

Pattern Architeturali e Dependency Injection

Software Architecture Patterns

Evitare l'accoppiamento tra le componenti e richiamo di codice inutile (**coupling**)

Astrarre andando a separare (orizzontalmente - ciascuno "strato - layer" con competenze separate):

- business logic (dati)
- presentation layer (viste)
- database layer (dati)

Parametri di confronto dei pattern

Characteristic	Rating	Description
Overall agility	↓	Small changes can be properly isolated, big one are more difficult due to the monolithic nature of the pattern
Ease of deployment	↓	Difficult for larger applications, due to monolithic deployments (that have to be properly scheduled)
Testability	↑	Very easy to mock or stub layers not affected by the testing process
Performance	↓	Most of requests have to go through multiple layers
Scalability	↓	The overall granularity is too broad, making it expensive to scale
Ease of development	↑	A well know pattern. In many cases It has a direct connection with company's structure

Monolite

Architettura unica non scalabile in tutte le dimensioni = diventa difficile modificare e mantenere e da replicare

Microservizi

Oggetti distribuiti singolarmente controllabili e scomponibili.

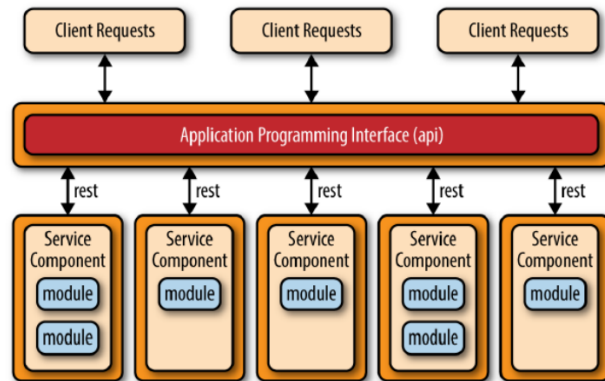
Scaling (ridimensionamento)

- Orizzontalmente = Numero di istanze
- Verticalmente = Numero di funzioni
- In profondità = Scaling dei dati

Abbiamo più tipi di topologie

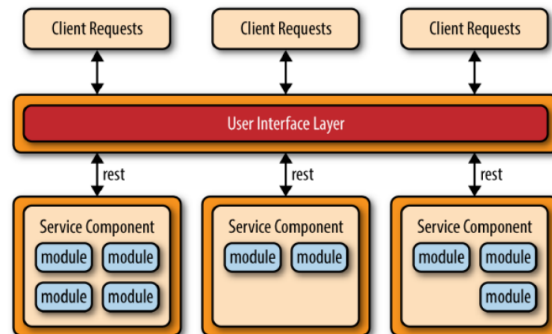
API REST

Scomporre modularmente (più a basso livello) le singole funzioni a basso livello.



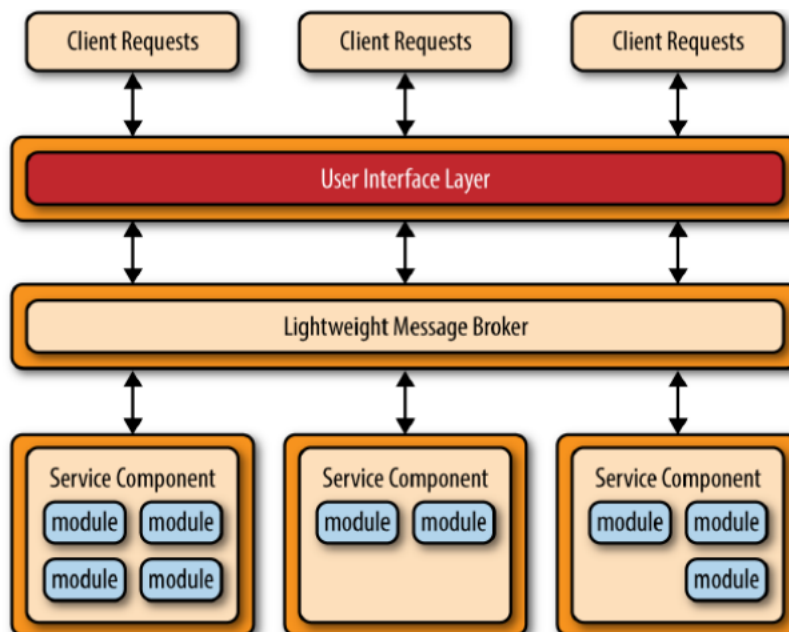
REST

Non c'è uno strato API di mezzo e c'è sempre la scomposizione in componenti singole



Message-based

Architettura fatta di comunicazione asincrona a messaggi singoli a basso livello (gestione a code, asincrone)



- Course-grained
- Too fine-grained

Dependency Injection

Ogni volta che la dipendenza concreta viene fornita nel costruttore specificando come parametro l'interfaccia e non la classe concreta si parla di Dependency Injection.

La componente dichiara solo le sue dipendenze:

- dentro il costruttore (constructor injection)

```
public class MovieLister {
    private MovieFinder finder;
    public MovieLister(MovieFinder finder) { // Dichiarazione dip.
        this.finder = finder;
    } // ...
}
```

- dentro i metodi setter (setter injection)

```
public class MovieLister {
    private MovieFinder finder;
    public void setFinder(MovieFinder finder) {
        this.finder = finder;
    } // ...
}
```

Puoi farlo con:


- framework (e.g. Google Guice)
- annotazioni (Java)
- dipendenze (Spring)

Migliorando testing, resilienza, flessibilità.

Esempio architettura utilizzabile nel tuo progetto:

Building ChatGPT-Like Experiences with Azure: A Guide to Retrieval Augmented Generation for JavaScript applications

Advances in artificial intelligence and machine learning help companies improve their customer experiences, such as the Retrieval Augmented Generation (RAG) pattern. RAG empowers businesses to create ChatGPT-like interactions tailored to their specific data sets. Using Azure OpenAI Service and Azure AI Search SDK, the RAG pattern can revolutionize the customer support experience.

 <https://devblogs.microsoft.com/azure-sdk/building-chatgpt-like-experiences-with-azure-a-guide-to-retrieval-augmented-generation-for-javascript-applications/>

 Microsoft

**RAG with
Azure AI Search for
JavaScript
Developers**

Towards Human-Bot Collaborative Software Architecting with ChatGPT

DOI: <https://doi.org/10.1145/3593434.3593468> EASE '23: Proceedings of the International Conference on Evaluation and Assessment in Software Engineering, Oulu, Finland, June 2023

 <https://dl.acm.org/doi/fullHtml/10.1145/3593434.3593468>