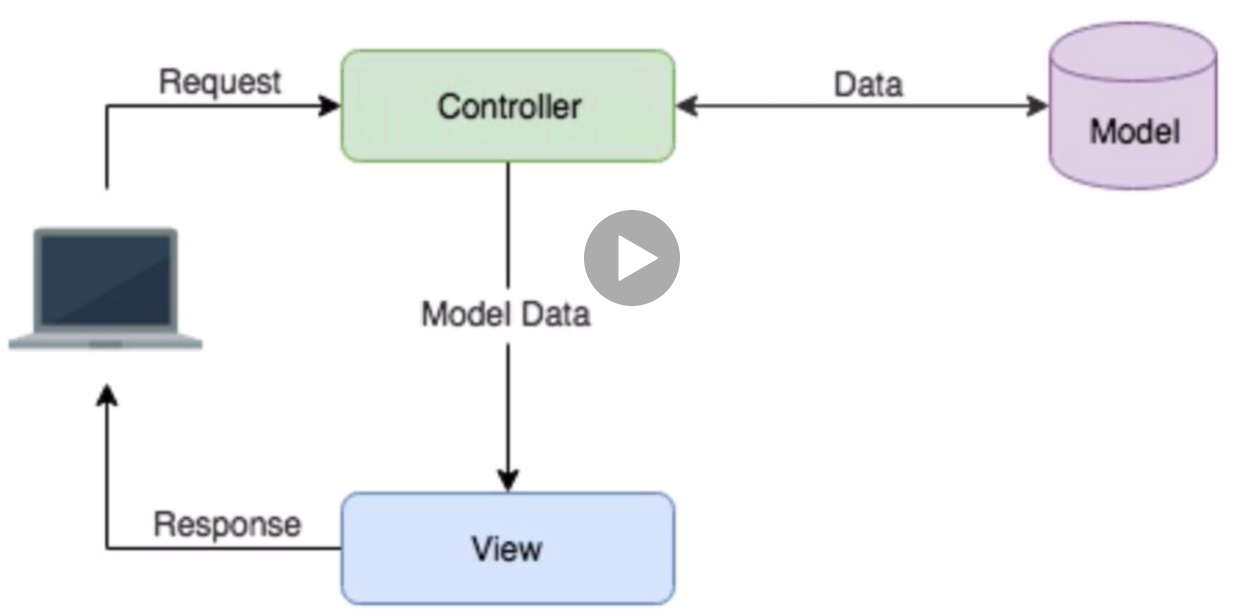
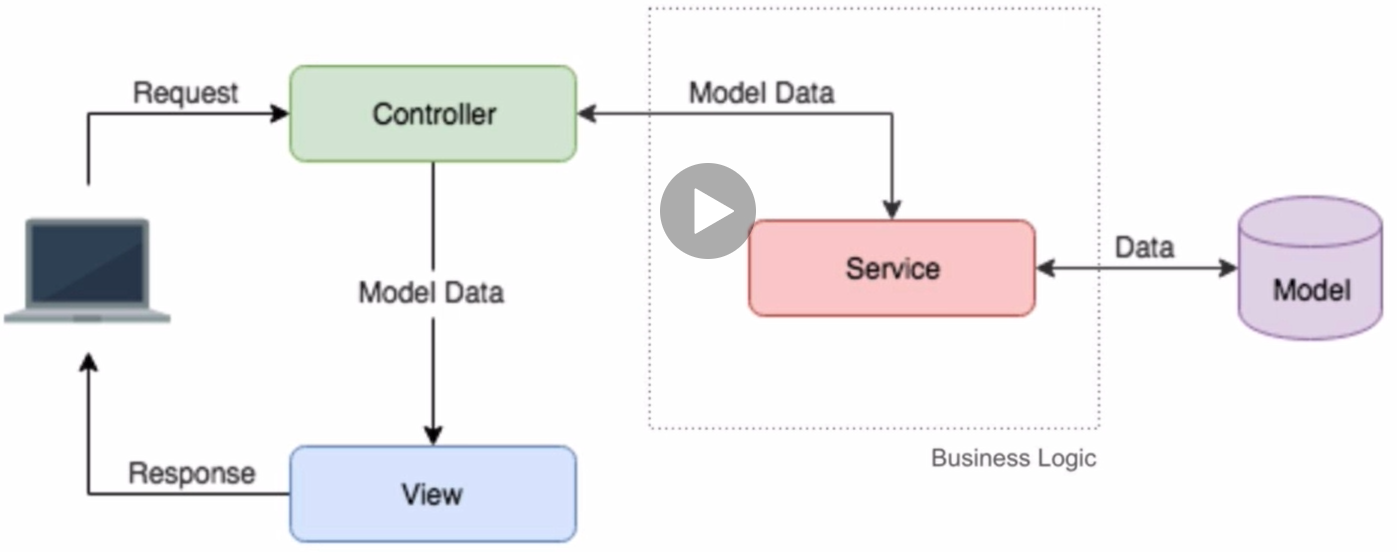
# MVC Pattern



1. **Client** sends a request against a **controller** by using a mapping, where the **controller** is the entry point for everything in a web app.
2. The **controller** interacts with the model to get the data from, or save it into the DB
3. And we basically build a **model data** that we send to a view and this view is what is getting back to the client.

But this is **limited**, as for every new access we would need a new controller layer to access the data. That’s why the following approach is implemented:



1. **Client** sends a request against a **controller** by using a mapping, where the **controller** is the entry point for everything in a web app.
2. The **controller** interacts with a **service** layer in order to request the modelled data. So the **service** is now in charge of interacting with the data model and performing any model operation

So now all our business logic is inside a service layer, we don’t have any business logic inside **controller** layer.

## Controllers

**Controllers** main characteristics are:

* They are the entry point of any web application.
* They’re accessed via URL mappings.
* They only validate that the incoming request has all the needed parameters.
* They DO NOT HOLD any business logic **(IMPORTANT, I THINK SOME LOGIC IS PLACED IN OUR CONTROLLERS??).**
* They trust on **services** to process each new request.
* They return the response to the client without adding any additional data.

## Services

**Services** main characteristics are:

* They contain the entire business logic of our application.
* Each service is responsible of handling a particular and unique entity (UsersService, ItemsService, etc.).
* They are **stateless** 🡪 They DO NOT share any data between different requests
* They are usually **singletons** 🡪 Because since they are stateless, they don’t need to be holding any information.
* They invoke other **services, models, external providers** and any other needed data source.
* They handle **errors, send metrics, send logs, tags** and any other support metric needed by our application.

## Model / Domain / DAOs (Data Access Objects)

* They hold the core domains of our services. Any other layer exists **to support and serve** these domain objects.
* They are in charge of defining the structure of domain objects.
* This is the only layer where we know about **persistence**. Only they know **where and how** they should persist.
* They are in charge of abstracting persistence logic by creating a clean and general interface

# Package Organization