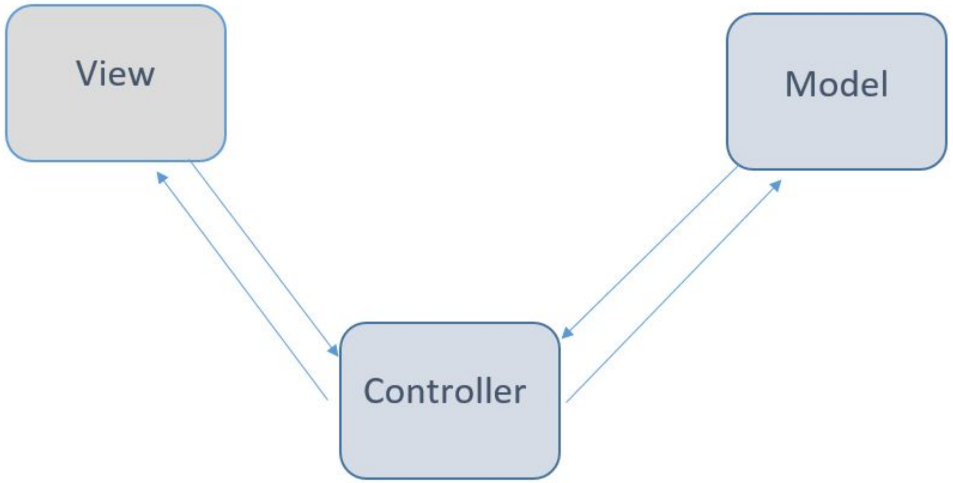
The Model View Controller

An Introduction



What is the MVC?

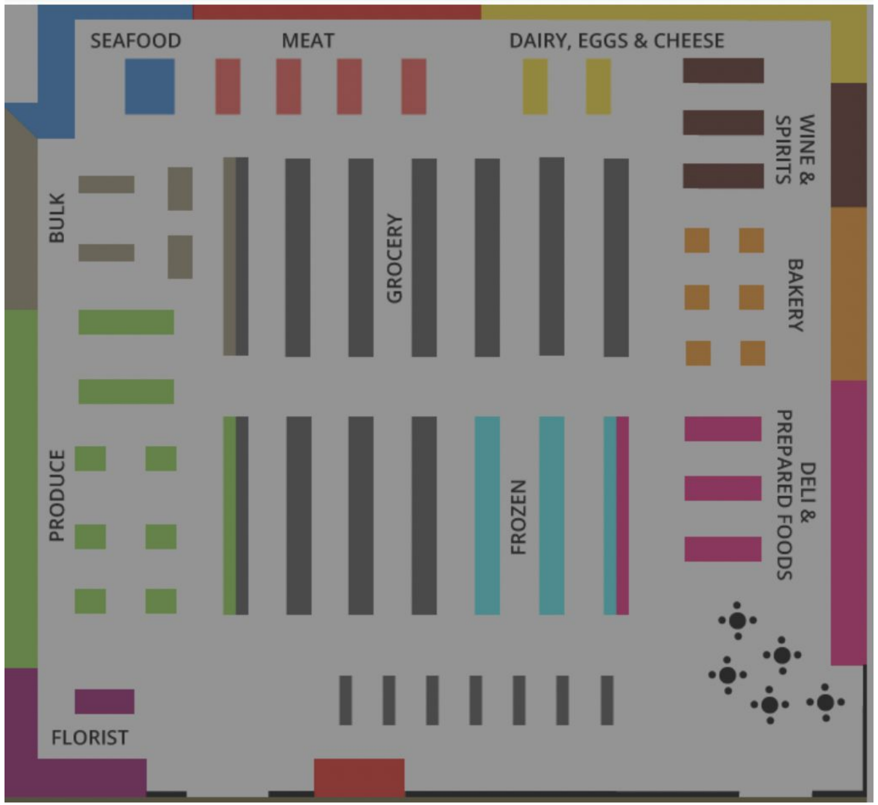
- MVC is not a language, nor is it specific to any language
- MVC is not a Library

You may see the Model View Controller being referred to as :

- An Architecture
- A Design Pattern
- A Framework

“WELL, WHICH IS IT?!”

Let’s consider the LAYOUT of a grocery store:



Regardless of where you live, you have probably seen a grocery store that follows a similar pattern when determining the layout of their store. This is not an arbitrary decision. It is based on ease of use for the client and worker, efficiency of available space, and compartmentalization of various activities taking place.

A grocery store would not place their cantaloupes in the meat case.. No one would know where to find them!

Think of the MVC as the architecture of your application. Its structure provides clarity for those working with it, as well as modularity: The ability to replace one piece without breaking everything.

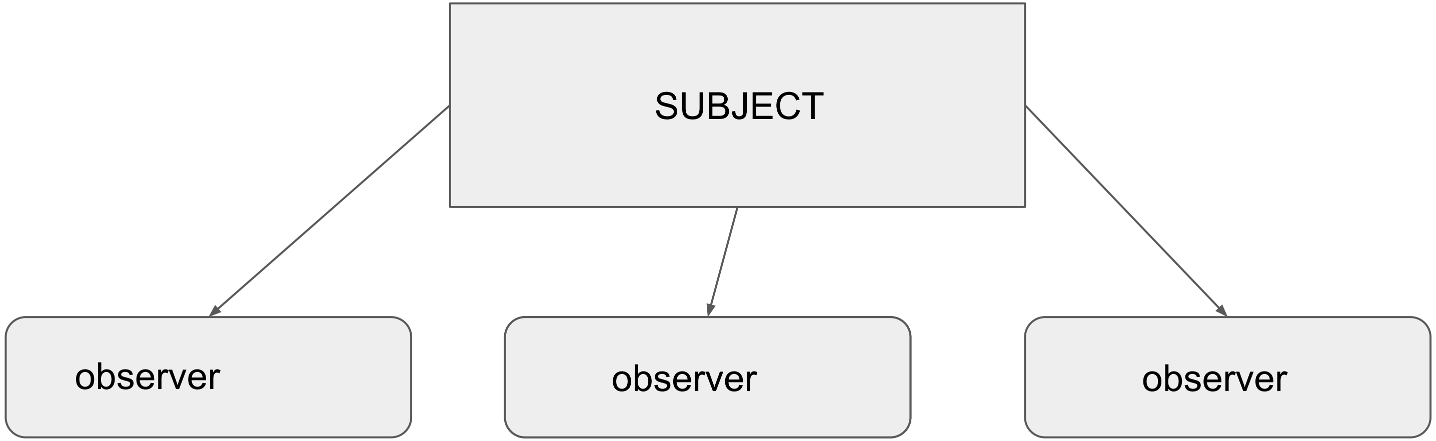
MVC is based on pre-existing design patterns. The three most commonly mentioned are:

- Observer
- Composite
- Strategy

Let’s take a look at these…

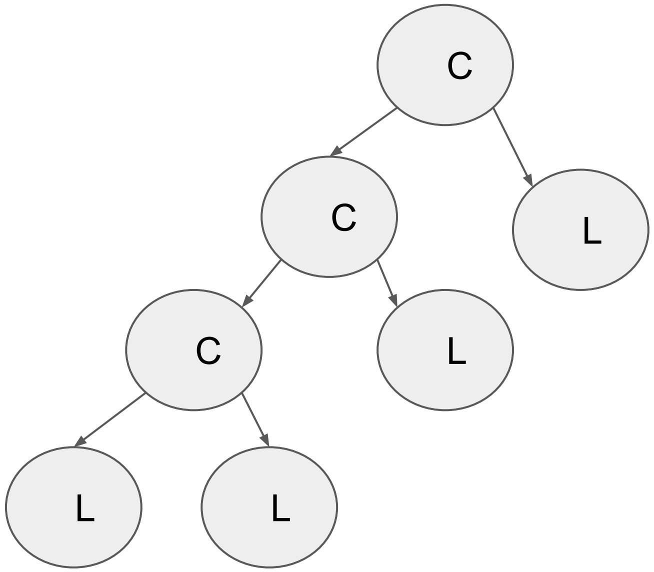
The Observer design pattern:

The Observer pattern describes the use of an object (referred to as the "subject") which has dependents, or "observers". The Subject notifies the observers of any changes in state.



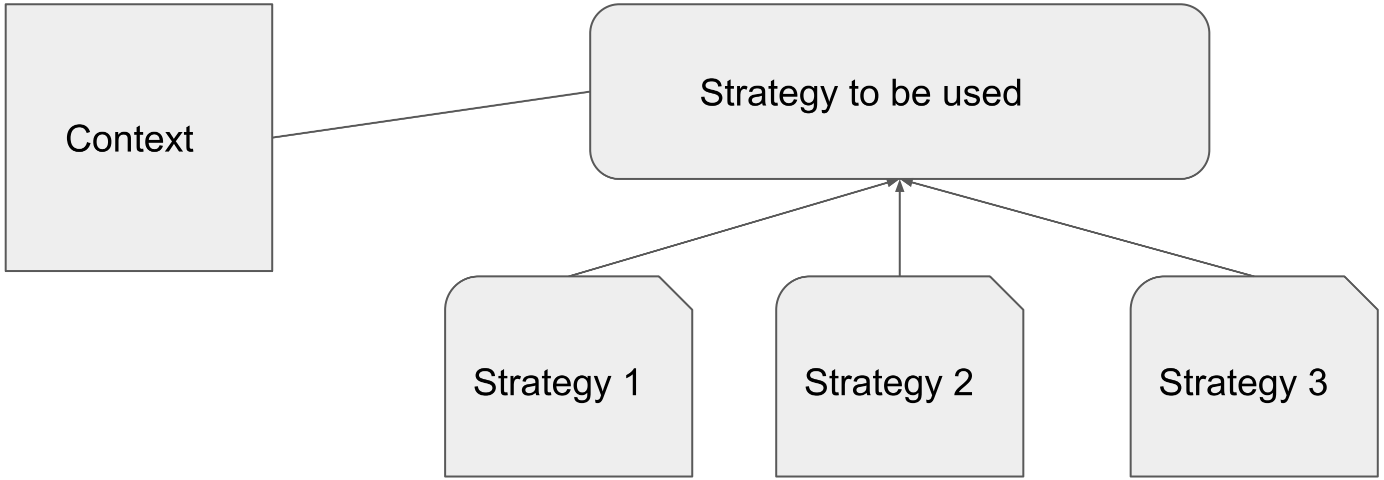
The Composite Design Pattern:

The composite pattern organizes groups of objects into related hierarchies. This allows for a group of objects to be manipulated in the same way a single object would. Tree structures are created where a “composite” contains more objects below it, and a “leaf” represents an object with no child nodes

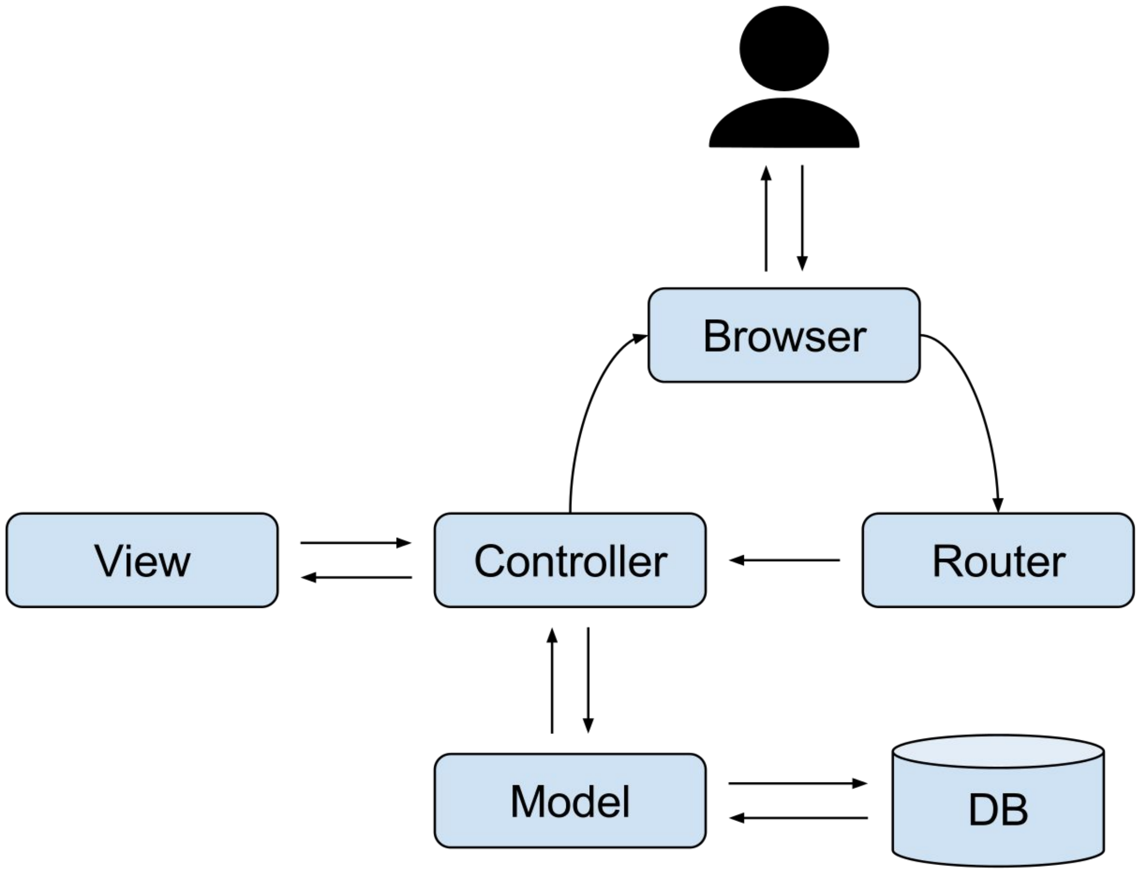


The Strategy Pattern:

The Strategy Pattern allows for the selection of various algorithms at runtime, depending on the action to be performed. This ensures that the most appropriate strategy can be implemented, rather than one generic catch-all solution.



COMMUNICATION FLOW IN MVC ARCHITECTURE



LET'S BREAK DOWN THE PARTS…

The Model

- Interacts with the database (retrieval, updates, etc.)
- May contain the schema (more on this later)
- Communicates with Controller, usually does not have access to Views

The Model is usually the only part of an application that has access to the database. It can perform operations such as finding a particular entry, changing a value stored in an entry, and many other tasks. It may also provide a structure for how data going into the database must be formatted.

This structure formatting can be thought of as creating a schema…

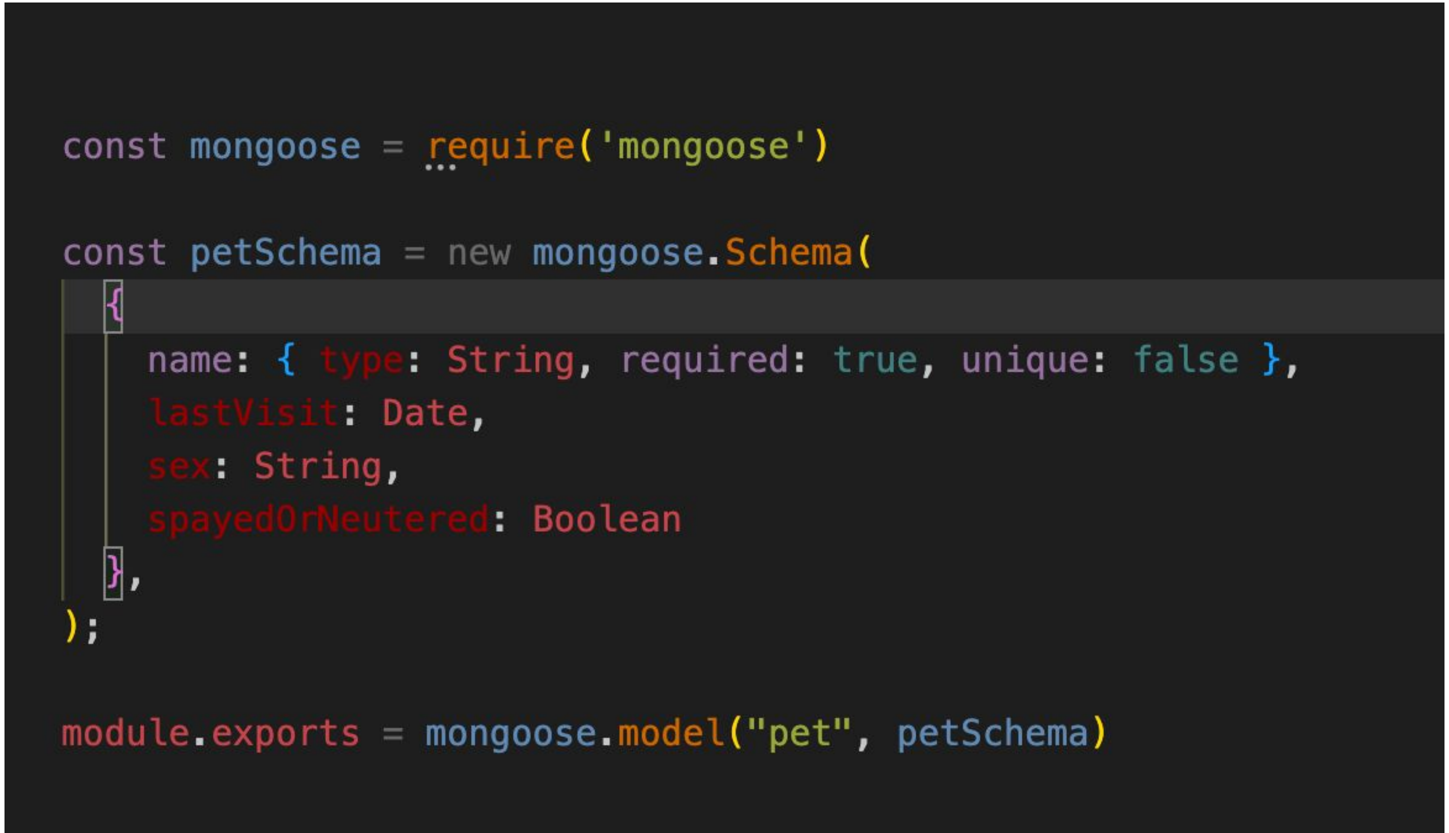
Model sidenote…

The Schema

- Older database management systems like MySQL have rigid structures when adding data to a collection. All data must follow the same format and contain all of the properties required or it cannot be added.
- This structure is referred to as the “Schema”. It is a blueprint for all entries, ensuring that they share the same structure with every other entry.
- MongoDB is more flexible in that it will allow for inconsistencies in data types or formatting. Instead, the designer can determine what schema they would like to employ.
- Libraries like Mongoose can act as an intermediary between data provided by a client and it’s representation in the database. It can use “schema validation” to ensure that data being entered into the database is presented in a way that it will remain retrievable and capable of being compared to other entries.

Let’s look at an example…

Here’s a schema for formatting a Veterinarian’s office database. Notice the structure every entry must follow… last visit must be provided as a date, whether the animal is spayed or neutered must be provided as a boolean, etc. Also, notice the designer can also specify whether a piece of data must be unique and whether it is required



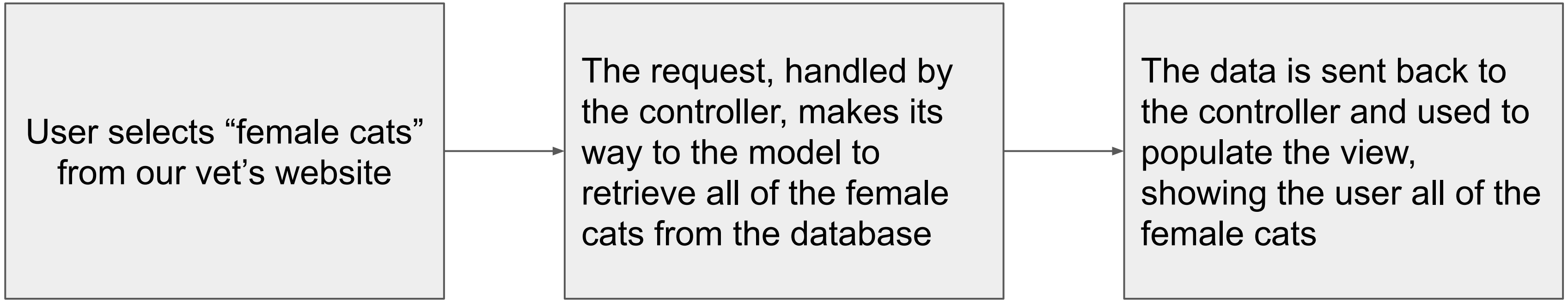
```
const mongoose = require('mongoose')

const petSchema = new mongoose.Schema(
  {
    name: { type: String, required: true, unique: false },
    lastVisit: Date,
    sex: String,
    spayedOrNeutered: Boolean
  },
);

module.exports = mongoose.model("pet", petSchema)
```

The View

- The view is our visual representation of the application. It’s what gets sent back to the user after they make a request. The user interacts with the view.
- It is often **dynamically rendered** using a templating language (EJS, handlebars, pug, react, etc.) This is because the view must be populated with ever-changing information. A static html page would not be able to change based on user input and an updated database.
- The view communicates with the controller.



User selects “female cats” from our vet’s website

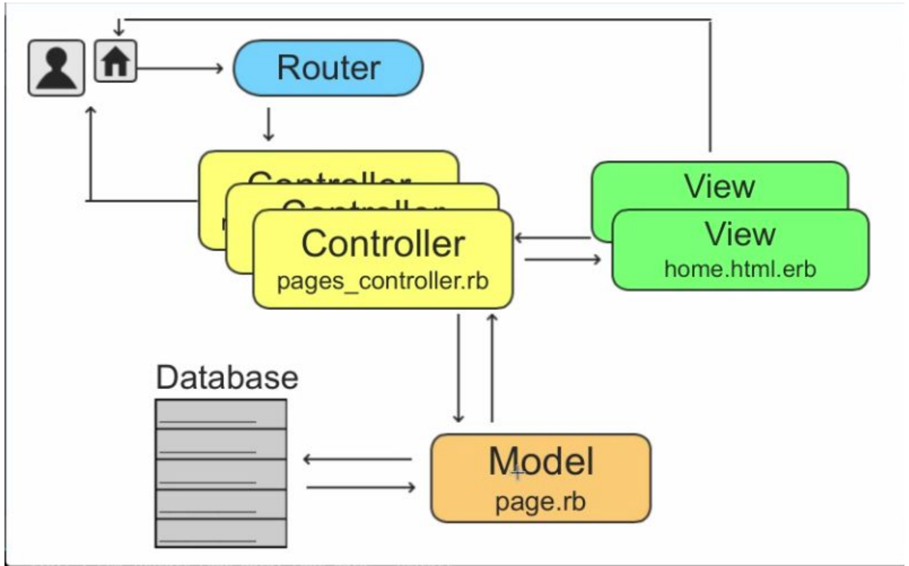
The request, handled by the controller, makes its way to the model to retrieve all of the female cats from the database

The data is sent back to the controller and used to populate the view, showing the user all of the female cats

The Controller

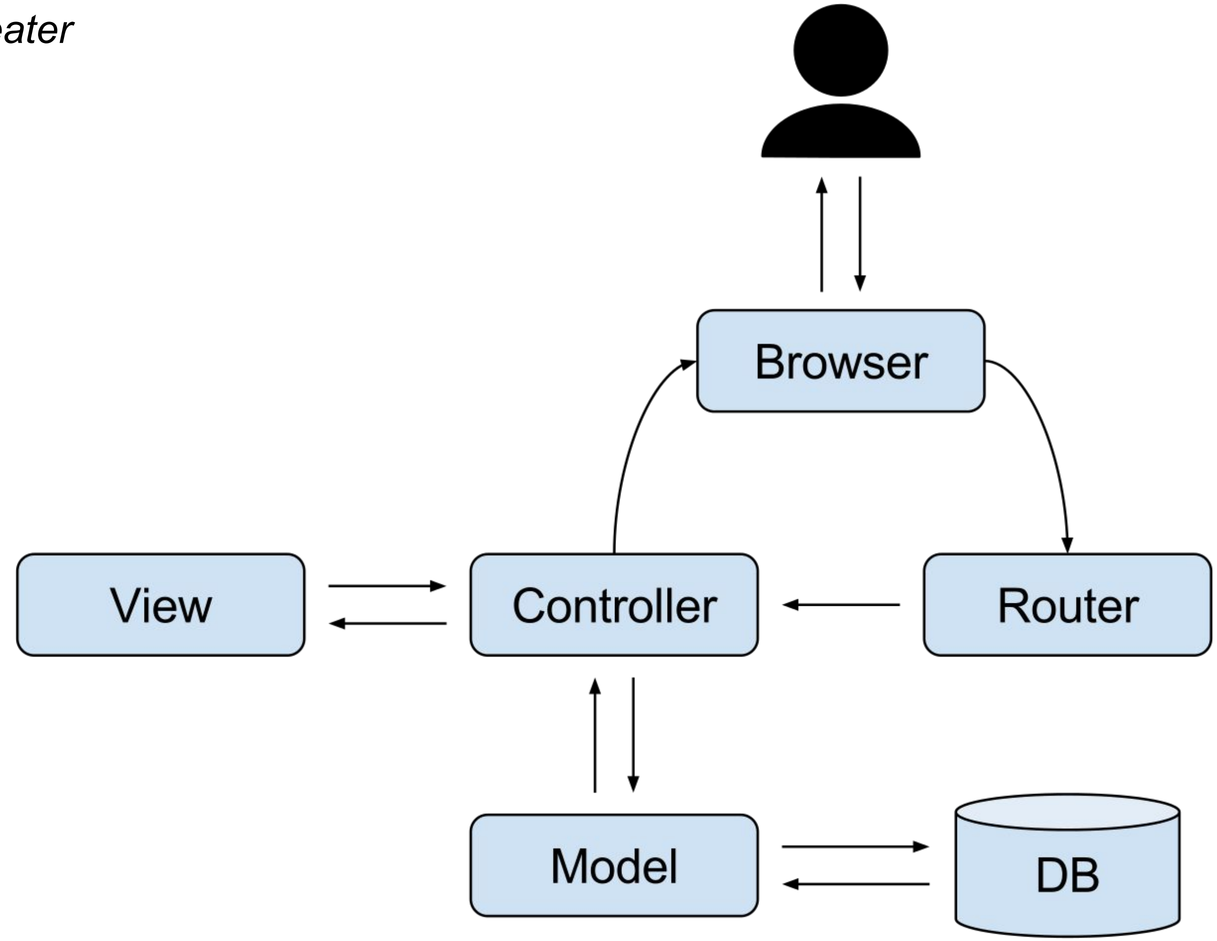
- The controller is the traffic cop of our application. It directs the flow of information to the appropriate destinations. There can be more than one.
- Routers can be separated out of the controller, used specifically for handling HTTP requests (get, post, put, delete) and the routes they were sent on (/login, /userprofile, etc.)
- Routers help to get requests to the necessary controller.
- The controller communicates with both the model and the view.

Picture: Routers may be used to pass requests off to different controllers intended for different tasks.



You may want the controller to behave differently when dealing with your login page than your home page. The router can look at our URL and pass off requests to different controllers based on where the request is coming from

Once more, in greater detail


Browser
View
Controller
Router
Model
DB