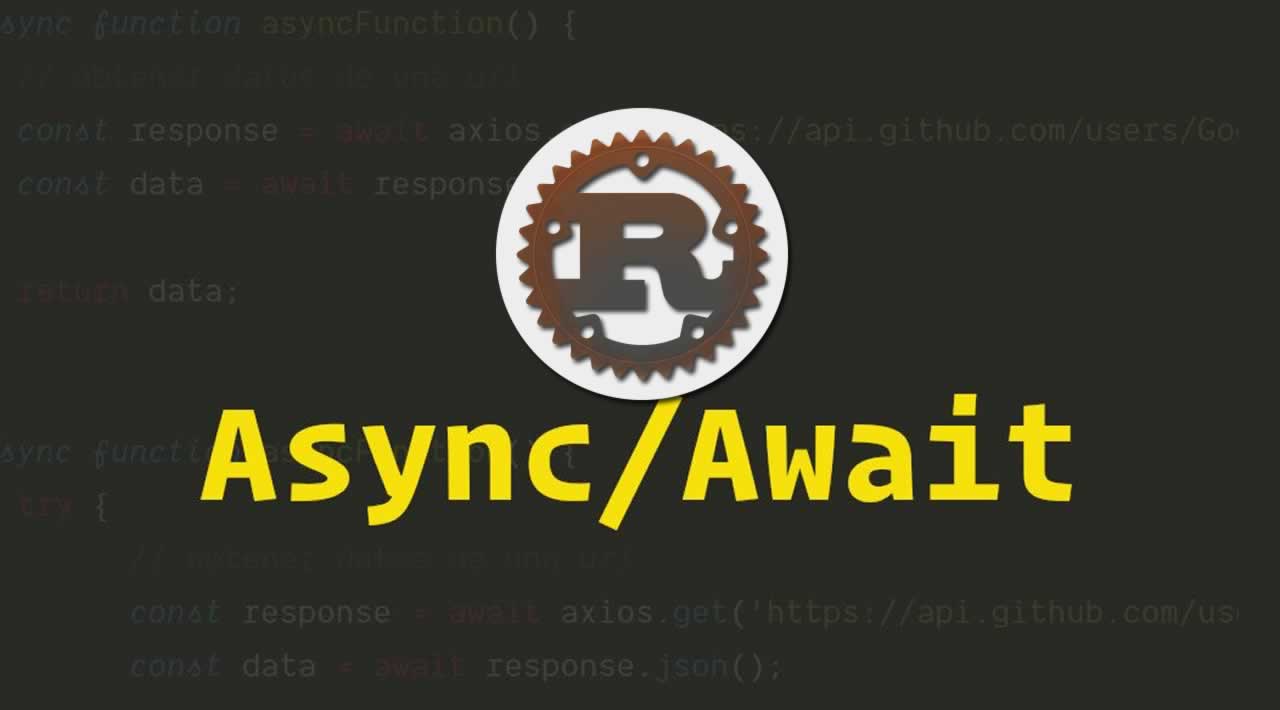
**An Article on Async Rust**



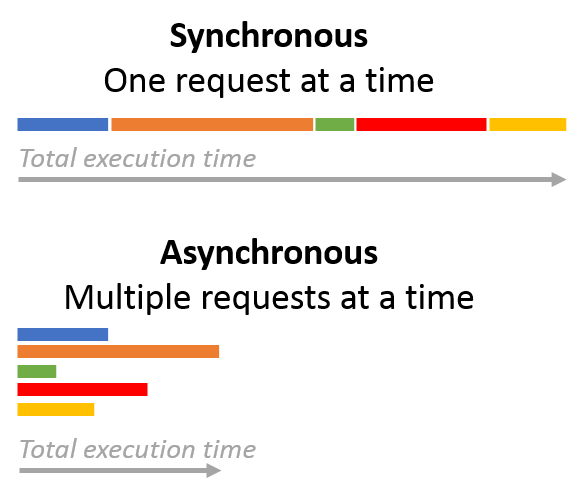
**what is async?**

Async is short for “asynchronous”. Async is a mean of running code concurrently. Also, it is meant to be multiple operations running in a time on a same OS thread.

In other words, Asynchronous programming is a parallel programming in which a unit work of an application run separately from the main application, and notifies to the calling thread after completion.

**Why Async?**

Async programming allows us to run multiple of these IO bound computations at a time on a single thread. Or we can say that, an asynchronous code let us to run several tasks simultaneously on the same OS thread.



**Async Rust**

Asynchronous coding in Rust Programming Language may be little bit different as it done in other programming languages like C# or JavaScript. In Async Rust, there is fearless concurrency while running multiple operations.

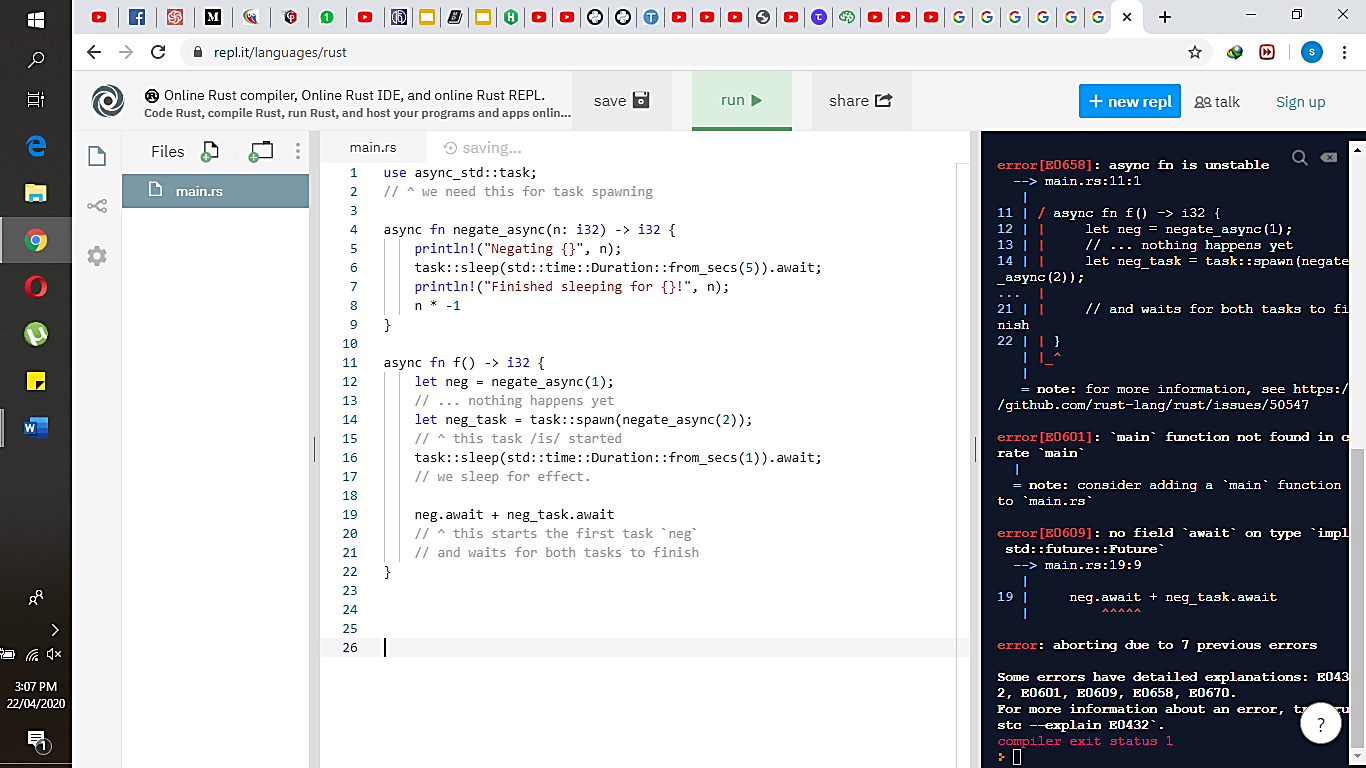
In Rust an Asynchronous function starts by .await or by launching a task using an executor.

But the standard library doesn’t come with an executer so, we need an external library to run futures. The executor takes care of executing the futures, polling them and returning the results after completion.

**What is meant by Future?**

In Rust, async fn creates an asynchronous function which returns a Future. To execute the body of the function, the returned Future must be run to completion. They aim to break code into small, composable actions that can be executed by a part of our system.

To understand it, these are few things that must be understood, so consider the following example.



In above Code:

The first line, async\_std::task .

The async function negate\_async takes a signed integer as an input, delays for five seconds and returns the number multiplied by -1.

In async function f, the first line makes a future of negate\_async function and assign this to the variable neg. The things to be noted that it does not start executing yet. Then in the next line, the task::spawn function to start the execution of Future returned by negate\_async and assign to the neg\_task variable. Then to make it obvious before start running the task, it sleeps for a second. In the last, await both the futures and return them after add them together. From neg.await it start executing the future and by neg\_task.await it just wait for it to finish because it has already been started.

The output will be:

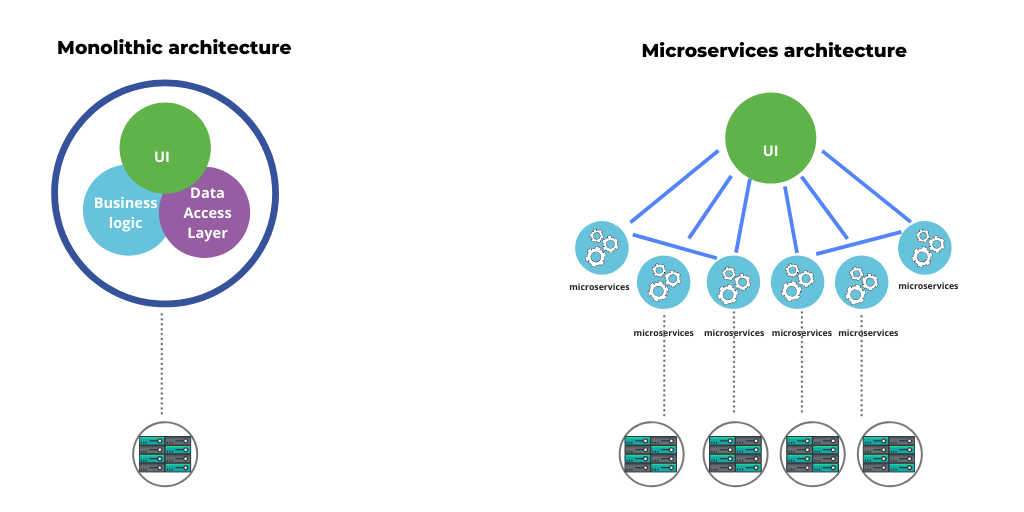
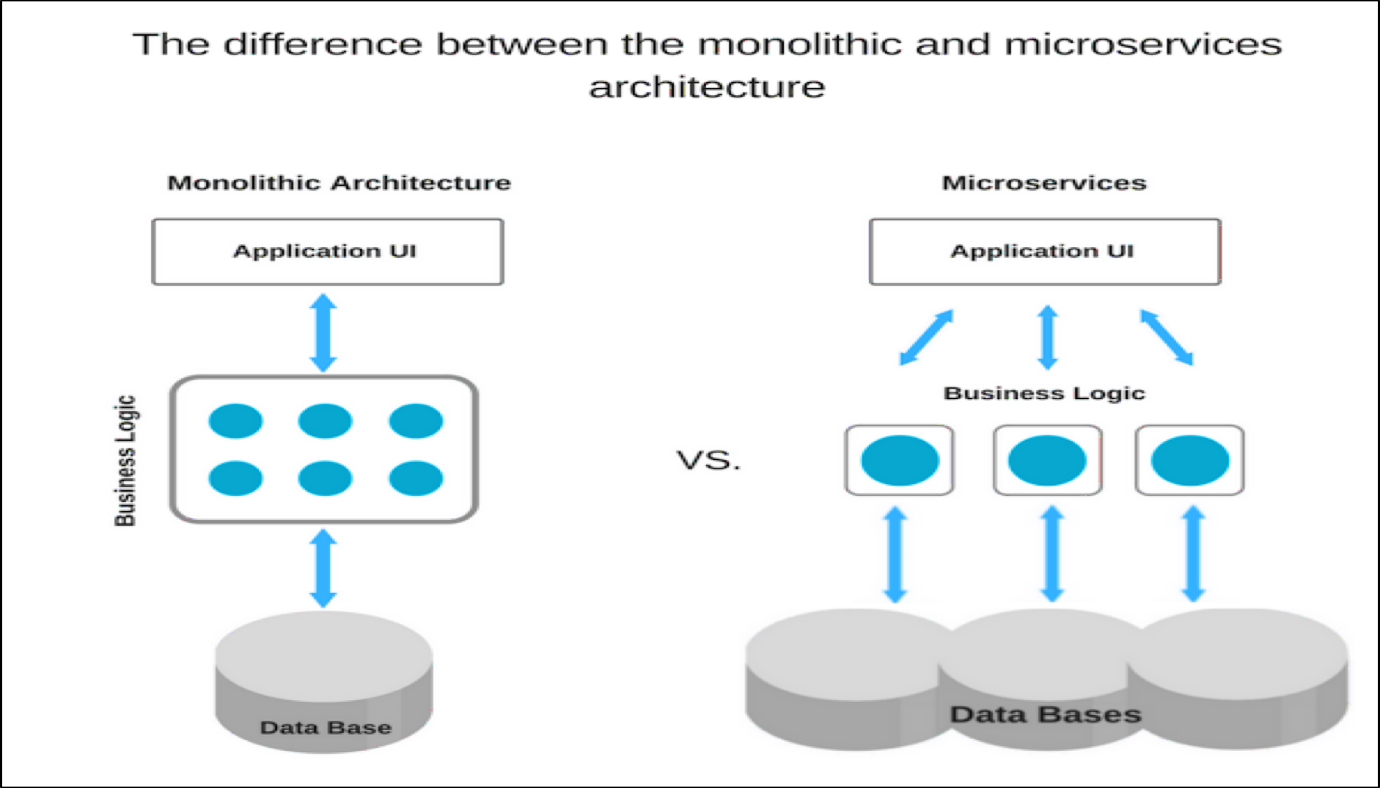
**Monolithic Architecture**

In Monolithic Architecture the entire software is composed of single piece that is designed to be self-contained. It means that all components of monolithic program are interconnected and interdependent where each component along with associated components must be present to execute the process of compilation of code.

There are some disadvantages of monolithic architecture that are defined below:

* As monolithic application grows in size, it become more difficult to do easy and frequent releases due to highly coupled components. Releases takes more time and people to plane. Also, frequent release is risked of breaking the application due to newly release feature.
* In case of larger monolithic application, the deployment time becomes slow and long. For a single change, the entire application needs to be redeployed and this become difficult to frequent deployment and thus obstacle for continuous delivery. So, in order to add new features every time in an application, this is a serious issue.
* It is difficult to manage project and team because It takes a huge time and effort to plan the release and manage tightly coupled interdependent modular development.
* It becomes very expensive to scale a monolithic application.
* It is very difficult to replace a component with another better designed component without effecting the whole application architecture.

**Microservices Architecture**



Microservices architecture expresses the setup where application components are standalone application of their own.

The following are characteristics of Microservices architecture:

* Microservices can offer scaling within seconds and can also integrate with third-party services.
* In Microservices architecture different components can be implemented and programmed in different programming language.
* Application can be developed and deploy independently. So, project management become easier.
* A microservice can be deployed by a small team easily.
* In Microservices architecture, required changes made into the particular component, so the entire application does not affect and need to be redeploy.

Along with the advantages, there are some disadvantages of microservices architecture. These are as under:

* In microservices architecture testing and debugging can become complicated because of distributed deployment.
* As the whole application is decomposed into several microservices, it becomes more complicated to handle operation of many components rather than a single component.
* Because of several microservices, it is quite challenging to secure application.

**Conclusion**

Designing and implementing microservices architecture is challenging as compared to monolithic architecture but microservices is better choice for a complex and evolving application. Microservices are useful when the application is very huge and need to be independent for each service. Also, any change can not affect the whole application. It is very useful and handy tool for modern software development. Microservices architecture is very efficient for team to move with new features without disrupting the whole software activities.

Thanks for reading..