**What I have implemented**

* **News Background Service Worker**: background job that runs every hour to fetch news.
* **News Enrichment**: added mock enrichment with instrument metadata
* **News APIs**: implemented APIs that allow retrieving news from the repository in multiple ways (free text search, dates range, etc).
* **Repository**: due to time limitations, used in-memory caching (ConcurrentDictionary) to simulate DB tables and enforce uniqueness.
* **Subscription**: functionality explained with pseudocode.
* **Authentication & Authorization**: due to time limitations,I implemented a mock JWT Bearer for authorization. In real production system, this would be implemented using Oauth/OpenID/ other authorization framework.
* **Swagger**: I added for API documentation and testing. An authorize key provided in appsettings. For test purpose, because JWT wasn’t implemented with code

**Architecture Plan:**

There are 2 approaches for design the following task project:

1. Monolithic architecture
2. Microservices based distributed architecture

**Monolithic Architecture:**

Components:

* **Auth**: Responsible for authenticate and authorize user and user`s actions
* **News Fetcher Job**: A scheduled task which retrieves updated news Articles from a provider(i.e. Polygon) enrich each article with extra data for marketing purpose
* **Repository-Storage** : Data Access layer for storing data(news,users,subscription)
* **Web API**: RESTful endpoint which handles client requests and responses
* **Caching**: storing frequently accessed data for improve performance
* **External Integration:** manage API integrations with 3rd party providers
* **Observability**: handle, analyze and manage logs, metrics

**Microservices Architecture:**

* **API Gateway**: Acts as a single entry point and responsible for routing requests to services, rate limiting the traffic, API versioning authorization and authentication.
* **NewsFetcher Service**: responsible for fetching news from External providers` APIs. Runs as a scheduled cron job
* **Enrichment service**: Responsible for extend and enrich news data with extra information for marketing(in our case) usage
* **Subscription Service**: responsible for users` subscription, including subscribe/unsubscribe user, change user`s subscription preferences
* **Notifications service**: responsible for send notifications to users
* **Messaging broker**: responsible for async communication between services and enables scaling and reliability
* **Caching**: distributed caching responsible for store frequently accessed data and handle Rate limiting.
* **Database**: responsible for storing and querying data

**Architectures comparison:**

|  |  |  |
| --- | --- | --- |
| Aspect | Monolithic | Microservices |
| Development | SIngle code base and shared resources. Easy to test and debug, but complexity grows as the system scales. | More complex to manage the development because of multiple codebases and services, but allow parallel work by independent teams |
| Observability | Easier to monitor and trace issues | Distributed system is more complex to monitor and trace. Requires tracing tools |
| Scalability | Vertical scalability. Single component scaling impossible | Single service scaling possible, allowing the system to seamlessly handle unpredictable workloads and scale to meet any unexpected demand |
| Deployment | Entire system deployed at the time. Single change requires deployment of the entire system. However, the deployment is fast and requires basic CI/CD practices | Single service deployment is possible. It allows zero-downtime of the system  Requires advanced DevOps practices |
| Fault tolerance | Fault in some module probably will affect the entire system | Faults are isolated to specific services |
| Communication | All components communicate directly with each other through simple function calls. | Components communicating over network using message queues or different network protocols and makes it more complex. |
| Flexibility | Low flexibility. It difficult to adopt new technologies | High flexibility. It is possible to develop each component with different technologies without affecting others. |

I recommend **Monolithic Architecture** for this task. It meets the company’s current needs for a simple news-fetcher service without unnecessary complexity. It enables rapid development, easier monitoring/debugging, reduced risks, and is a better approach for small teams.

**Tasks List:**

1. Project bootstrap : (**1 day**)

* Set up the project in IDE, DB and environment configurations

1. Implement Auth Infrastructure: (**3 days**)

* Implement JWT authentication infrastructure
* Create authorization middleware and policies
* Implement JWT token authorization

1. Initialize DataModel and Repository (**1.5 days**)

* Design entities schema for system entities (such as news, subscription and users, enrichment etc)
* Setup databases, including setup entity relationships, keys, indexes

1. Develop the observability infrastructure: (**2 days**)

* Setup logging and metrics including configuration and integration with monitoring tools (such as Grafana,NewRelic)
* Setup logging middleware

1. Implement Background News Fetcher Service: (**3 days**)

* Setup background job and configure it to run every 1 hour
* Implement Polygon provide integration including define API credentials, secure RestAPI call, response objects Map
* Implement rate limiter and circuit breaker
* Handle retry logic, handling errors, logging and monitoring and fallbacks

1. Implement news Enrichment service: (**1.5 days**)

* Implement API integration to external service (i.e. Polygon) to get extensive information about the ticker in order to enrich the information for marketing purpose.
* Add caching logic to store already enriched tickers with long TTL. (Cache key is tickerCode)
* Save enriched news articles data in repository and cache

1. Implement News WebAPI: (**3.5 days**)

* Create a news controller and define endpoint method for all required endpoints(6 in the assignment document) and add access policy annotations for each
* Implement each Endpoint method logic using services and repositories.
* Handle caching logic for store responses
* Add logs, metric and errors handlings

1. Implement Subscription: (**1.5 day**)

* Implement subscription controller and API endpoints
* Add an ability to unsubscribe as well
* Implement notification service and send newsletter to subscribers logic

1. Implement User service: (**1.5 day**)

* Implement user controller and API endpoint
* Add ability to users to login and new users to create an account
* Add rate limiter for security by email

1. Caching and Security: (**2.5 days**)

* Setup cache system(such as Redis) and integrate it in the system
* Add input validations such as input validations, prevent SQL injections
* Add CORS configuration
* Add logs and monitoring across the system

1. QA and Test: (**3 days**)

* Write Unit tests and integration tests for wide tests
* Add Swagger for testing and documentation purposes
* Add mocking for external integrations tests
* Write documentation for better understanding

Total Estimated time 24 man days