Back To Module Home

Architecture of Scalable Applications

0% completed

Different Tiers in Software Architecture

Monolith and Microservices

Conclusion

Mark Module as Completed

N-Tier Applications

In this lesson, we will go over the n-tier applications and their components.

We'll cover the following

N-tier applications#

> An *n-tier* application is an application that has more than three components (*user interface, backend server, database*) involved in its architecture.

What are those components?

- Cache
- Message queues for asynchronous behavior
- Load balancers
- Search servers for searching through massive amounts of data

- Components involved in processing massive amounts of data
- Components running heterogeneous tech commonly known as web services, microservices, etc.

All the social applications like *Instagram, Facebook, TikTok*, large-scale consumer services like *Uber, Airbnb*, online massive multiplayer games like *Pokémon Go, Roblox*, etc., are *n-tier* applications. *n-tier* applications are more popularly known as *distributed systems*.

Now let's understand the need for so many *tiers*.

What's the need for so many tiers?#

Two software design principles that are key to explaining this are the *single responsibility principle* and the *separation of concerns*.

Let's understand what they are.

Single responsibility principle#

Single responsibility principle means giving one dedicated responsibility to a component and letting it execute it flawlessly, be it saving data, running the application logic or ensuring the delivery of the messages throughout the system.

This approach gives us a lot of flexibility, making management easy. We can have dedicated teams and code repositories for individual components keeping things cleaner.

With the *single responsibility principle*, the components are loosely coupled in terms of responsibility and making a change to one doesn't impact the functionality of other components. For instance, upgrading a *database server* with a new operating system or a patch won't affect other service components. Even if something amiss happens during the *OS* installation, just the *database* will go down. The application as a whole would still

be up and only the features requiring the *database* would be impacted.

The *single responsibility principle* is why I was never a fan of *stored procedures*.

Stored procedures enable us to add *business logic* to the *database*, which is always a big no. What if, in the future, we want to plug in a different *database*? Where do we take the *business logic*? Do we take it to the new *database*? Or do we refactor the application code and squeeze in the *stored procedure* logic somewhere?

A *database* should not hold *business logic*. It should only take care of persisting the data. This is what the *single responsibility principle* is, which is why we have separate *tiers* for separate components.

Separation Of concerns#

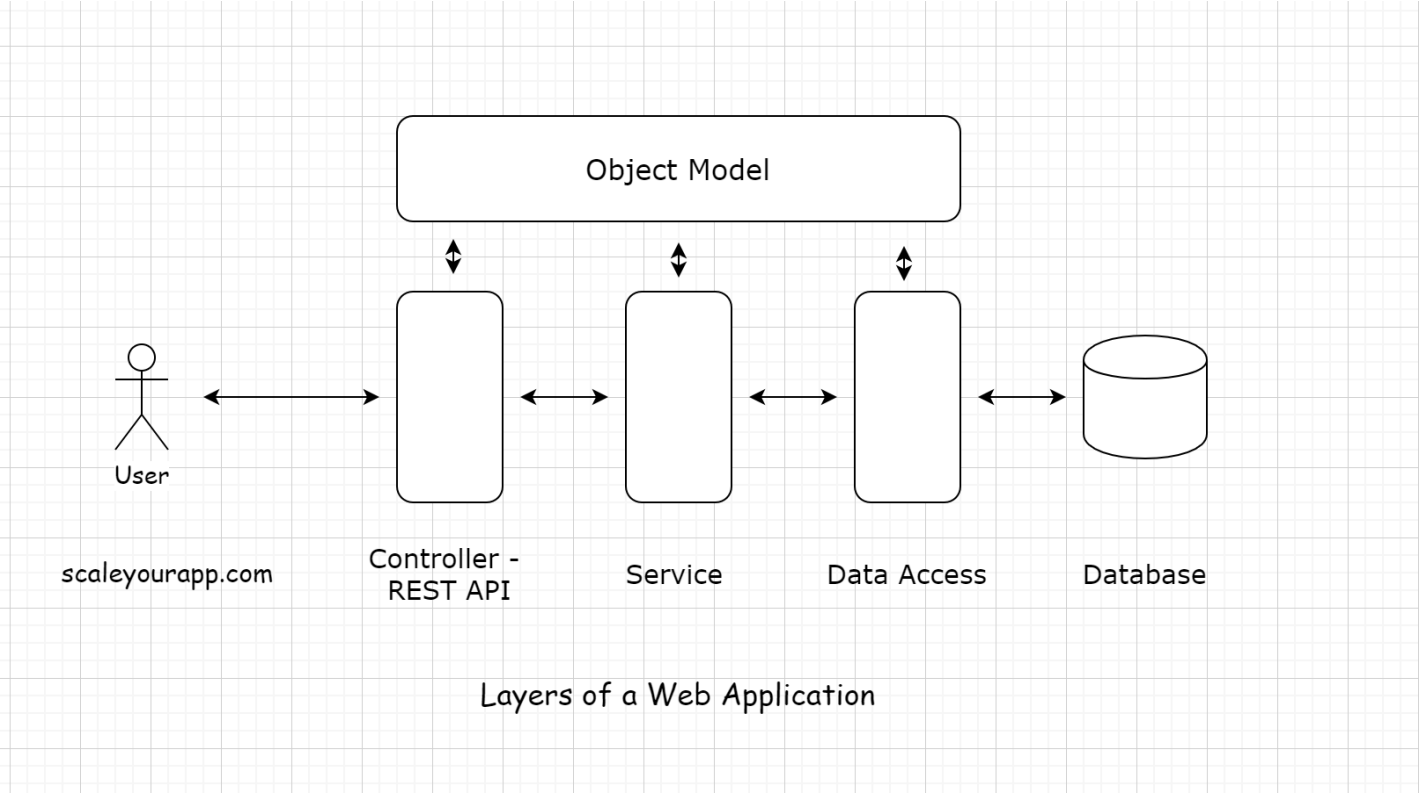
Separation of concerns loosely means the same thing, be concerned about your work only and stop worrying about the rest of the stuff. These principles act at all the levels of the service, be it the *tier* level or the *code* level.

Keeping the components separate makes them reusable. Different services can use the same *database, messaging server* or any other component as long as they are not tightly coupled with each other.

Having loosely coupled components is the way to go. This approach enables us to scale our service easily when things grow beyond a certain scale in the future.

Difference between layers & tiers#

> **Note:** Don't confuse *tiers* with the *layers* of the application. Some prefer to use them interchangeably. However, in the industry, application layers typically mean the *user interface, business, service* and the *data access* layers.



These layers are at the code level. The difference between *layers* and *tiers* is that *layers* represent the conceptual/logical organization of the code, whereas *tiers* represent the physical separation of components.

All these layers together can be used in any tiered application. Be it *single, two, three* or *n-tiered*. I'll discuss these layers in detail in the course ahead.

Alright, now we have an understanding of tiers. Let's zoom in one notch and focus on web architecture.

Three-Tier Applications

Next

Different Tiers in Software Architectur…

Mark as Completed

Report an Issue