News API Assignment - Liran Rouzentur

1. Possible Solution: All-in-One Web API (Monolith)

One single .NET Web API project that does everything:

- Fetches data every hour (in a background job) from News API
- Stores it in a database (SQL)
- Serves all API endpoints while handling authentication

.NET Core Structure

Controllers

AuthController.cs	# User registration, login, JWT token management
NewsController.cs	# Public and authorized news retrieval endpoints

Services

IAuthService.cs	# Authentication service interface
AuthService.cs	# Authentication service implementation
INewsService.cs	# News service interface
NewsService.cs	# News service implementation
INewsJobService.cs	# News fetching service interface (polygon.io.)
NewsJobService.cs	# News fetching service implementation (polygon.io.)

Data

ApplicationDbContext.cs	# Entity Framework DbContext

Models

…

…

SQL Structure

USERS

NEWS

TICKERS

Scheduled JOB

We would use a third-party service like “Hangfire” to control and fire the data fetch call to polygon.io.

Pros	**Cons**
Fast to build and test	Harder to scale ingestion separately
Single codebase, easy deploy	Failures may affect API
Minimal setup	Less flexible long-term

----

2. Possible Solution: Microservices (3 Separate Services)

Each piece is its own deployable app.

Similar files that were listed in the first solution, but divided into microservices

API Gateway: Single point of access to a **"centralized news service".**
Leads to three micro-services:

1. Micro-service: data fetch and store from polygon.io.
   Azure Function: Timer job.
   Would run and fetch on an hourly basis triggered at Azure.
   Gives us the option to create more fetch jobs in the future in a centralized place.
2. Micro-service: Auth API (Login and Subscription)
   User would receive a JWT token that would authorize him to use the News API.
   Gives us the option to create more user type functions like, deeper registration process, token management, forgot PW and more

3. Micro-service: News API

   The data fetch API, this is the API that would fetch the stored news data.

Pros	Cons
Clean separation	Slower to implement
Scales independently	More moving parts to test and deploy
	Overkill for current assignment scope

As a full-stack developer, I typically approach system design with the following mindset:

1. Expected outcome – What are the core features? What's the actual app size?

2. Scalability needs – Is this a lightweight project with no real growth plans? Then don't over-engineer it.

3. Budget constraints – Are we forced to use outdated tools or shortcuts due to lack of funding?

Taking all of this under consideration:

3. Selected Solution: Hybrid (Monolith + Azure Function Fetcher)

Given the project's small scale, undefined budget, and unknown future requirements, I'd structure it as a single service for user integration, with a separate, lightweight service dedicated to fetching data from the external source. I believe it's critical to separate the data source from how users consume it. We can't afford to have our data collection depend on a less stable or less controlled runtime environment.

Task List

(All estimations are for pre AI powered IDE's area.........)

1. Set up the core .NET Web API structure

Estimate: 0.5 day

- Scaffold project structure with folders for Controllers, Services, Models, Data
- Configure Startup.cs / Program.cs (dependency injection, EF, JWT)
- Set up ApplicationDbContext, migrations, and initial database schema

2. Implement Authentication Flow

Estimate: 1 day

- Create AuthController, IAuthService, AuthService
- Register/login endpoints
- Generate and validate JWT tokens
- Configure [Authorize] attribute globally
- Add user entity to EF model

3. Build News API Endpoints (secured + public)

Estimate: 2 days

- Create NewsController, INewsService, NewsService
- Endpoints:
  - GET /news (all)
  - GET /news/recent?n=...
  - GET /news/instrument?limit=...
  - GET /news/search?text=...
  - GET /news/latest (public)
  - POST /subscribe
- Include pagination or limits where appropriate

4. Create the Azure Function for data fetching

Estimate: 2 days

- Create a timer-triggered function (hourly)
- Fetch news from Polygon.io
- Parse and enrich each item
- Store items in shared DB
- Handle retries, errors, logging

5. Data Layer and Models

Estimate: 2 day

- Design NewsItem, User, and Subscription models
- Add EF Core configurations
- Apply migrations or schema setup