My Path
Catalog

Codecademy Resources > ruby

MVC: Model, View, Controller

App organization explained

Background

MVC is short for Model, View, and Controller. MVC is a popular way of organizing your code. The big idea behind MVC is that each section of your code has a purpose, and those purposes are different. Some of your code holds the data of your app, some of your code makes your app look nice, and some of your code controls how your app functions.

MVC is a way to organize your code's core functions into their own, neatly organized boxes. This makes thinking about your app, revisiting your app, and sharing your app with others much easier and cleaner.

The parts of MVC

Model: Model code typically reflects real-world things. This code can hold raw data, or it will define the essential components of your app. For instance, if you were building a To-do app, the model code would define what a "task" is and what a "list" is – since those are the main components of a todo app.
View: View code is made up of all the functions that directly interact with the user. This is the code that makes your app look nice, and otherwise defines how your user sees and interacts with it.
Controller: Controller code acts as a liaison between the Model and the View, receiving user input and deciding what to do with it. It's the brains of the application, and ties together the model and the view.

An analogy

MVC is a way to think about how an web application works.

It's kind of like how you make Thanksgiving dinner. You have a fridge full of food, which is like the Model. The fridge (Model) contains the raw materials we will use to make dinner.

You also probably have a recipe or two. A recipe (assuming you follow it exactly) is like the Controller of Thanksgiving dinner. Recipes dictate which stuff in the fridge you'll take out, how you'll put it together, and how long you need to cook it.

Then, you have table-settings, silverware, etc., which are what your hungry friends and family use to eat dinner. Table-top items are like the View. They let your guests interact with your Model and Controller's creation.

MVC in the real-world

MVC is helpful when planning your app, because it gives you an outline of how your ideas should be organized into actual code.

For instance, let's imagine you're creating a To-do list app. This app will let users create tasks and organize them into lists.

The **Model** in a todo app might define what what a “task” is and that a “list” is a collection of tasks.

The **View** code will define what the todos and lists looks like, visually. The tasks could have large font, or be a certain color.

Finally, the **Controller** could define how a user adds a task, or marks another as complete. The Controller connects the View’s add button to the Model, so that when you click “add task,” the Model adds a new task.

Wrapping up

MVC is a framework for thinking about programming, and for organizing your program’s files. To signify the idea that your code should be organized by its function, developers will create folders for each part of MVC. (The idea that aps should be divided based on the *function* of each part of the code is sometimes referred to as *separation of concerns.*) If you’ve looked at Codecademy’s Ruby on Rails course, you might have noticed that there is a folder for each part of MVC within every Rails app it introduces.

MVC gives you a starting place to translate your ideas into code, and it also makes coming back to your code easier, since you will be able to identify which code does what. In addition, the organizational standard MVC promotes makes it easy for other developers to understand your code.

Thinking about how code interacts with other code is a significant part of programming, and learning to collaborate with other developers is an important skill. Taking the time to think about how your app fits into the MVC framework will level-up your skills as a developer by teaching you both. It'll also make your apps better!

codecademy

Teaching the world how to code.

Company

- About
- Stories
- We're hiring
- Blog

Resources

- Articles
- Schools

Learn To Code

- Make a Website
- Make an Interactive Website
- Learn Rails
- Ruby on Rails Authentication
- Learn AngularJS
- Learn the Command Line
- Learn SQL
- SQL: Analyzing Business Metrics
- Learn Java
- Learn Git

- HTML & CSS
- JavaScript
- jQuery
- PHP
- Python
- Ruby
- Learn APIs

Privacy Policy Terms

English ▼