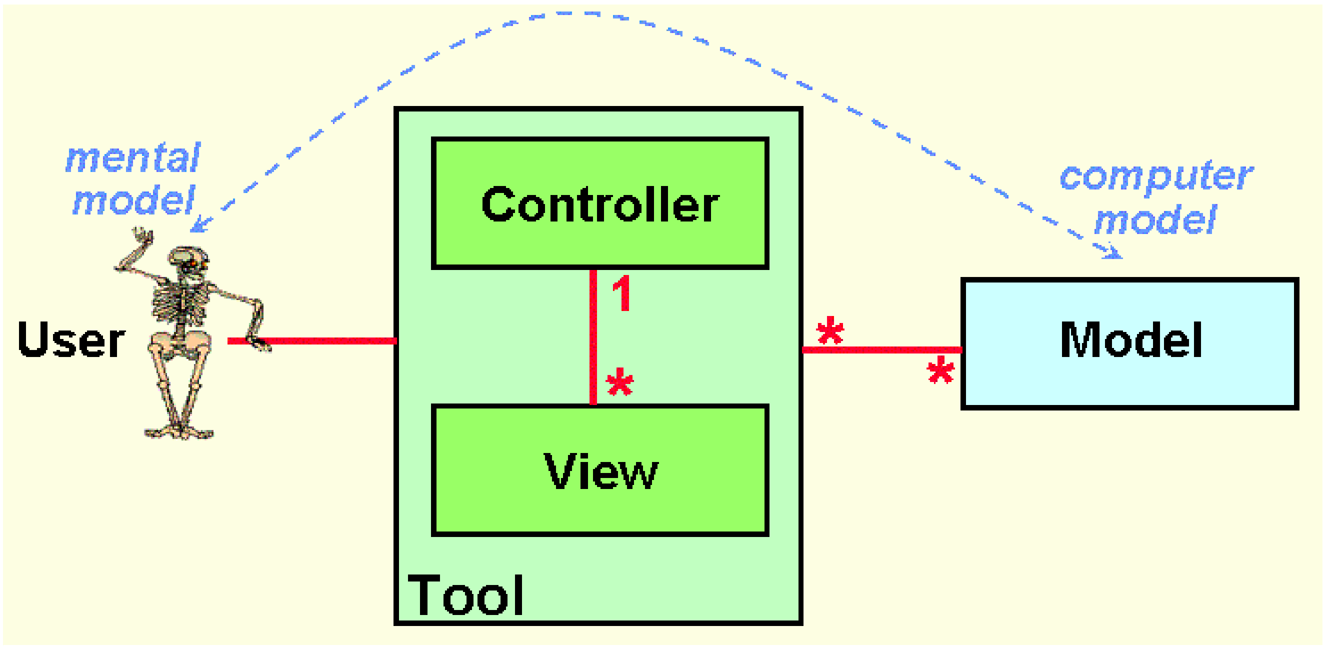
OCTOBER 7, 2017 / #PROGRAMMING

What is MVC, and how is it like a sandwich shop?



by Adam Wattis

In today's Internet, websites tend to be interactive, dynamic, and serve some sort of function. They can be more than a static HTML and CSS page. Here's where the **Model View Controller** (MVC) architectural pattern comes in.

User interaction enables use cases which would be impossible with only a statically loaded page. This is why, in modern web development, it is important to understand how dynamic pages are created. Perhaps the key to this is familiarity with the MVC architectural pattern.

If you are a beginner at web development, words such as “architectural pattern” may sound dauntingly complex and abstract. But the general idea behind MVC is actually very intuitive. I will do my best to explain it in such a way in this article.

Is MVC important to understand?

In my mind, the answer to this question is yes.

MVC is important to understand because it is the basic structure which most web applications are built on. The same is also true for mobile apps and desktop programs.

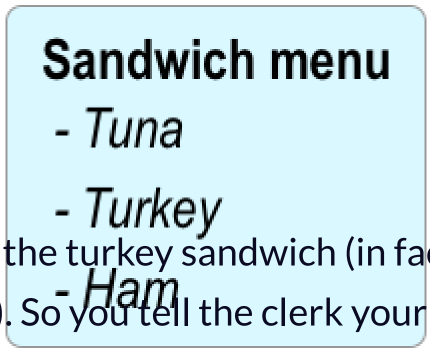
There are many variations around the basic idea of MVC. The initial conceptwas created around 1978 at Xerox PARC by Trygve Reenskaug. It was intended to help an end user manipulate and control an underlying computer system in a more visual and intuitive way.

MVC achieves this though letting a user interact with a User Interface. This allows for manipulation and control over the system.

The high-level concept

Without using any fancy words, I will now explain the basic idea behind MVC through a simple use case.

Imagine you are in a sandwich shop. You walk up to the counter and look at the menu.



Very simple sandwich menu

You decide you want the turkey sandwich (in fact, you can already imagine biting into it). So you tell the clerk your order.

The clerk knows exactly what you want when you order the turkey sandwich. He turns around to the sandwich-making station, and tells the people there what they need to know to fulfill your order.

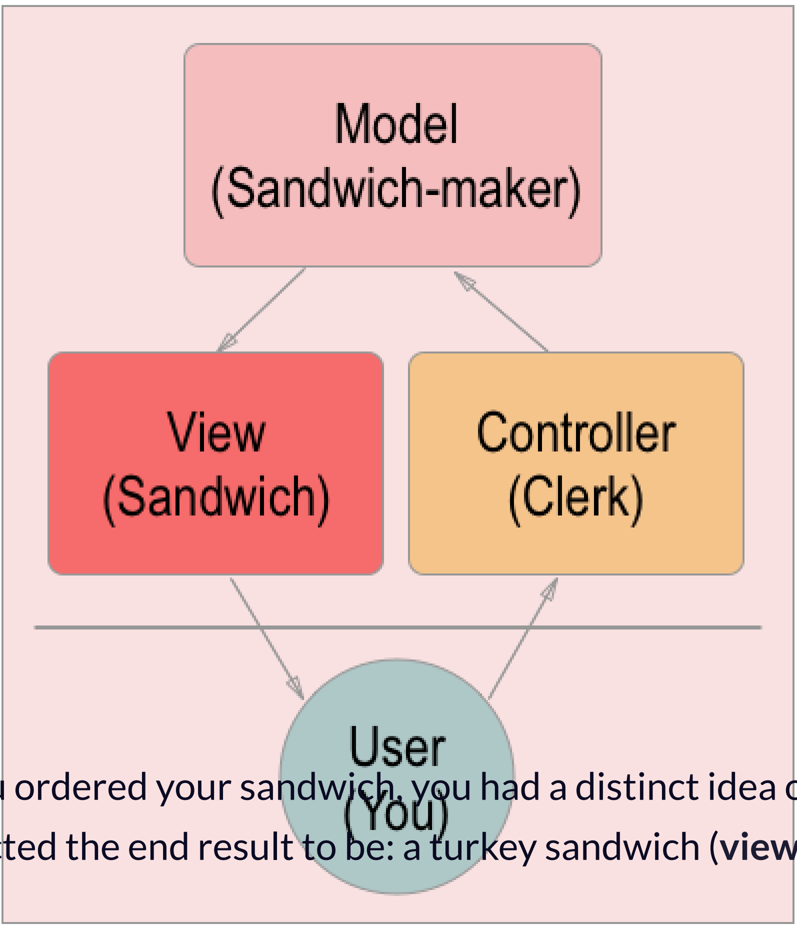
The sandwich-making team has many resources at their disposal. Ham, turkey, tuna, salad and cheese can all go in the sandwiches. They take the ingredients required for your order, and assemble them into the turkey sandwich that you have ordered.

After the sandwich is complete, it is handed over to you. You now have the turkey sandwich you wanted.

An explanation

In this previous example, there were three separate and distinct objects which all represent one part in our MVC:

- The sandwich-making station (Model)
- The finished sandwich you received in the end (View)
- The clerk (Controller)



When you ordered your sandwich, you had a distinct idea of what you expected the end result to be: a turkey sandwich (**view**).

Pretty simple flow of activities

This is the same as when you're on a website. For example, on Facebook you can press the 'Friends' button to see a list of your friends. You would expect your friend list to appear, and can already visualize it in your mind.

When you press the 'Friends' button, you make a request to Facebook's servers. The request is to serve you with your friends list, just like you requested your sandwich to the clerk (**controller**).

Your request arrives at Facebook's servers. It hits their controller, which tries to resolve it. It then grabs all your friends from a database, just like the sandwich-maker (**model**) grabs all the ingredients.

These resources (your friends list data) is assembled into a response. This is similar to how the sandwich-maker assembled all the ingredients into a finished sandwich (**view**).

This friends list is then sent to you for consumption, like the sandwich was in the end of your order.

Summary

The clerk is the controller:

- He knows all the possible combinations of sandwiches you can order
- He gathers your info and sends an order back to be resolved

The sandwich-makers are the models:

- They know what items are needed to assemble your finished sandwich

The sandwich is the view:

- It is the 'thing' the end user finally receives

Using a web framework

Controller:

The controller handles incoming requests. In a web framework, this would be a declaration of specific URLs that map to specific functionality that composes your request.

```
Example URLswebsite.com/profile/ --> returns your profilewebsite.cor
```

Model:

This is what your data looks like on the back end.

```
User:- userName- firstName- lastName- friends
```

View:
This is the HTML template that is returned after your request is resolved. If the request is successful, you should get a page with your friends. Otherwise, you might get a 404 ‘Not found’ page.

```
<ul>  <li>Friend 1: {friendList[0].userName}</li>  <li>Friend 2: {fr
```

Conclusion

When interacting with a system, you are usually able to **C**reate, **R**etreive, **U**pdate and **D**elete objects in the underlying database. This is often abbreviated to “**CRUD**.” Here, we have looked at retrieving data.

I did not explain here how a user can modify the data in the database (the **C**, **U** and **D** in **CRUD**). Usually, you are able to add, update and delete things on a website.

The functionality for this is pretty much the same as explained above. The difference is, your data is attached to your request to the controller.

I hope that you now have a clearer understanding of what MVC architecture is and how it might work.

If you thought this explanation was helpful, or have any questions or thought about how to improve this article, please feel free to comment!

If this article was helpful, tweet it.

Learn to code for free. freeCodeCamp's open source curriculum has helped more than 40,000 people get jobs as developers.

Get started

Trending Guides

What is Docker?	What is STEM?
TCP/IP Model	JavaScript Void 0
RTF File	SQL Delete Row
CSS Transition	JavaScript Replace
How to Use Instagram?	Python JSON Parser
MBR VS GPT	cmd Delete Folder
FAT32 Format	What is NFC?
Error 503 Code	Content Type JSON
Windows Hosts File	Convert HEIC to JPG
Mobi to PDF	Math Random Java
WordPress for Beginners	Google Docs Landscape
Qualitative VS Quantitative	Antimalware Executable