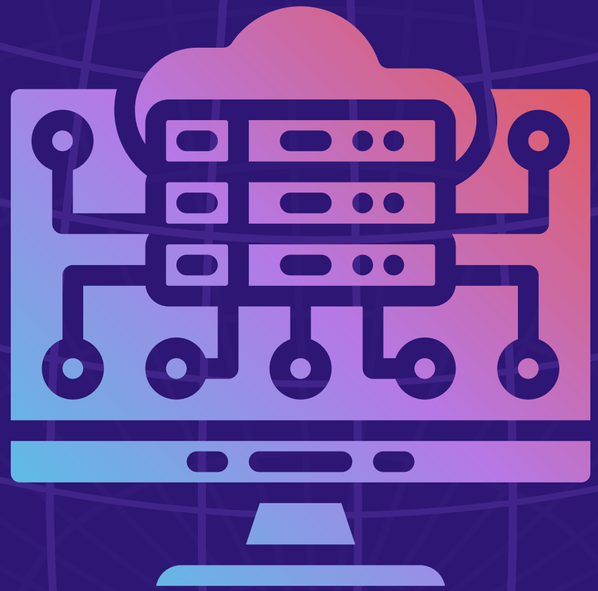
ALL ABOUT MVC



Presented by Katie Hom

#100DEVS

WHAT IS MVC?

MVC stands for

Model-View-Controller

It is one design or architectural pattern used to organize the structure of a web application.

WHAT IS MVC?

MODEL

Database

VIEW

Client

CONTROLLER

Server

What does MVC look like in a web application?

Let's think about our application as if it was a restaurant...

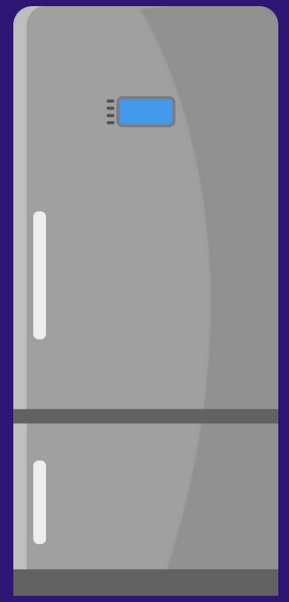
A customer (user) enters the restaurant and asks for a table in the bar area where they can order appetizers

The manager (router) determines where the customer should sit and takes them to their table (route)

The waiter (controller) takes the customer's order and tells the kitchen (model) what to prepare for their appetizer (view)

The chef and kitchen staff (model) consult the recipe (schema) and ask for ingredients to begin preparing the customer's appetizer...

...and ask restocking staff (Mongoose)
to get extra ingredients (data) from the cooler
and pantry (MongoDB) to fulfill the order



Once all of the ingredients are there, the kitchen (model) finishes cooking the appetizer

The waiter (controller) then sends the appetizer to the kitchen expo (view)

The expo (view) checks to make sure that the food is arranged properly, adds garnish, and sends it back to the waiter

The waiter (controller) then serves the customer their appetizer (view in browser)

And the customer (user) enjoys their appetizer (view) but flags down the manager (router)...

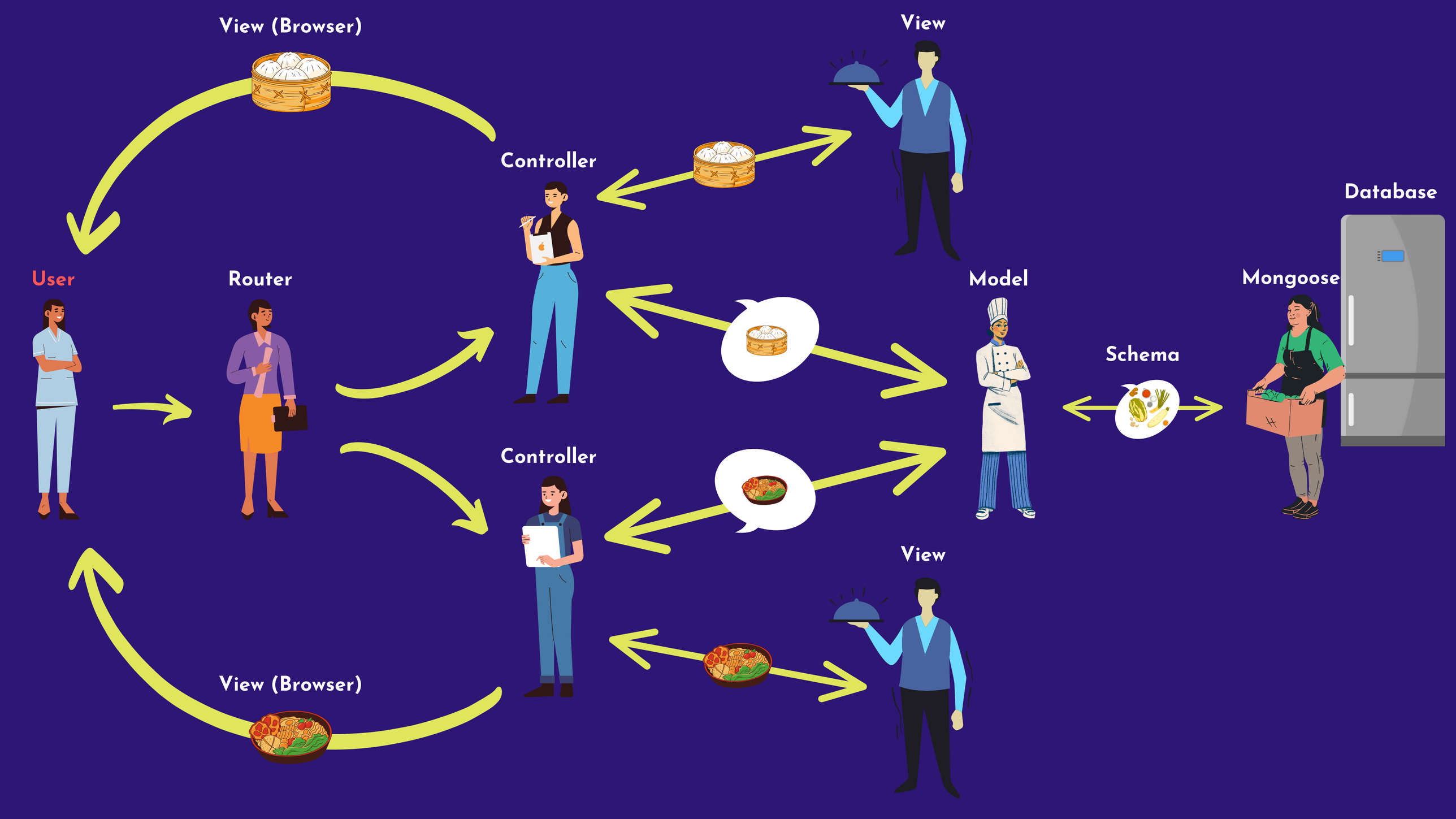
Because the customer (user) wants a new table in the main restaurant where they can order a main course (view)...

So the manager (router) determines where the customer should sit now and directs them to their new table (route)

The new waiter (controller) takes the customer's order and tells the kitchen (model) what to prepare for their main course (view)

The chef and kitchen staff (model) consult the recipe (schema) and ask for ingredients to begin preparing the customer's main course...

And the cycle continues . . .


View (Browser)
View
Controller
Database
User
Router
Model
Mongoose
Schema
Controller
View
View (Browser)

MODEL

Our application is built on our Model (kitchen) - the central component.

The model determines the types of data (food) served,
and the dynamic structure (meals) for those data.

The model (kitchen) communicates with Mongoose (restockers)
to retrieve specific pieces of data (ingredients) from the
database (pantry) following the schema (recipe)
to fulfill user requests (meal orders).

CONTROLLER

The controller (waiter), or server route for each respective request responds to user input and converts it into commands for the model (kitchen) and the view (expo).

The controller takes the input (order), validates it, and communicates with the model (kitchen).

Once the meal is served up, the controller takes the data (food) to the view (expo).

VIEW

The view (expo) verifies that the data being served (meal) is in the correct format and ensures the correct presentation (adds garnish).

The view (meal) is then sent back to the controller (waiter) who serves the view to the user in the browser.

WHY USE MVC?

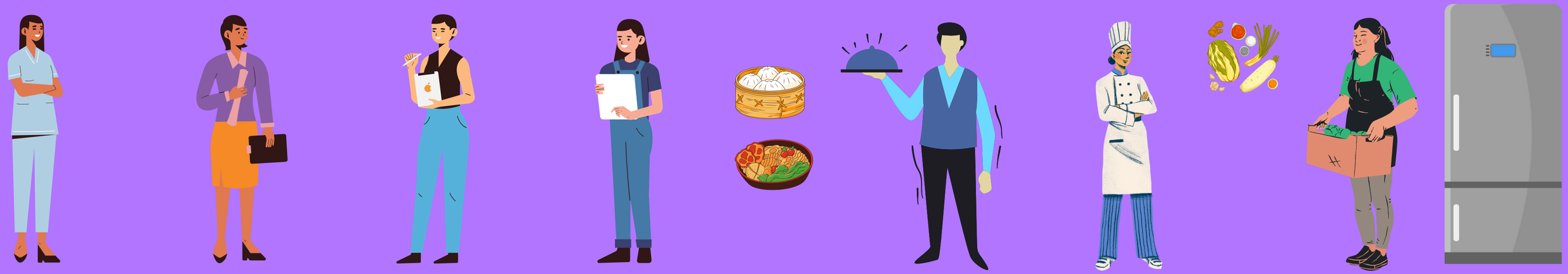
Separating the Model from the Views and Controller, allows us to better see the connections in our applications with greater understanding and structure for teams.

Using MVC design patterns separates, or abstracts, the different elements of our code so that they can be more easily scaled, debugged, and maintained.

WHAT IS A USE CASE OF MVC?

If there is an application and the team wants to change the template or framework it is using, there is no need to touch the code for the models or controllers.

The team can, instead, focus on changing the code within the View and not worry about making any changes that will affect database communication or server connections.



MVC: In Conclusion

While MVC is only one design pattern, its goals remain consistent:

- Improve the structure and architecture of our applications by separating out the Model (database), View (client), and Controller (server)
- Allow for better code maintainability & scalability
- Abstraction for ease of changes for future code upgrades or improvements