, text }`	Add a new comment for a stock.
GET	`/api/users/profile`	–	(Optional) Get current user profile info.

Each endpoint uses standard ASP.NET Core attributes like `[HttpGet]`, `[HttpPost]`, `[FromBody]`, etc., for routing and binding [10]. Request/response bodies use JSON. For example, creating a portfolio entry or comment expects a JSON object in the request body. Responses include status codes (200, 201, 400, etc.) and data or error messages via `IActionResult` [10] [7].

Architecture & Project Structure

The solution follows a layered architecture common in ASP.NET Core projects. Key components include:

- **Data Layer:** An EF Core `DbContext` (likely in a `Data/` folder) defines `DbSet` entities (User, Stock, Portfolio, Comment, etc.). Migrations (in `Migrations/`) track schema changes. The connection to SQL Server is configured here via the connection string.
- **Models and DTOs:** Domain models (in `Models/`) represent database tables. Separate DTO classes (in `DTOs/`) define shapes for API requests/responses. Mapping between models and DTOs is probably handled by a library like AutoMapper (noted in dependencies) or by custom mappers [11] [7].
- **Repositories & Services:** A `Repository` folder (and corresponding `Interfaces`) contains classes following the Repository pattern for data access. A `Service` layer implements business logic on top of repositories. Dependency Injection is used throughout (registered in `Program.cs`) to inject repositories and services into controllers [7].
- **API Controllers:** Controllers (in `Controllers/`) define the HTTP endpoints. Each controller action calls services/repositories and returns results. Typical controllers might include `AuthController`, `StocksController`, `PortfolioController`, `CommentsController`, etc. Attributes like `[ApiController]` and `[Route("api/[controller]")]` are used on controllers [10].
- **Security:** If implemented, ASP.NET Identity (or JWT auth middleware) handles user authentication. The project's `Startup` (or `Program.cs` in .NET 6/8) configures JWT bearer authentication with the signing key from `appsettings.json`.
- **Dependencies:** The project uses packages such as `Microsoft.EntityFrameworkCore.SqlServer` (for EF Core SQL Server) and

`Microsoft.AspNetCore.Authentication.JwtBearer` (for JWT) [11]. AutoMapper is included for object mapping. Swagger/OpenAPI (via `Microsoft.AspNetCore.OpenApi`) may provide API documentation. These are all listed in the project's dependencies [11].

Overall, this layered setup promotes separation of concerns: controllers handle HTTP, services implement logic, repositories handle data access, and EF Core maps objects to the SQL Server schema [7] [11].

Technologies Used

- **C# and .NET 8 / ASP.NET Core:** Core framework for building the API and server-side logic. ASP.NET Core's high performance and cross-platform capabilities are leveraged [12] [11].
- **Entity Framework Core (EF Core):** ORM used for database access. EF Core auto-maps C# classes (models) to SQL tables and supports LINQ queries [11] [5]. Migrations generate the schema.
- **SQL Server:** The relational database system for persisting data. The project connects via a SQL Server connection string (e.g. using `SqlServer` provider) [12] [3].
- **JWT (JSON Web Tokens):** Used for stateless user authentication. The ASP.NET Core JWT middleware issues and validates tokens for protected endpoints [12] [7].
- **ASP.NET Core Identity:** (Likely) for user account management (storing passwords, roles, etc.), as hinted by identity-related dependencies [11].
- **AutoMapper:** Library for mapping between domain models and DTOs [11].
- **Swagger / OpenAPI:** Tools for API documentation/testing, as indicated by the `Microsoft.AspNetCore.OpenApi` package [11].
- **Other Utilities:** Newtonsoft.Json (for JSON serialization) and various ASP.NET Core libraries (MVC, Logging, etc.) [11].
- **Development Tools:** Visual Studio or VS Code with C# extensions, EF Core CLI for migrations.

Deployment Notes

FInShark can be deployed like any ASP.NET Core application. For production:
- **Publish:** Use `dotnet publish` to compile the app and package the output. This produces a folder of files to deploy.
- **Configure Environment:** Set up the production `appsettings.json` (or environment variables) with the live SQL Server connection string and JWT secret. Ensure the database exists or use migrations to create it.
- **Hosting:** The app can run on Windows (IIS), Linux (Kestrel/nginx), or in a container. For Azure, one could deploy to Azure App Service or Azure Container Instances.
- **CORS and HTTPS:** If a separate frontend consumes the API, configure CORS policies. Always use HTTPS in production for security.
- **Note:** The source repo had no explicit deployment scripts, but these steps follow standard ASP.NET Core deployment practices. (No specific cloud deployment info was found in the repository's docs.)

Database Schema & ER Diagram

The SQL schema defines tables for Users, Stocks, Portfolios, and Comments. The relationships are: each user can own many stocks via the Portfolio table (a many-to-many *User–Stock* linkage with quantity), and users can leave multiple comments on any stock.

Below is a conceptual ER diagram (inferred from the models) illustrating the main entities and relationships:

【diagram†embed_image】 *Figure: Entity-Relationship diagram of the FInShark SQL database.*

- **User** (UserID PK, Name, Email, etc.) ––< owns >–– **Portfolio** (PortfolioID PK, UserID FK, StockID FK, Quantity) ––< contains >–– **Stock** (StockID PK, Symbol, CompanyName, etc.).
- **User** ––< writes >–– **Comment** (CommentID PK, UserID FK, StockID FK, Content, Timestamp).
- **Stock** ––< receives >–– **Comment**.

This captures that a **User** can have many **Portfolio** entries (each linking to a **Stock** with a quantity) and can write many **Comments** on various **Stocks**. Each **Stock** can appear in many users' portfolios and have many associated comments.

Each entity's PK/FK relationships match the SQL tables defined via Entity Framework Core in the project's migrations.

---

1 3 5 12 GitHub - YaruZeng/FinShark: A stock portfolio platform built by C# ASP.NET, EF Core, React, TypeScript, TailwindCSS, and JWT.
https://github.com/YaruZeng/FinShark

2 6 8 9 GitHub - VedadOhran/Finshark-Api: Full stack web application that informs about stock data
https://github.com/VedadOhran/Finshark-Api

4 11 GitHub - ManasesLovera/finshark: Web API finances app
https://github.com/ManasesLovera/finshark

7 GitHub - nikeshkrjha/aspnet-core-finshark-webapi: A Web API Project Using ASP.NET Core
https://github.com/nikeshkrjha/aspnet-core-finshark-webapi

10 GitHub - Ynlay/FinShark: Stock Market API created for practicing .NET and SQL Server Management Studio
https://github.com/Ynlay/FinShark