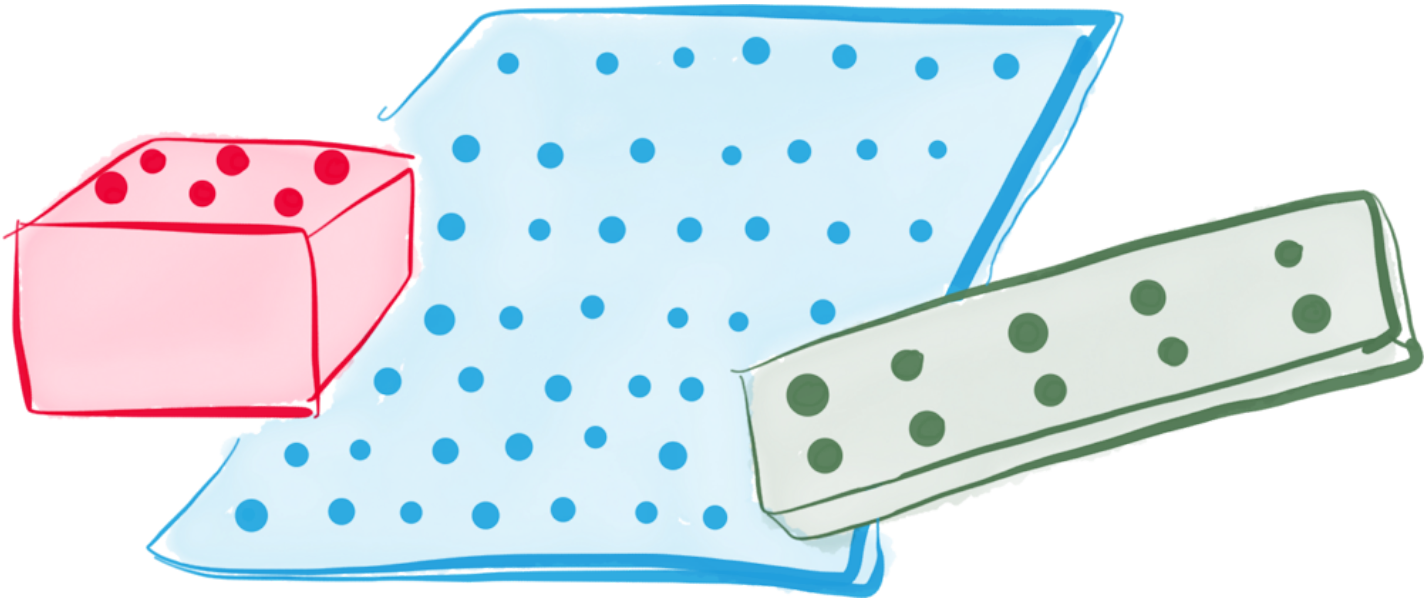
DEC 20TH, 2014 | COMMENTS

Model-View-Controller (MVC) Explained -- With Legos



This is a guest post by Alex Coleman, a coding instructor and consulting web developer. He is the founder of Your First Web Development (http://yourfirst.io/), which provides web development courses and instruction for beginners.

To demonstrate how a web application structured using the **Model-View-Controller (http://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller)** pattern (or **MVC**) works in practice, let's take a trip down memory lane...

Legos!

You're ten years old, sitting on your family room floor, and in front of you is a big bucket of Legos. There are Legos of all different shapes and sizes. Some blue, tall, and long. Like a tractor trailer. Some red and almost cube shaped. And some are yellow – big wide planes, like sheets of glass. With all these different types of Legos, there's no telling what you could build.

But surprise, surprise, there's already a **request**. Your older brother runs up and says, "Hey! Build me a spaceship!"

"Alright," you think, "that could actually be pretty cool!" A spaceship it is.

So you get to work. You start pulling out the Legos you think you're going to need. Some big, some small. Different colors for the outside of the spaceship, different colors for the engines. Oh, and different colors for the blaster guns. (You gotta have blaster guns!)

Now that you have all of your **building blocks** in place, it's time to assemble the spaceship. And after a few hours of hard work, you now have in front of you – a spaceship!

You run to find your brother to show him the finished product. "Wow, nice work!", he says. "Huh," he thinks, "I just asked for that a few hours ago, didn't have to do a thing, and there it is. I wish *everything* was that easy."

What if I were to tell you that building a web application is exactly like building with Legos?

It all starts with a *request*...

In the case of the Legos, it was your brother who asked you to build something. In the case of a web app, it's a user entering a URL, requesting to view a certain page.

So your brother is the user.

The request reaches the *controller*...

With the Legos, you are the controller.

The controller is responsible for grabbing all of the necessary **building blocks** and organizing them as necessary.

Those building blocks are known as *models*...

The different types of Legos are the models. You have all different sizes and shapes, and you grab the ones you need to build the spaceship. In a web app, models help the controller retrieve all of the information it needs from the database.

So the request comes in...

The controller (you) receives the request.

It goes to the models (Legos) to retrieve the necessary items.

And now everything is in place to produce the final product.

The final product is known as the *view*...

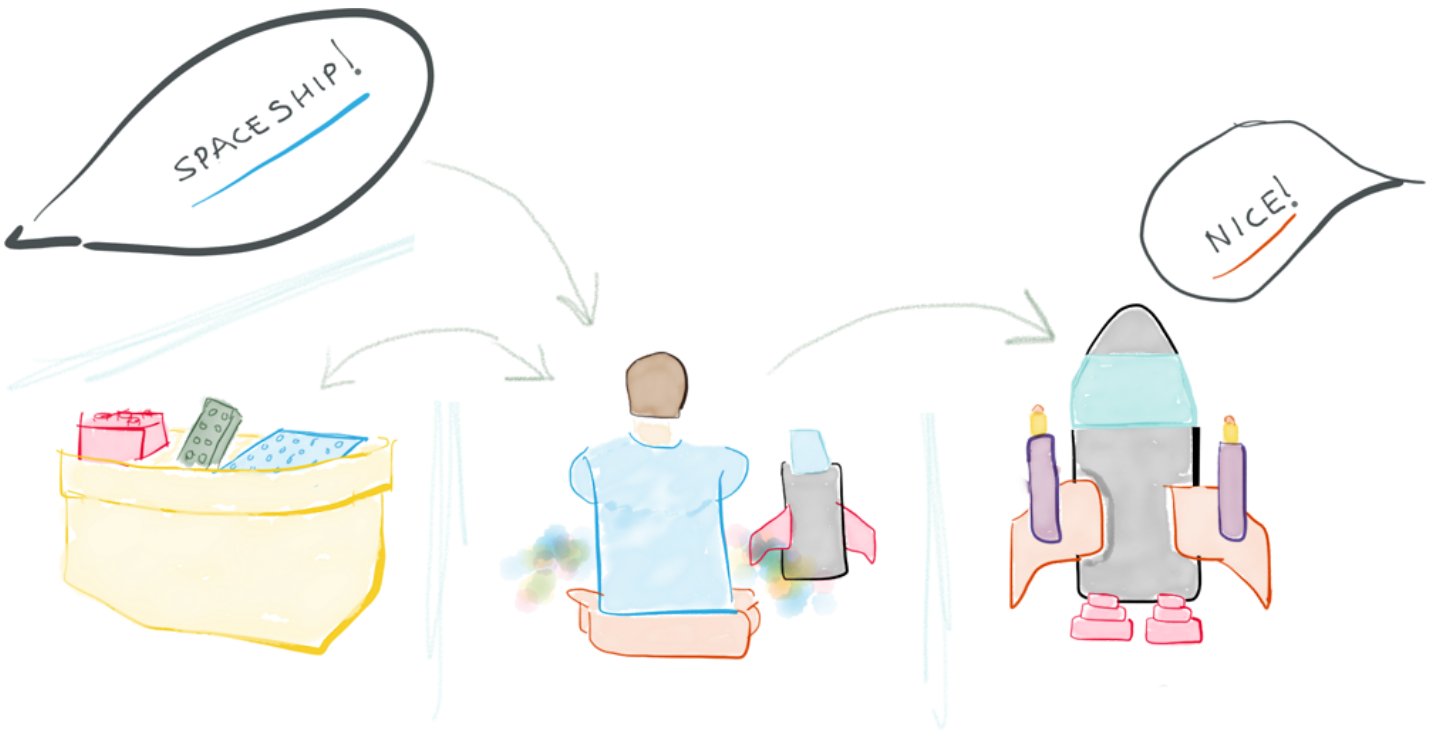
The spaceship is the view. It's the final product that's ultimately shown to the person who made the request (your brother).

In a web application, the view is the final page the user sees in their browser.

To summarize...

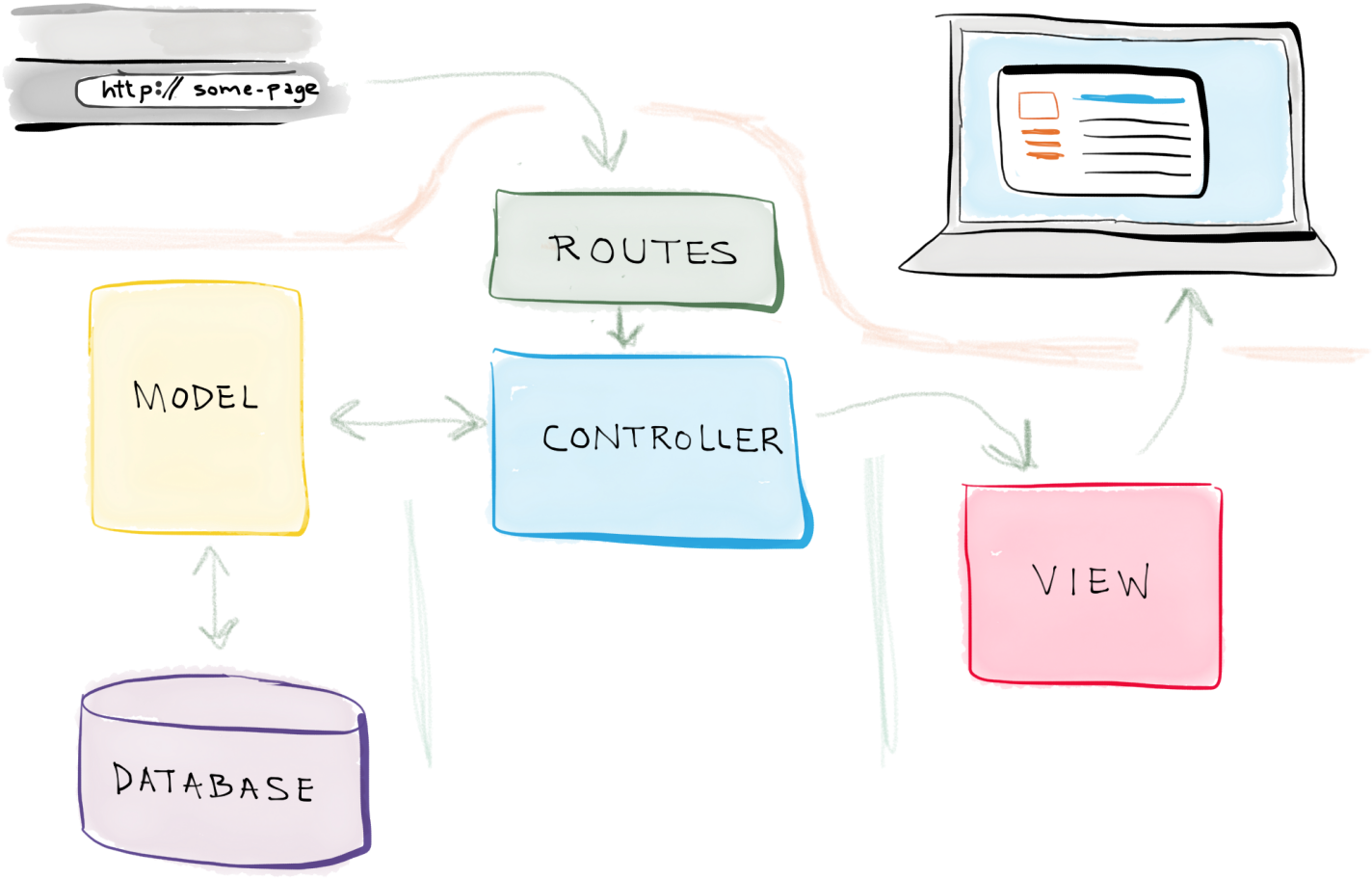
When building with Legos:

1. Your brother makes a request that you build a spaceship.
2. You receive the request.
3. You retrieve and organize all the Legos you need to construct the spaceship.
4. You use the Legos to build the spaceship and present the finished spaceship back to your brother.



And in a web app:

1. A user requests to view a page by entering a URL.
2. The **Controller** receives that request.
3. It uses the **Models** to retrieve all of the necessary data, organizes it, and sends it off to the...
4. **View**, which then uses that data to render the final webpage presented to the the user in their browser.



From a more technical standpoint

With the MVC functionality summarized, let's dive a bit deeper and see how everything functions on a more technical level.

When you type in a URL in your browser to access a web application, you're making a request to view a certain page within the application. But how does the application know which page to display/render?

When building a web app, you define what are known as **routes**. Routes are, essentially, URL patterns associated with different pages. So when someone enters a URL, behind the scenes, the application tries to match that URL to one of these predefined routes.

So, in fact, there are really *four* major components in play: **routes**, **models**, **views**, and **controllers**.

Routes

Each route is associated with a controller – more specifically, a certain function *within* a controller, known as a **controller action**. So when you enter a URL, the application attempts to find a matching route, and, if it's successful, it calls that route's associated controller action.

Let's look at a basic Flask route as an example:

```
@app.route('/')
def main_page():
    pass
```

Here we establish the `/` route associated with the `main_page()` view function.

Models and Controllers

Within the controller action, two main things typically occur: the models are used to retrieve all of the necessary data from a database; and that data is passed to a view, which renders the requested page. The data retrieved via the models is generally added to a data structure (like a list or dictionary), and that structure is what's sent to the view.

Back to our Flask example:

```
@app.route('/')
def main_page():
    """Searches the database for entries, then displays them."""
    db = get_db()
    cur = db.execute('select * from entries order by id desc')
    entries = cur.fetchall()
    return render_template('index.html', entries=entries)
```

Now within the view function, we grab data from the database and perform some basic logic. This returns a list, which we assign to the variable `entries`, that is accessible within the *index.html* template.

Views

Finally, in the view, that structure of data is accessed and the information contained within is used to render the HTML content of the page the user ultimately sees in their browser.

Again, back to our Flask app, we can loop through the `entries`, displaying each one using the Jinja syntax:

```
{% for entry in entries %}
  <li>
    <h2>{{ entry.title }}</h2>
    <div>{{ entry.text|safe }}</div>
  </li>
{% else %}
  <li><em>No entries yet. Add some!</em></li>
{% endfor %}
```

Summary

So a more detailed, technical summary of the MVC request process is as follows:

1. A user requests to view a page by entering a URL.
2. The application matches the URL to a predefined **route**.
3. The **controller action** associated with the route is called.
4. The controller action uses the **models** to retrieve all of the necessary data from a database, places the data in an array, and loads a **view**, passing along the data structure.
5. The **view** accesses the structure of data and uses it to render the requested page, which is then presented to the user in their browser.

Want to learn more?

Follow along with The Web App Journey (http://yourfirst.io/web-app-journey/) where I discuss every step I take to build a brand new web application from scratch – no code involved. No code? That's right. The focus is on the process – learn this and you'll be better equipped for when you do add actual coding into the mix with the Real Python (https://realpython.com) course!

Posted by Real Python Dec 20th, 2014 fundamentals (/blog/categories/fundamentals/), python (/blog/categories/python/)

See an error in this post? Please submit a pull request on Github (https://github.com/realpython/realpython-blog).

« Fingerprinting Images for Near-Duplicate Detection (/blog/python/fingerprinting-images-for-near-duplicate-detection/)

Handling Email Confirmation during Registration in Flask » (/blog/python/handling-email-confirmation-in-flask/)

Comments

Python Programming Contest - 'first to five'

72 comments • 5 months ago

michaelherman — I am grading them right now.

Fun With Django's New Postgres Features

7 comments • 10 months ago

michaelherman — Cheers!

Comparing Python Command-Line Parsing Libraries: Argparse, Docopt, and

9 comments • 6 months ago

Saurabh Hirani — This was a very useful comparison. Thanks a lot for putting this up!

Scaffold a Flask Project

15 comments • 10 months ago

michaelherman — Change this line- virtualenv_exe = which('pyvenv') To - virtualenv_exe = which('virtualenv')

Categories

- analytics (/blog/categories/analytics/) [3]
- api (/blog/categories/api/) [6]
- bottle (/blog/categories/bottle/) [2]
- data science (/blog/categories/data-science/) [7]
- devops (/blog/categories/devops/) [14]
- django (/blog/categories/django/) [28]
- docker (/blog/categories/docker/) [6]
- editors (/blog/categories/editors/) [3]
- flask (/blog/categories/flask/) [29]
- front-end (/blog/categories/front-end/) [10]
- fundamentals (/blog/categories/fundamentals/) [15]
- migrations (/blog/categories/migrations/) [3]
- opencv (/blog/categories/opencv/) [3]
- pyramid (/blog/categories/pyramid/) [1]
- scraping (/blog/categories/scraping/) [2]
- sql (/blog/categories/sql/) [1]
- testing (/blog/categories/testing/) [8]
- web2py (/blog/categories/web2py/) [1]

Questions? info@realpython.com (mailto:info@realpython.com).

Back to top