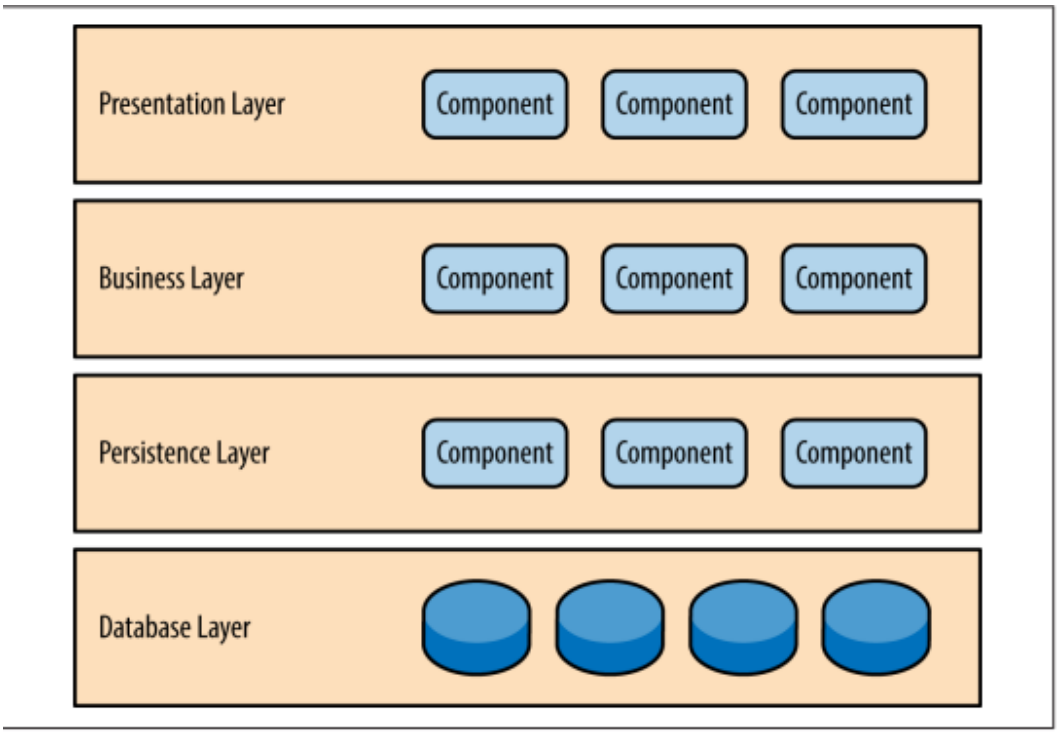
Software Architecture Pattern

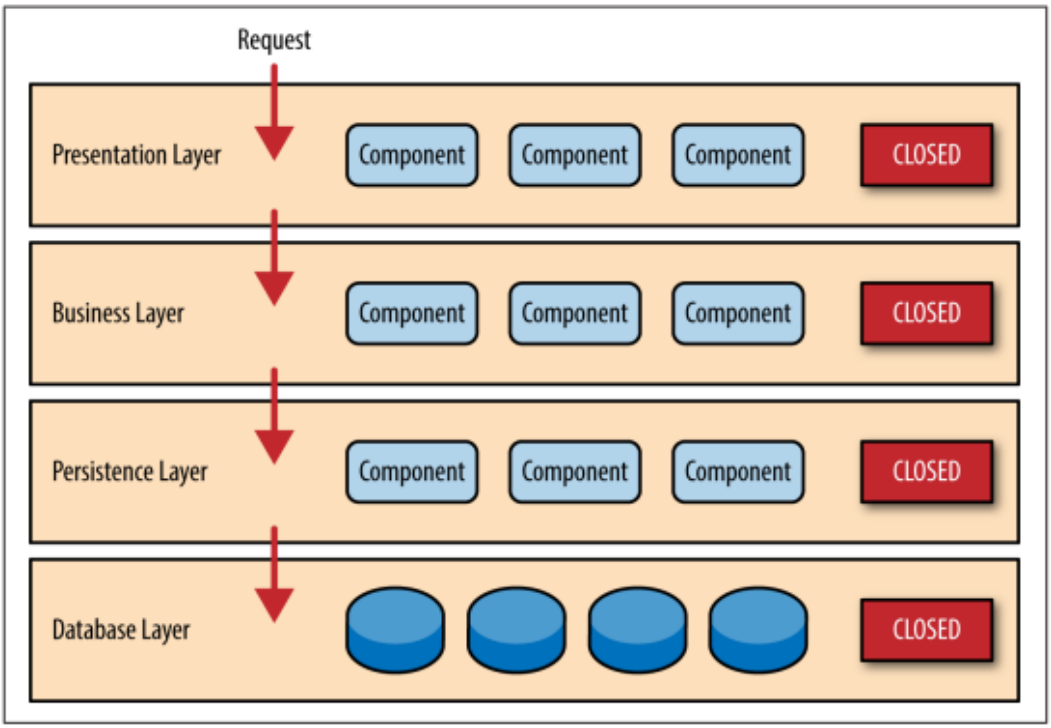
1. Layered Architecture

Components within the layered architecture pattern are organized into horizontal layers, each layer performing a specific role within the application



Each layer can only communicate with its adjacent layer

Separate and abstract each layer logic from another layer



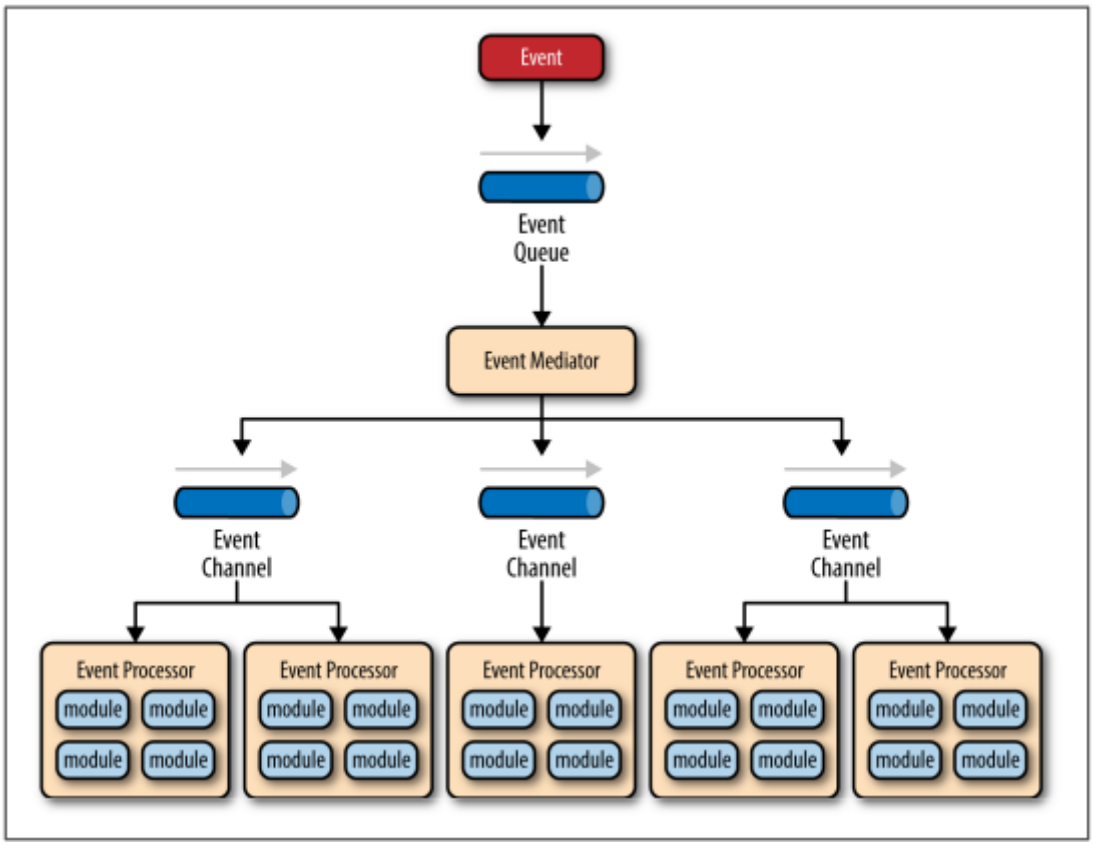
Pattern Analysis

- [ ] Overall-Agility : Low (Found Tightly Coupled Layer in this pattern Any Changes in Application Can Cause more layers to check)
- [ ] Ease of Deployment : Low (Re-Deployment an Entire Application on every change)
- [ ] Testing : High (Can Test Each Layer Separately)
- [ ] Performance : High/Low (Depend on How many Layer Request has gone through to get the data)
- [ ] Scalability : Low (Can make it more scalable by turning layer to separate Deployment or Application)
- [ ] Ease of Development : High

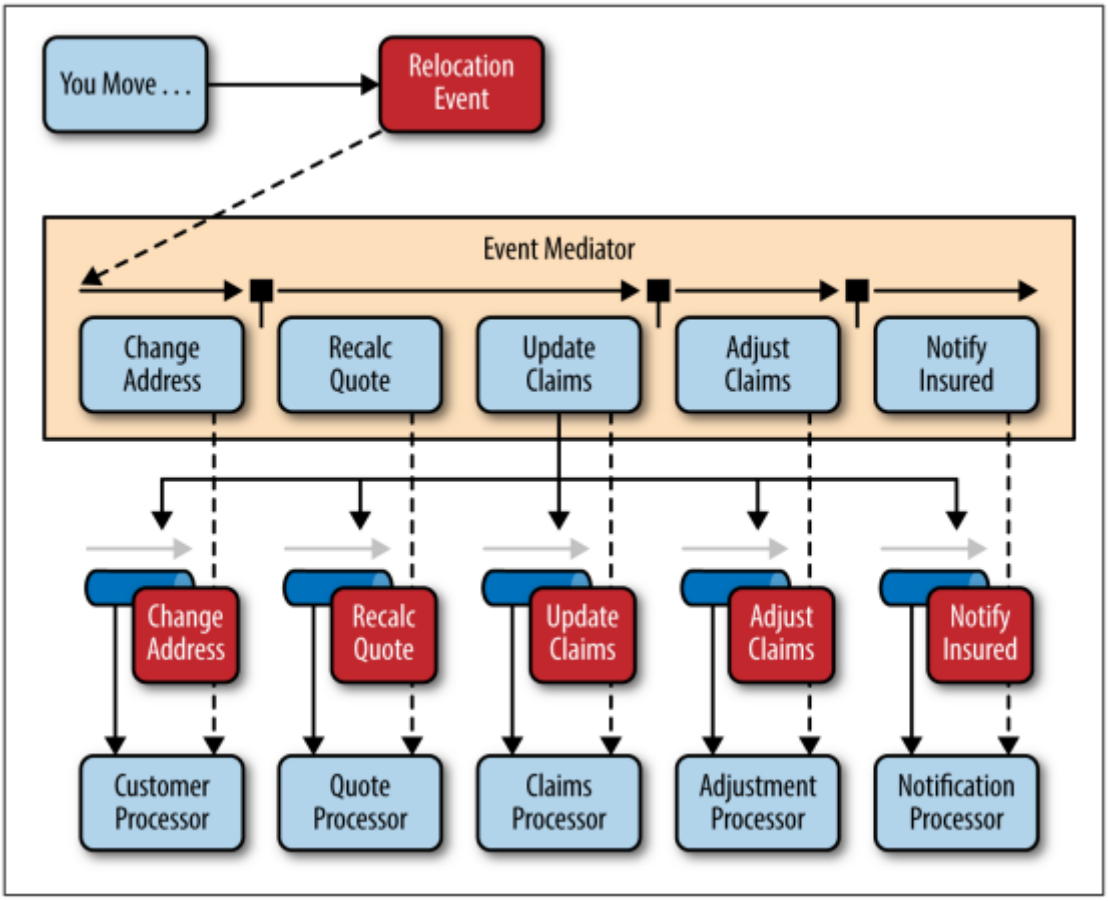
Event Driven Architecture

The event-driven architecture pattern is a popular distributed asynchronous architecture pattern. The event-driven architecture is made up of highly decoupled, single-purpose event processing components that asynchronously receive and process events.

1. Mediator Topology



2. Broker Topology



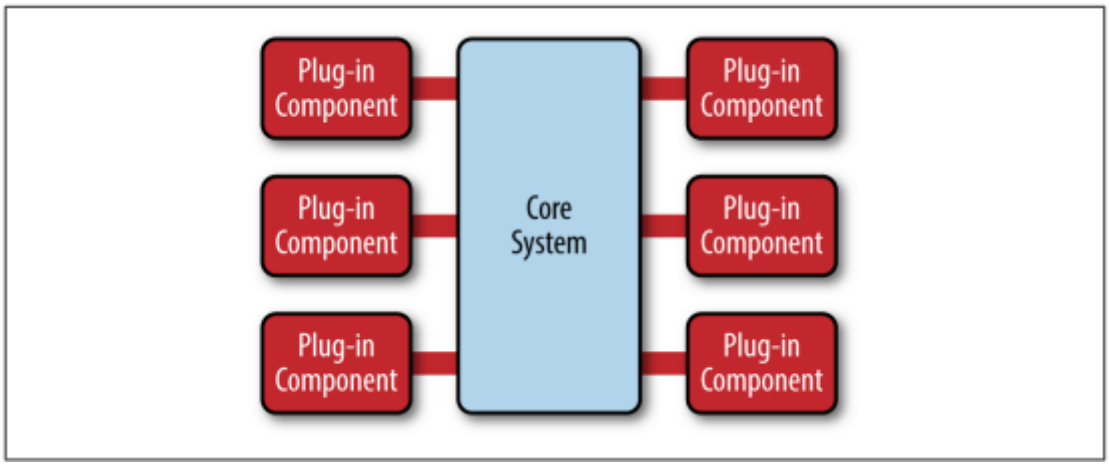
Pattern Analysis

1. Ease of Deployment : High
    a. Due to decoupled nature of this architecture
    b. event mediator is some what tightly coupled any change in events can cause event mediator to change
2. Testability : Low
    a. Due to its async nature
3. Performance : High
    a. Due to its Async nature
    b. some how it not performed very well due to its messaging infrastructure involved
4. Scalability : High
    a. Due to Decoupled architecture and async nature

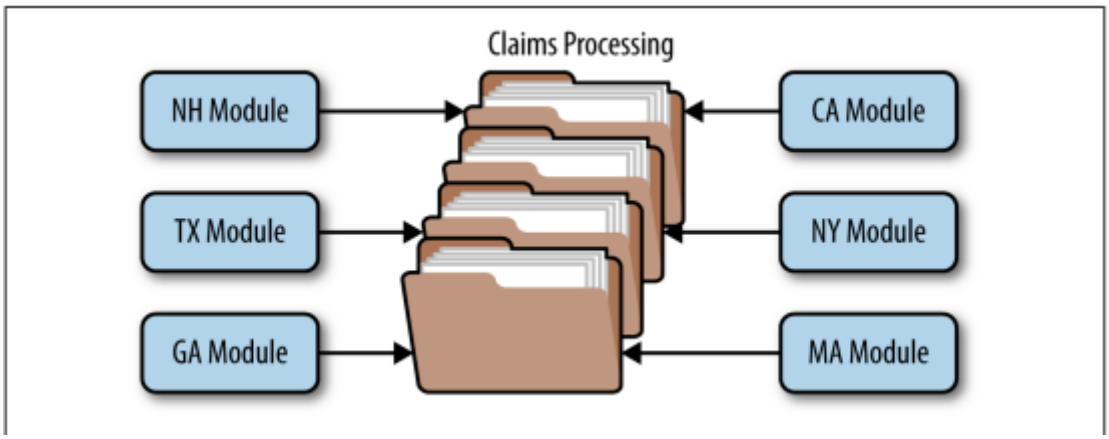
5. Ease of Development : Low
    a. Due to async nature and decoupled architecture we need to handle every possible error

3. Micro-Kernel Architecture

A product-based application is one that
is packaged and made available for download in versions as a typical third-party product.



One great thing about the microkernel architecture pattern is that it can be embedded or used as part of another architecture pattern.

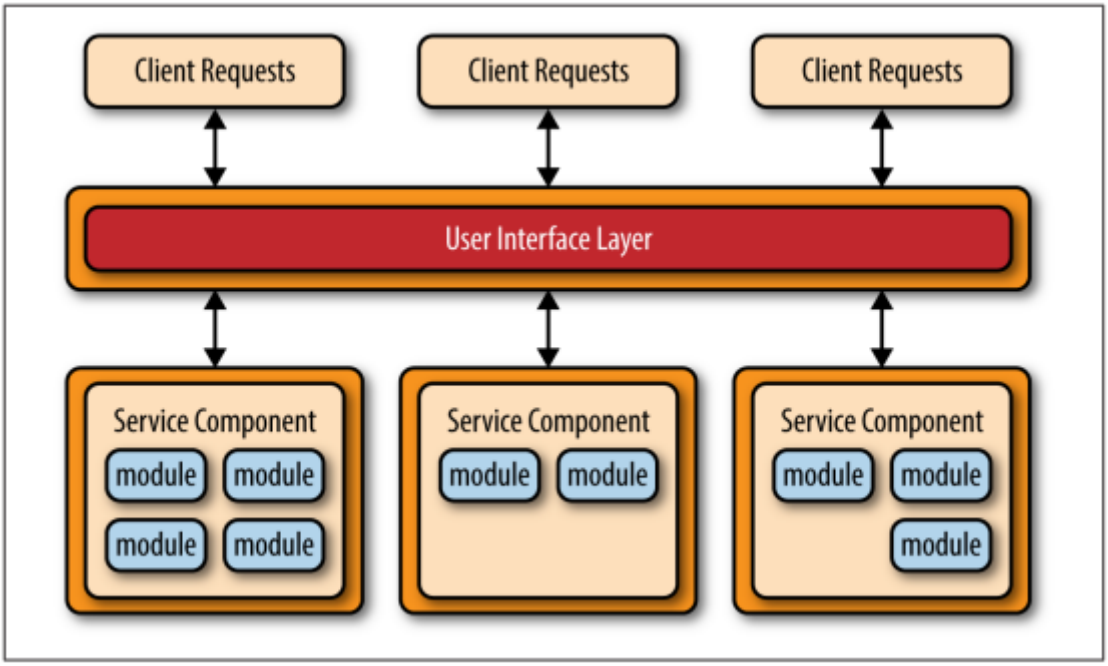


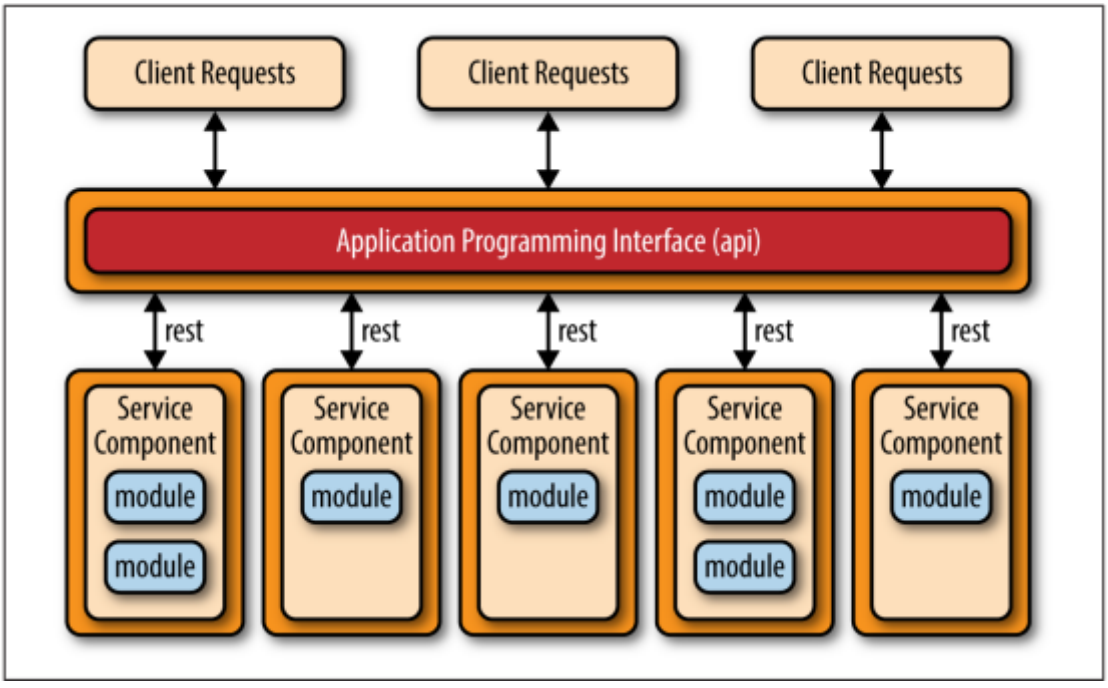
Pattern Analysis

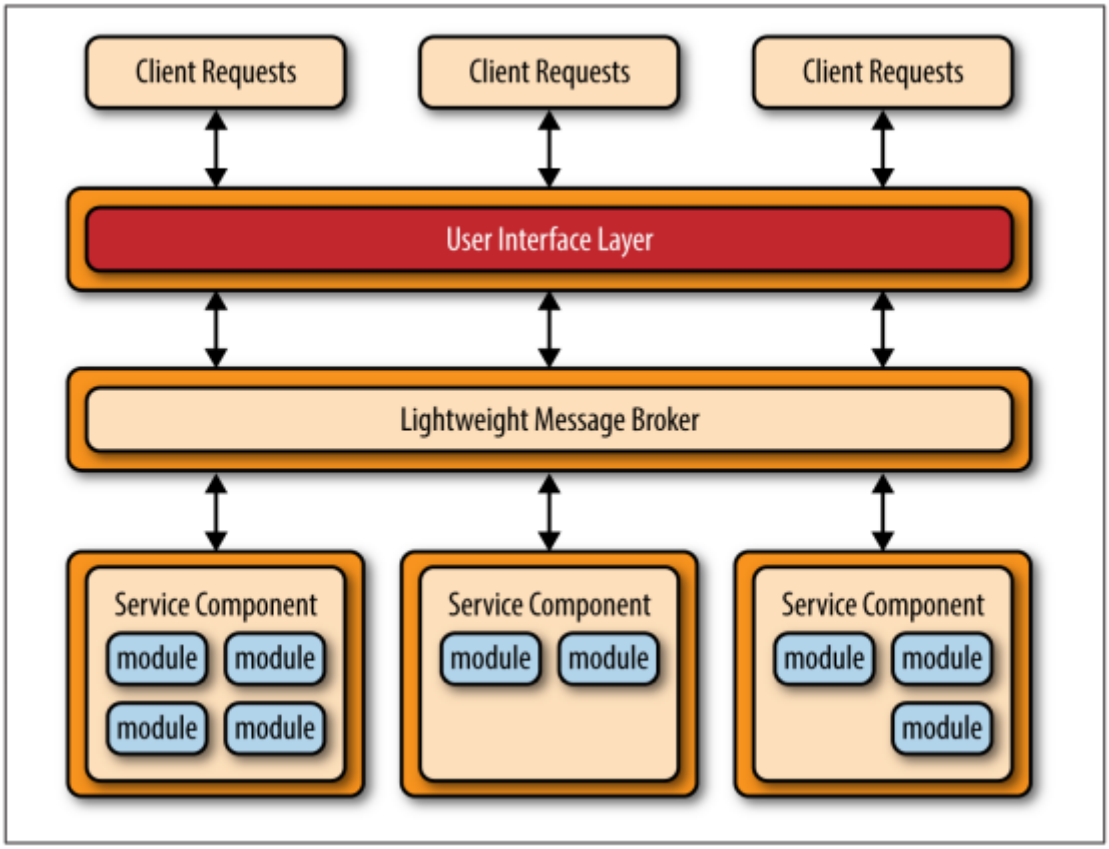
1. Overall Agility : High
    a. Due to division in Plug-in Component of services these component can also use in other project or product
2. Ease of Deployment : High
    a. application are divided into component which we just want to use in application so any change in component won’t effect the deployment
3. Testability : High
    a. Can be check each component separately coz they are independent
4. Performance : High
    a. you can customize easily and make performance better of each component separately
5. Scalability : High
    a. Product will be scalable because if we don’t have to use some or delete component just have to use/unuse it in project don’t have to delete the whole component
6. Ease of Development : High
    a. we just have to use another component in our project not copy paste the whole source code of that component

4. Micro-Service Architecture

The microservices architecture pattern solves many of the common
issues found in both monolithic applications as well as service oriented architectures.
Since major application components are
split up into smaller, separately deployed units, applications built
using the microservices architecture pattern are generally more
robust, provide better scalability, and can more easily support continuous delivery


Client Requests
Client Requests
Client Requests
User Interface Layer
Service Component
module
module
module
module
Service Component
module
module
Service Component
module
module
module


Client Requests
Client Requests
Client Requests
Application Programming Interface (api)
rest
rest
rest
rest
rest
Service Component
module
module
Service Component
module
Service Component
module
Service Component
module
module
Service Component
module



Pattern Analysis

1. Overall Agility : High
    a. Due to division in Services Component these component can just communicate other component with any user interface
2. Ease of Deployment : High
    a. Each service can be a separate application which has been deployed and any change in some component should not effect other deployed application
3. Testability : High
    a. Due to the separation and isolation of business functionality into independent applications, testing can be scoped
4. Performance : Low
    a. Each Component won't directly communicate with each other it must go through some gateway to request other component

5. Scalability : High
    a. Because the application is split into separately deployed units, each service component can be individual scaled, allowing for fine-tuned scaling of the application.
6. Ease of Development : High
    a. Because functionality is isolated into separate and distinct service components, development becomes easier due to the smaller and isolated scope.