Back To Module Home

# Architecture of Scalable Applications

0% completed



## Different Tiers in Software Architecture



## Monolith and Microservices

## Conclusion

Mark Module as Completed

# Single-Tier Applications

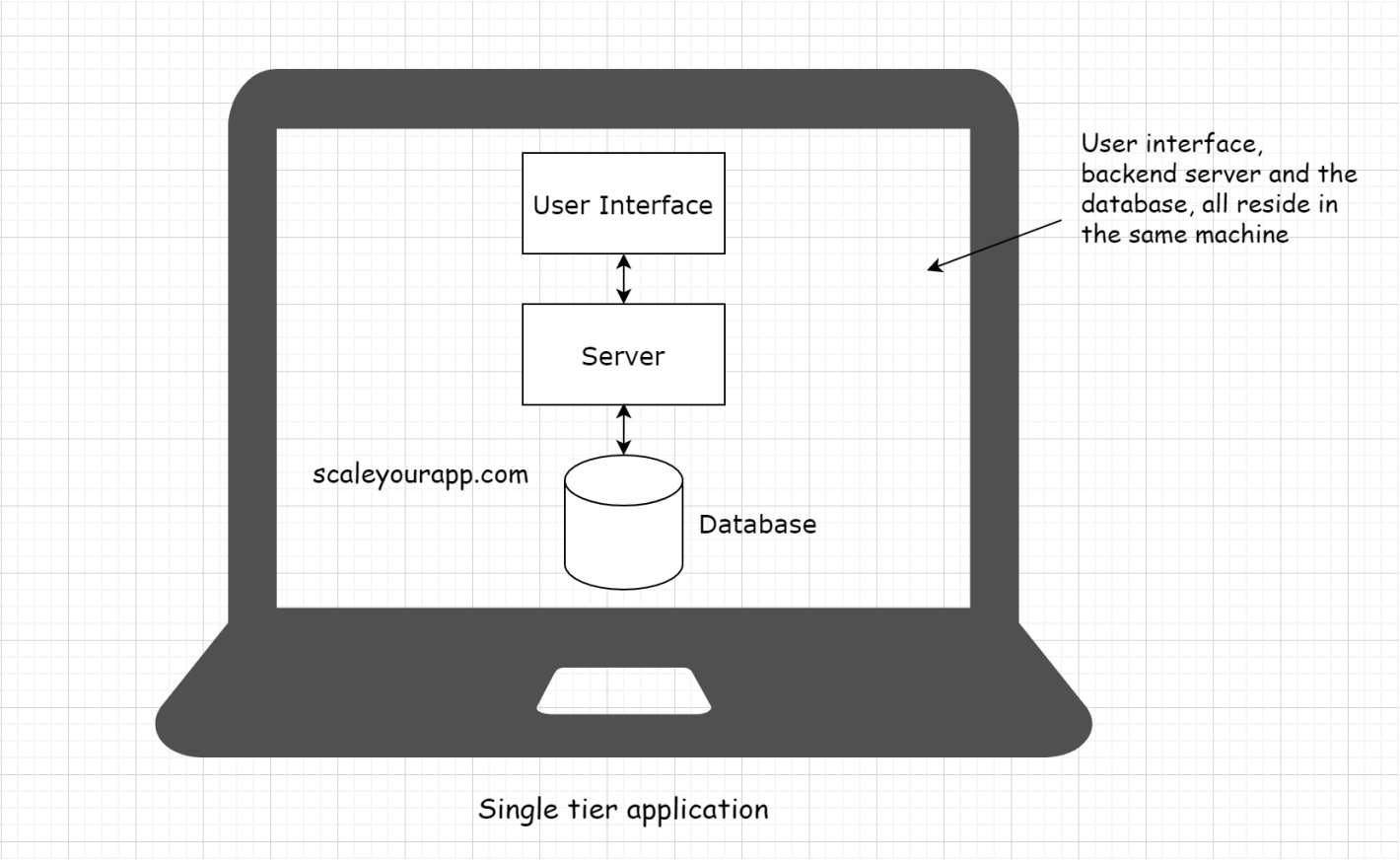
In this lesson, you will learn about single-tier applications.

**We'll cover the following** 



## Single-tier applications#

> In a *single-tier* application, the user interface, backend business logic, and the database reside in the same machine.




Typical examples of *single-tier* applications are desktop applications like *MS Office*, PC Games, image editing software like *Gimp, Photoshop*, etc.

# Upsides of single-tier applications#

The primary upside of *single-tier* applications is that they have no network latency since every component is located on the same machine. This adds up to the performance of the software.

*Two, three* and *n-tier* apps have to send data requests to the backend server often. This adds network latency to the system making the user experience slow compared to *single-tier* apps. In *single-tier* apps, the data is readily available since all the components are located in the same machine.

However, the actual performance of a single-tier app largely depends on the application's hardware requirements and how powerful the machine it runs on is.

Also, when it comes to data privacy and safety, it is of the highest order in *single-tier*

apps since the user's data always stays in their machine and doesn't need to be transmitted over a network for persistence.

# Downsides of single-tier applications#

One big downside of *single-tier* apps is that the application's publisher has no control over the application. Once the software is shipped, no code or feature updates can be made until the customer manually updates it by connecting to the remote server or downloading and installing a patch.

Due to this, in the *90s*, there was nothing that the studios could do if their game was shipped with buggy code. They eventually had to face a lot of heat from the gamers due to the buggy nature of their software. This made product testing vital, responsible for making or breaking a business. Software testing had to be thorough since there was no room for any mistakes.

The code in *single-tier* applications is also vulnerable to being tweaked and reversed engineered. The product security for the app publisher is minimal. An evil person with some effort can get access to the application's source code, modifying or copying it for profit. This is unlikely in an architecture where the company controls the application server and implements security to fend off the hackers.

Finally, a *single-tier* applications' performance and look and feel can be inconsistent as the app rendering largely depends on the configuration of the user's machine.

Introduction

Next

Two-Tier Applications

Mark as Completed

Report an Issue