Web Application Layout

No two web applications are identical. Businesses create web applications for a multitude of uses and audiences. Web applications are designed and programmed differently, and back end infrastructure can be set up in many different ways. It is important to understand the various ways web applications can run behind the scenes, the structure of a web application, its components, and how they can be set up within a company's infrastructure.

Web application layouts consist of many different layers that can be summarized with the following three main categories:

Category	Description
`Web Application Infrastructure`	Describes the structure of required components, such as the database, needed for the web application to function as intended. Since the web application can be set up to run on a separate server, it is essential to know which database server it needs to access.
`Web Application Components`	The components that make up a web application represent all the components that the web application interacts with. These are divided into the following three areas: `UI/UX`, `Client`, and `Server` components.
`Web Application Architecture`	Architecture comprises all the relationships between the various web application components.

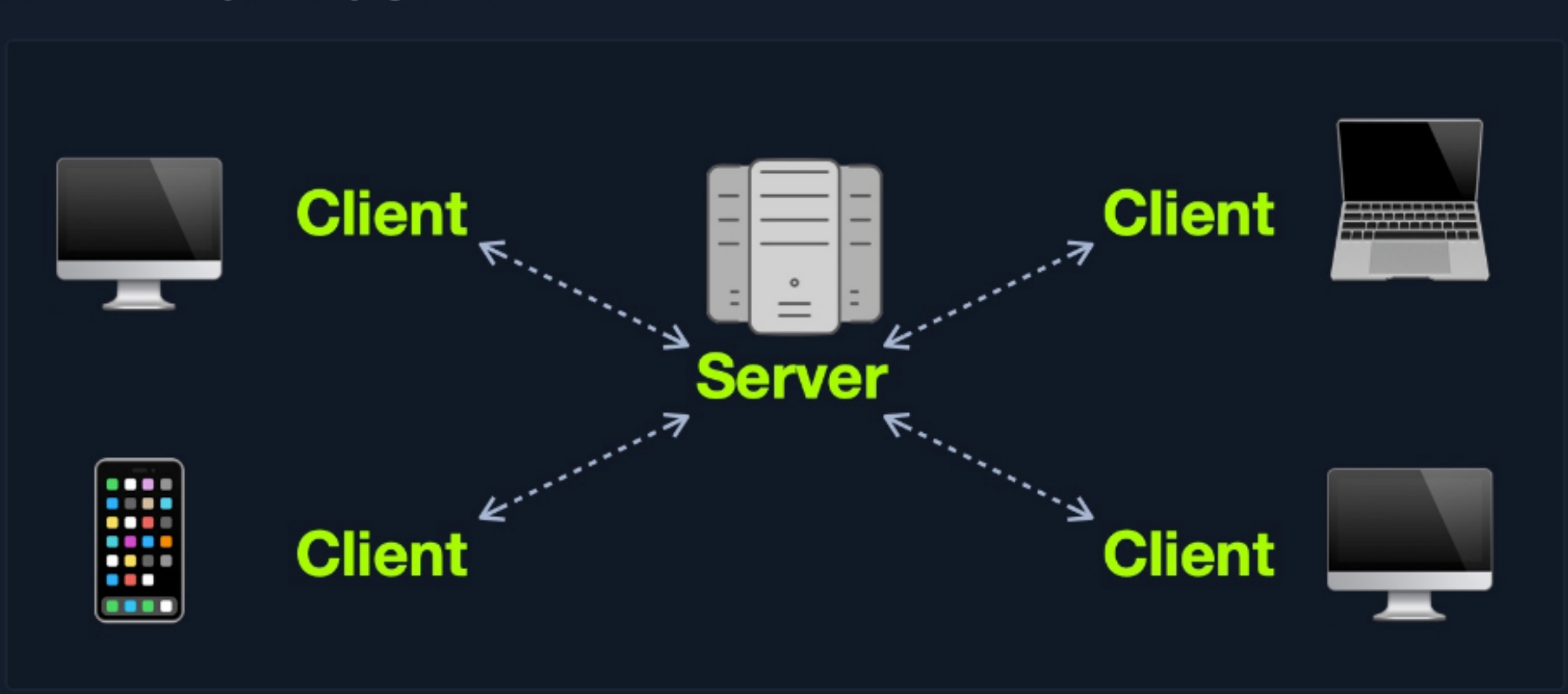
Web Application Infrastructure

Web applications can use many different infrastructure setups. These are also called `models`. The most common ones can be grouped into the following four types:

- `Client-Server`
- `One Server`
- `Many Servers - One Database`
- `Many Servers - Many Databases`

Client-Server

Web applications often adopt the `client-server` model. A server hosts the web application in a client-server model and distributes it to any clients trying to access it.



In this model, web applications have two types of components, those in the front end, which are usually interpreted and executed on the client-side (browser), and components in the back end, usually compiled, interpreted, and executed by the hosting server.

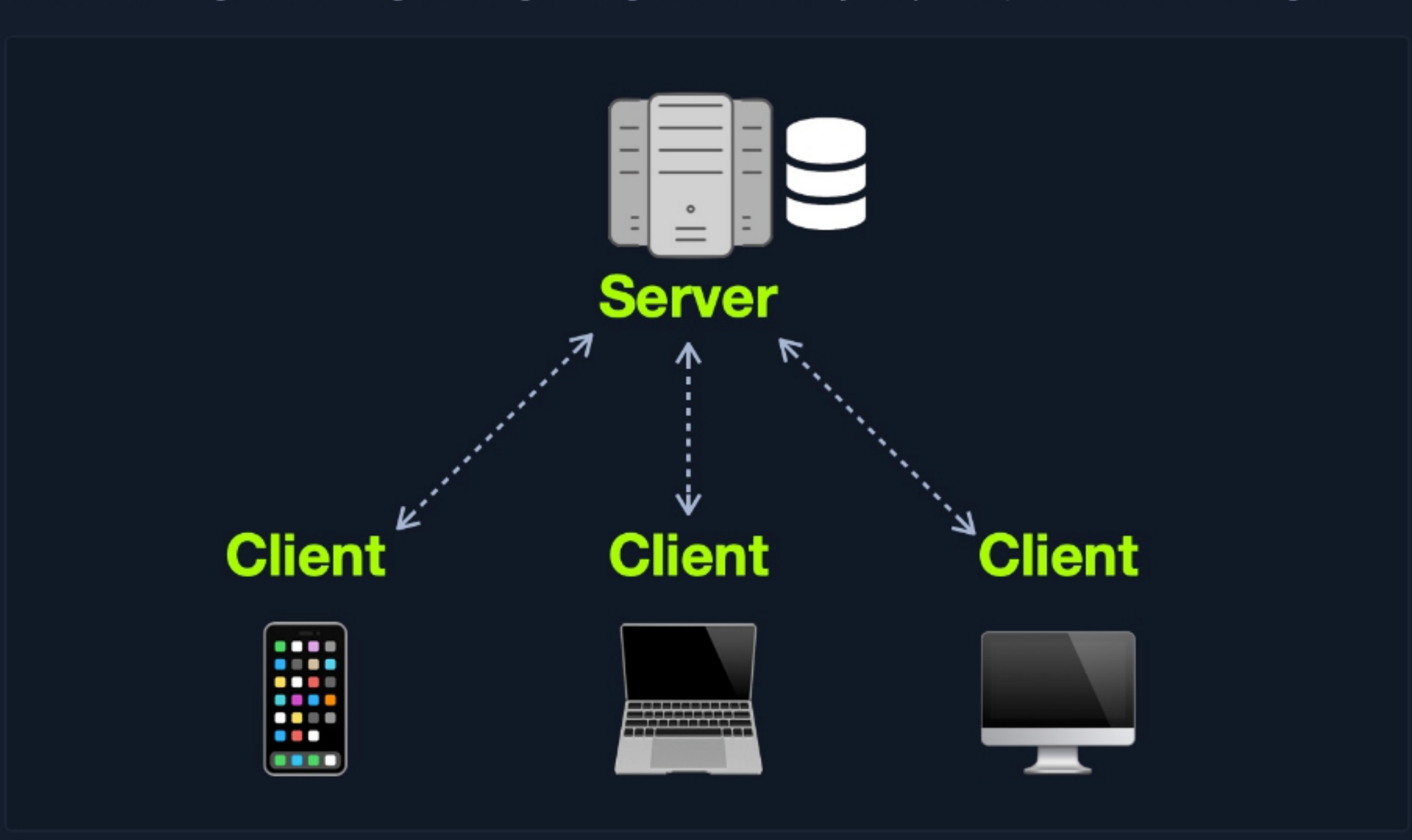
When a client visits the web application's URL (web address, i.e., https://www.acme.local), the server uses the main web application interface (`UI`). Once the user clicks on a button or requests a specific function, the browser sends an HTTP web request to the server, which interprets this request and performs the necessary task(s) to complete the request (i.e., logging the user in, adding an item to the shopping cart, browsing to another page, etc.). Once the server has the required data, it sends the result back to the client's browser, displaying the result in a human-readable way.

`This very website we are interacting with right now and are reading is also a web application, developed and hosted by Hack The Box (webserver), and being used and interacted with by us via our web browser (client).`

However, even though most web applications utilize a client-server front-back end architecture, there are many design implementations.

One Server

In this architecture, the entire web application or even several web applications and their components, including the database, are hosted on a single server. Though this design is straightforward and easy to implement, it is also the riskiest design.

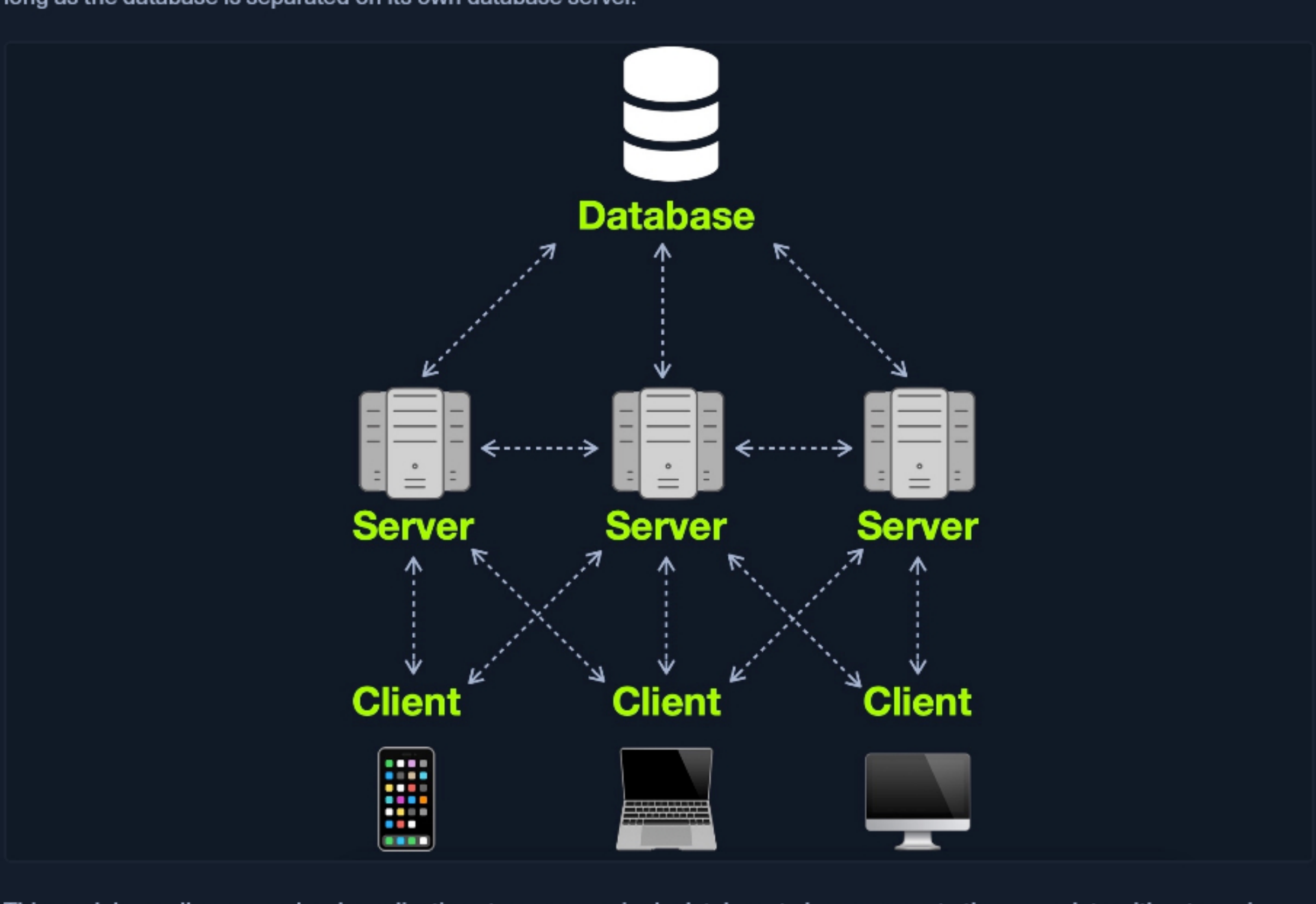


If any web application hosted on this server is compromised in this architecture, then all web applications' data will be compromised. This design represents an "`all eggs in one basket`" approach since if any of the hosted web applications are vulnerable, the entire webserver becomes vulnerable.

Furthermore, if the webserver goes down for any reason, all hosted web applications become entirely inaccessible until the issue is resolved.

Many Servers - One Database

This model separates the database onto its own database server and allows the web applications' hosting server to access the database server to store and retrieve data. It can be seen as many-servers to one-database and one-server to one-database, as long as the database is separated on its own database server.

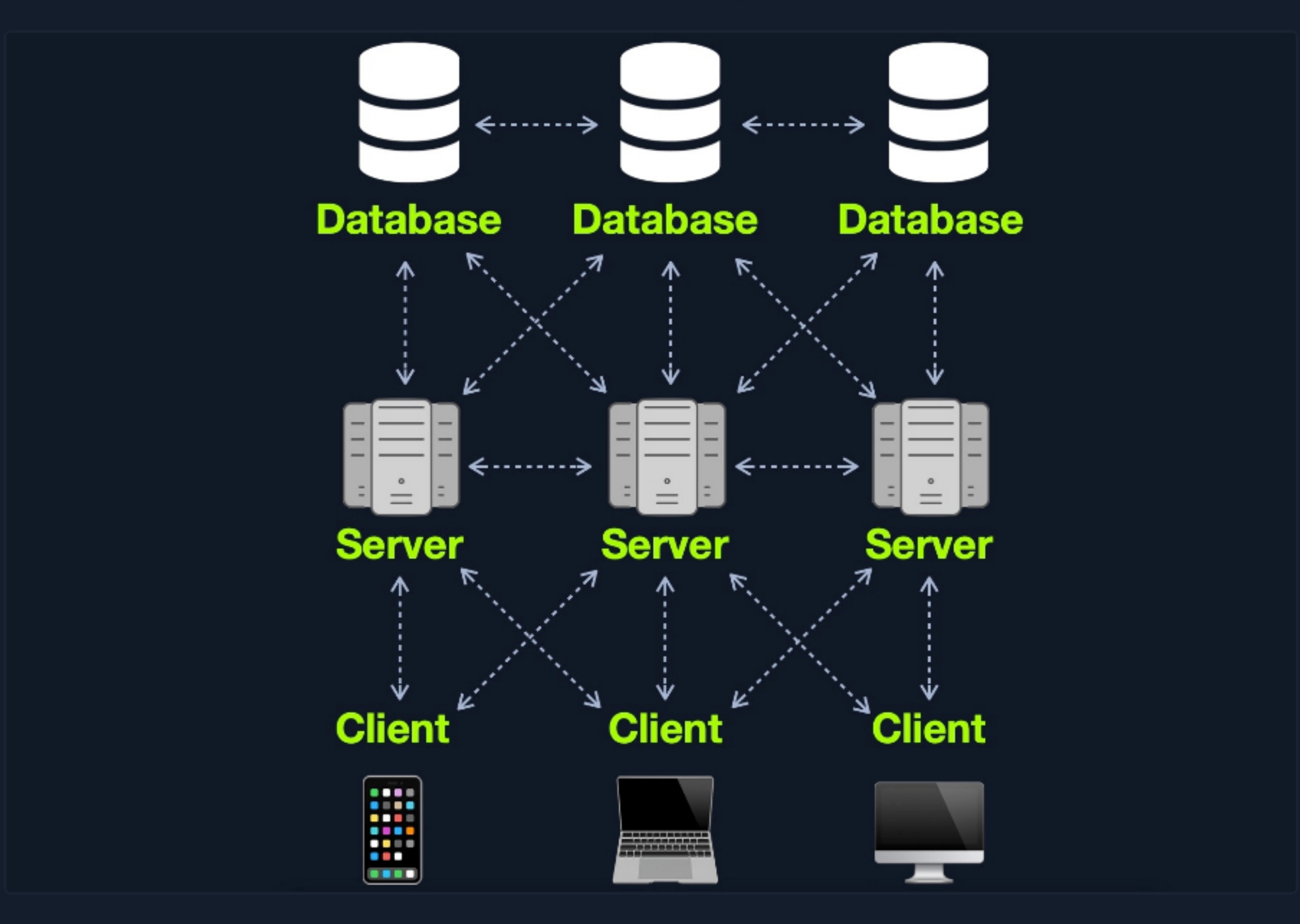


This model can allow several web applications to access a single database to have access to the same data without syncing the data between them. The web applications can be replicates of one main application (i.e., primary/backup), or they can be separate web applications that share common data.

This model's main advantage (`from a security point of view`) is 'segmentation,' where each of the main components of a web application is located and hosted separately. In case one webserver is compromised, other webservers are not directly affected. Similarly, if the database is compromised (i.e., through a SQL injection vulnerability), the web application itself is not directly affected. There are still access control measures that need to be implemented after asset segmentation, such as limiting web application access to only data needed to function as intended.

Many Servers - Many Databases

This model builds upon the `Many Servers, One Database` model. However, within the database server, each web application's data is hosted in a separate database. The web application can only access private data and only common data that is shared across web applications. It is also possible to host each web application's database on its separate database server.



This design is also widely used for redundancy purposes, so if any web server or database goes offline, a backup will run in its place to reduce downtime as much as possible. Although this may be more difficult to implement and may require tools like load balancers to function appropriately, this architecture is one of the best choices in terms of security due to its proper access control measures and proper asset segmentation.

Aside from these models, there are other web application models available such as serverless web applications or web applications that utilize microservices.

Web Application Components

Each web application can have a different number of components. Nevertheless, all of the components of the models mentioned previously can be broken down to:

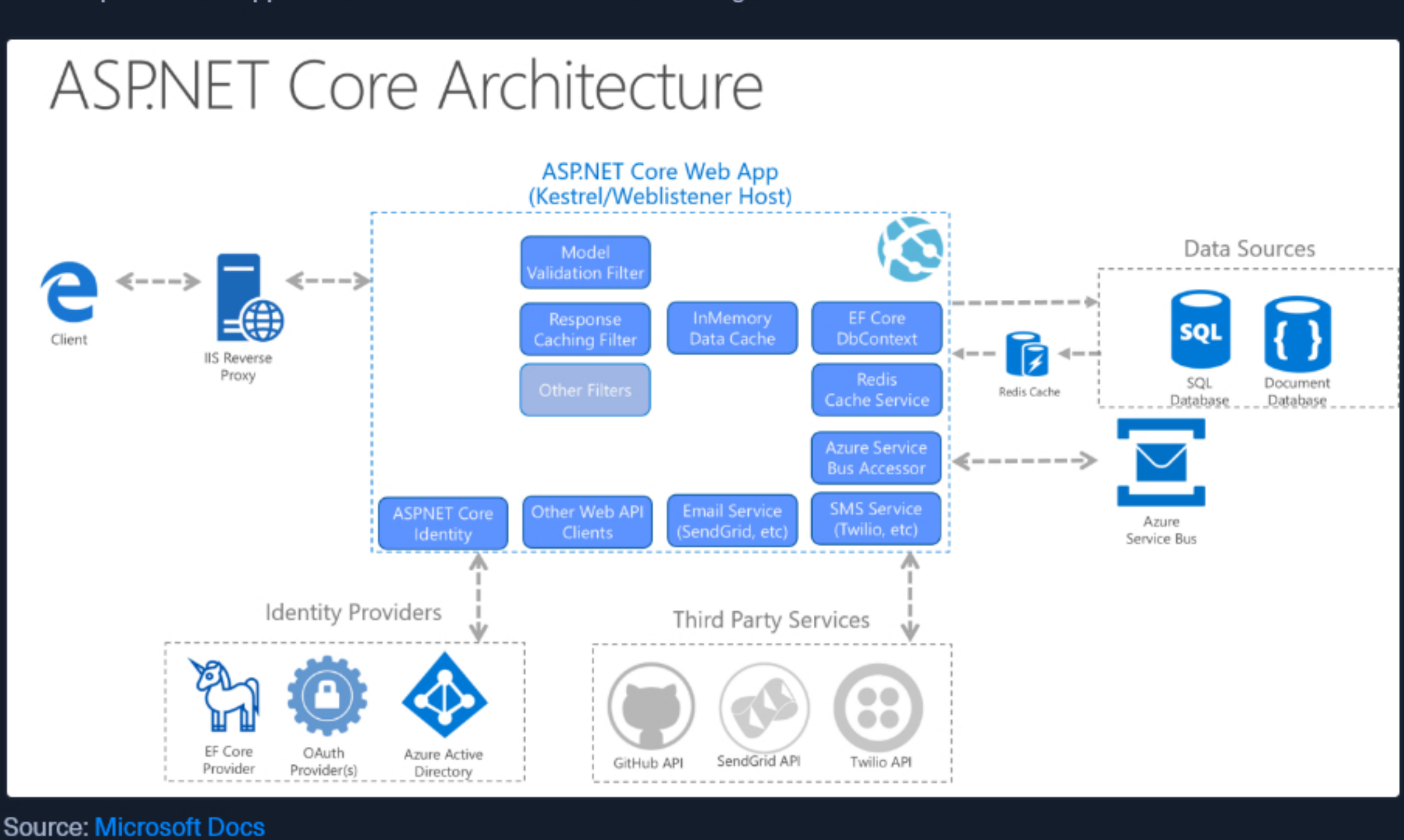
1. `Client`
2. `Server`
   - `Webserver`
   - `Web Application Logic`
   - `Database`
3. `Services (Microservices)`
   - `3rd Party Integrations`
   - `Web Application Integrations`
4. `Functions (Serverless)`

Web Application Architecture

The components of a web application are divided into three different layers.

Layer	Description
`Presentation Layer`	Consists of UI process components that enable communication with the application and the system. These can be accessed by the client via the web browser and are returned in the form of HTML, JavaScript, and CSS.
`Application Layer`	This layer ensures that all client requests (web requests) are correctly processed. Various criteria are checked, such as authorization, privileges, and data passed on to the client.
`Data Layer`	The data layer works closely with the application layer to determine exactly where the required data is stored and can be accessed.

An example of a web application architecture could look something like this:



Source: Microsoft Docs

Microservices

We can think of microservices as independent components of the web application, which in most cases are programmed for one task only. For example, for an online store, we can decompose core tasks into the following components:

- `Registration`
- `Search`
- `Payments`
- `Ratings`
- `Reviews`

These components communicate with the client and with each other. The communication between these microservices is `stateless`, which means that the request and response are independent. This is because the stored data is `stored separately` from the respective microservices. The use of microservices is considered service-oriented architecture (SOA), built as a collection of different automated functions focused on a single business goal. Nevertheless, these microservices depend on each other.

Another essential and efficient microservice component is that they can be written in different programming languages and still interact. Microservices benefit from easier scaling and faster development of applications, which encourages innovation and speeds upmarket delivery of new features. Some benefits of microservices include:

- `Agility`
- `Flexible scaling`
- `Easy deployment`
- `Reusable code`
- `Resilence`

This AWS whitepaper provides an excellent overview of microservice implementation.

Serverless

Cloud providers such as AWS, GCP, Azure, among others, offer serverless architectures. These platforms provide application frameworks to build such web applications without having to worry about the servers themselves. These web applications then run in stateless computing containers (Docker, for example). This type of architecture gives a company the flexibility to build and deploy applications and services without having to manage infrastructure; all server management is done by the cloud provider, which gets rid of the need to provision, scale, and maintain servers needed to run applications and databases.

You can read more about serverless computing and its various use cases here.

Architecture Security

Understanding the general architecture of web applications and each web application's specific design is important when performing a penetration test on any web application. In many cases, an individual web application's vulnerability may not necessarily be caused by a programming error but by a design error in its architecture.

For example, an individual web application may have all of its core functionality secure implemented. However, due to a lack of proper access control measures in its design, i.e., use of Role-Based Access Control(RBAC), users may be able to access some admin features that are not intended to be directly accessible to them or even access other user's private information without having the privileges to do so. To fix this type of issue, a significant design change would need to be implemented, which would likely be both costly and time-consuming.

Another example would be if we cannot find the database after exploiting a vulnerability and gaining control over the back-end server, which may mean that the database is hosted on a separate server. We may only find part of the database data, which may mean there are several databases in use. This is why security must be considered at each phase of web application development, and penetration tests must be carried throughout the web application development lifecycle.

← Previous Next →

Mark Complete & Next

Table of Contents

My Workstation

OFFLINE

Start Instance

∞ / 1 spawns left