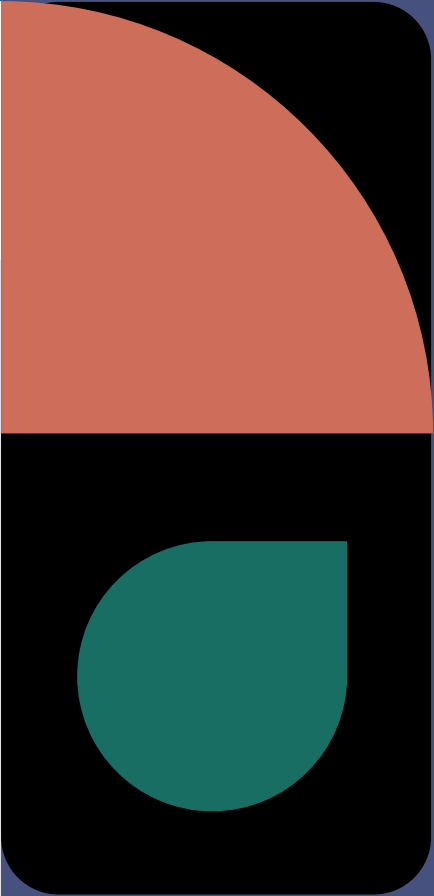
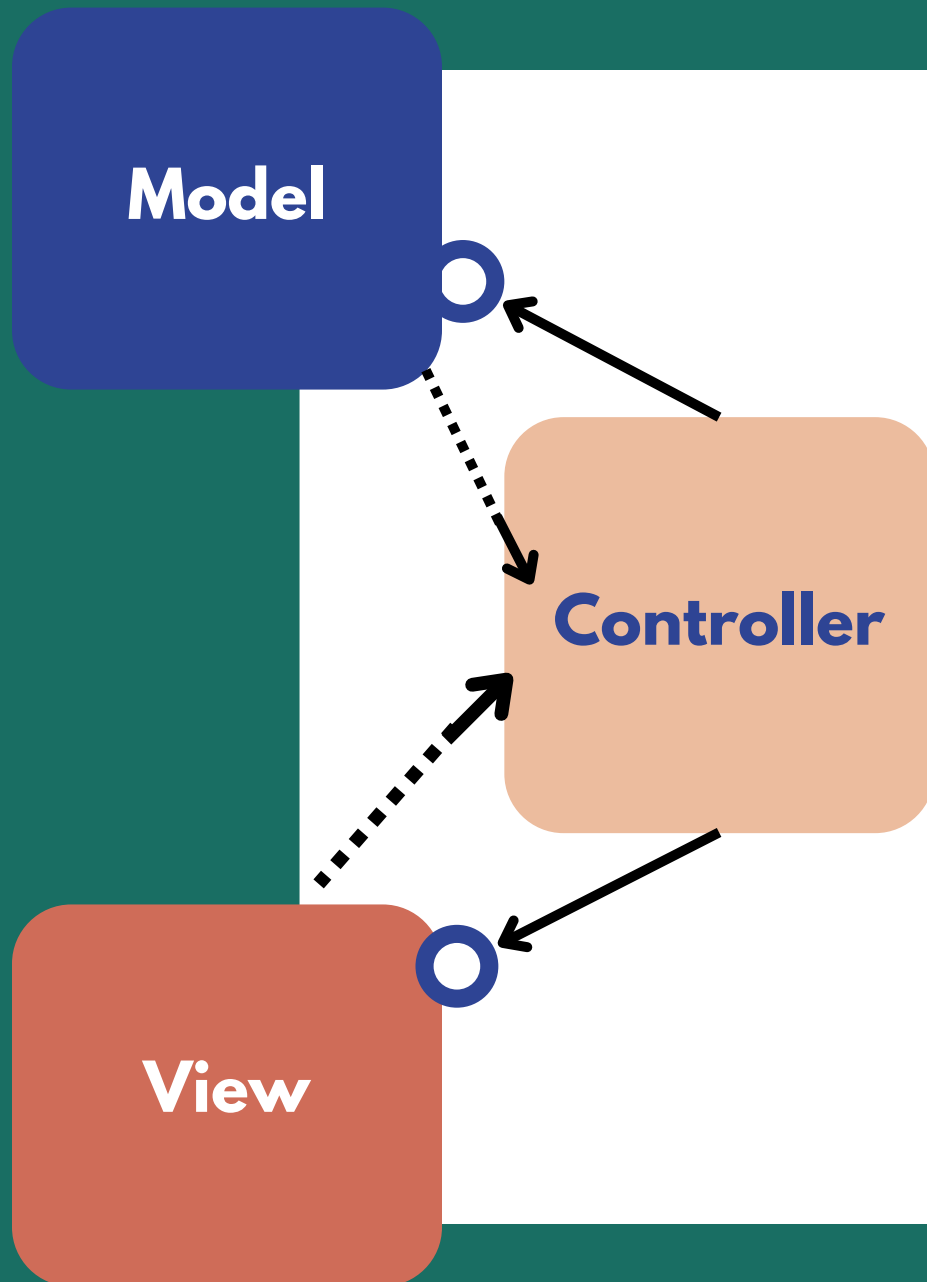




Model View Controller

Researched and Presented By
Micah Elm





What the F\$%^ is going on??

We will get to that. This will be the image we bring with us on this on this journey to understand MVC

MVC is an architecture pattern or design pattern, depending on whom you are talking with, what matter is that this is about **Separation of Concerns, Modularity, Abstraction, Maintainability, Structure, and Scalability.**

model - view - controller

Model :

Business Logic; Creating new documents(data) in DB, checking validation(user input data), or simply just processing data to or from the database. Speaks only to the Controller (Middleman).

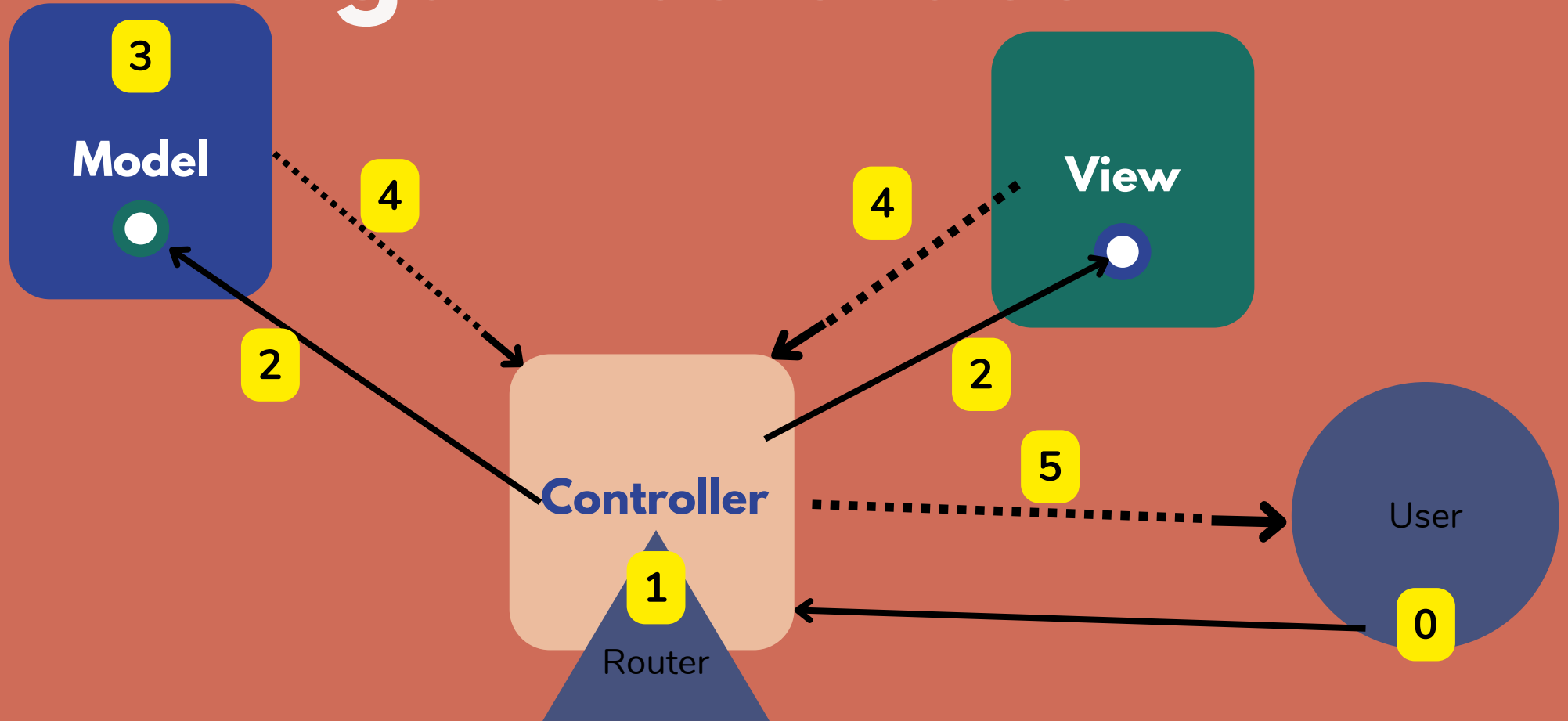
View:

Presentation Logic; This is the only thing the user ever sees. Speaks only with the Controller --- doesn't really talk to the controller it Listens to the controller. The controller tells the view what to do.

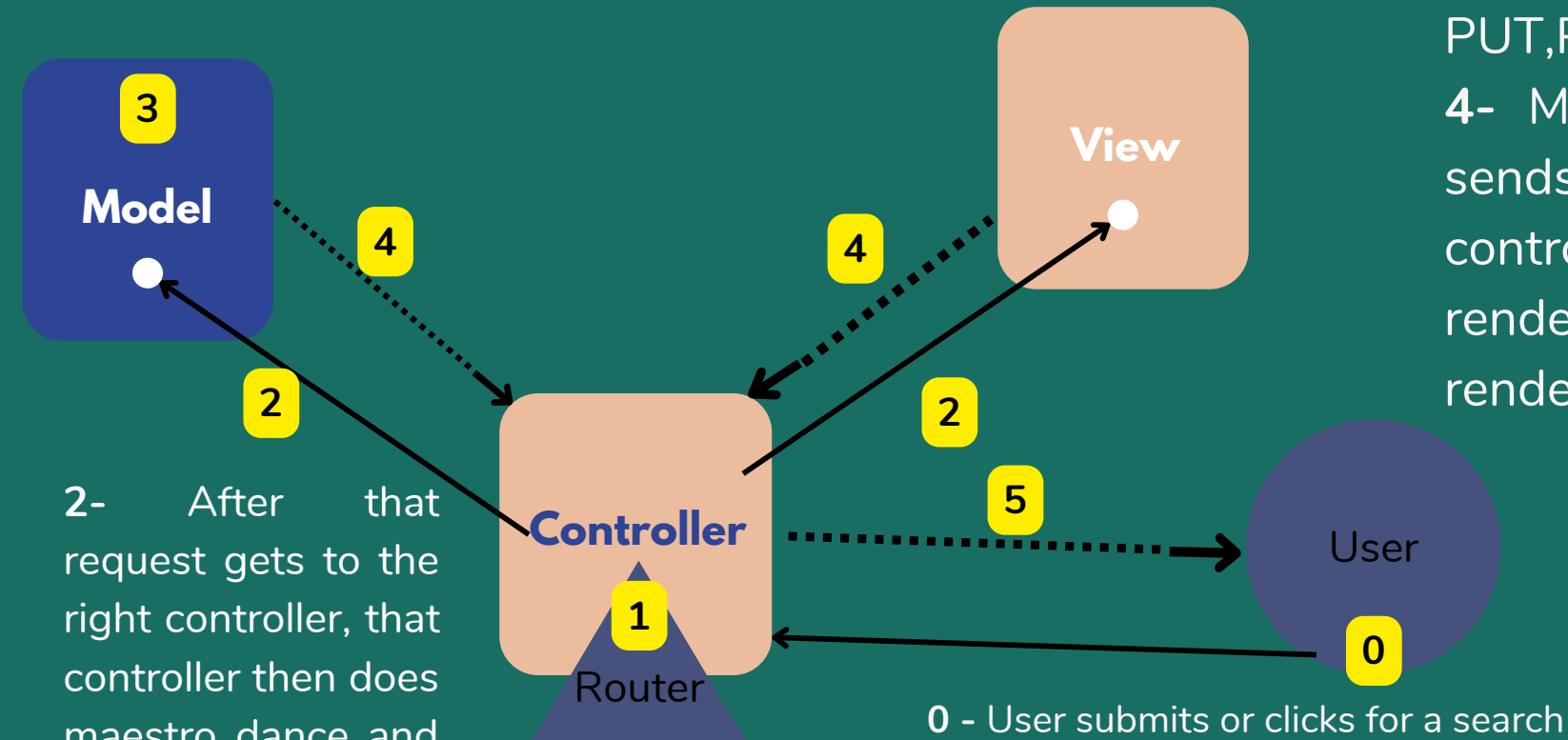
Controller:

Application Logic; The Middleman. The user sends a request the controller is what receives it and determines where it will go. Because it is the Middleman it does not allow the view and the model to talk to each other.

Organized chaos



Let's break it down



2- After that request gets to the right controller, that controller then does maestro dance and sends what is needed from the model and if we need to render a spinner for loading as example in the view.

1-That function makes a request and it gets handled by the router(like an executive assistant) which will direct to which controller(middleman) to manage the request.

3-The model receives instructions on what to do and hits the DB up to PUT,POST,or DELETE etc.

4- Model then takes that back and sends it back to controller. The controller communicates with view to render data for user. The view sends rendered file back to controller

5- Controller takes that rendered view and sends it to the user

What and Why Router

Router will process the request type and path and then like the sorting hat in Harry Potter the router will send request to right controller.

Less Repetition

When coding our paths this would stop us from repeating so often. Would allow specific routes to be within a file related to that route.

Maintain

Allows for long-term maintenance and traceability.

SoC

Uses the principle Separation Of Concerns

what is mongoose to mongodb : it is a Object Data Modeling library. A bit like the relationship of express and node.

the ODM is just a way for us to write JS code that will then interact with a DB.



the mongoose schema - where we model our data by describing the structure of data, default values and validation.



With that schema we put that in a model providing an interface to MongoDB

**Model,
Mongoose
, and
MongoDb**



```
const userSchema = new mongoose.Schema({
  name: String,
  age: Number
});

const UserModel = mongoose.model('User', userSchema);

const doc = new UserModel({
  name: 'Jean-Luc Picard',
  age: 59,
  rank: 'Captain'
});
doc.name; // 'Jean-Luc Picard'
doc.age; // 59

// undefined, Mongoose strips out `rank` because it isn't in the schema
doc.rank;
```

Here is our example of a schema: because we have mongoose schema in place and then wrapped with a model, when **rank** was attempted to be passed in mongoose takes that out and keeps what the schema had structured for MongoDB already.

- <https://masteringjs.io/tutorials/mongoose/schema>