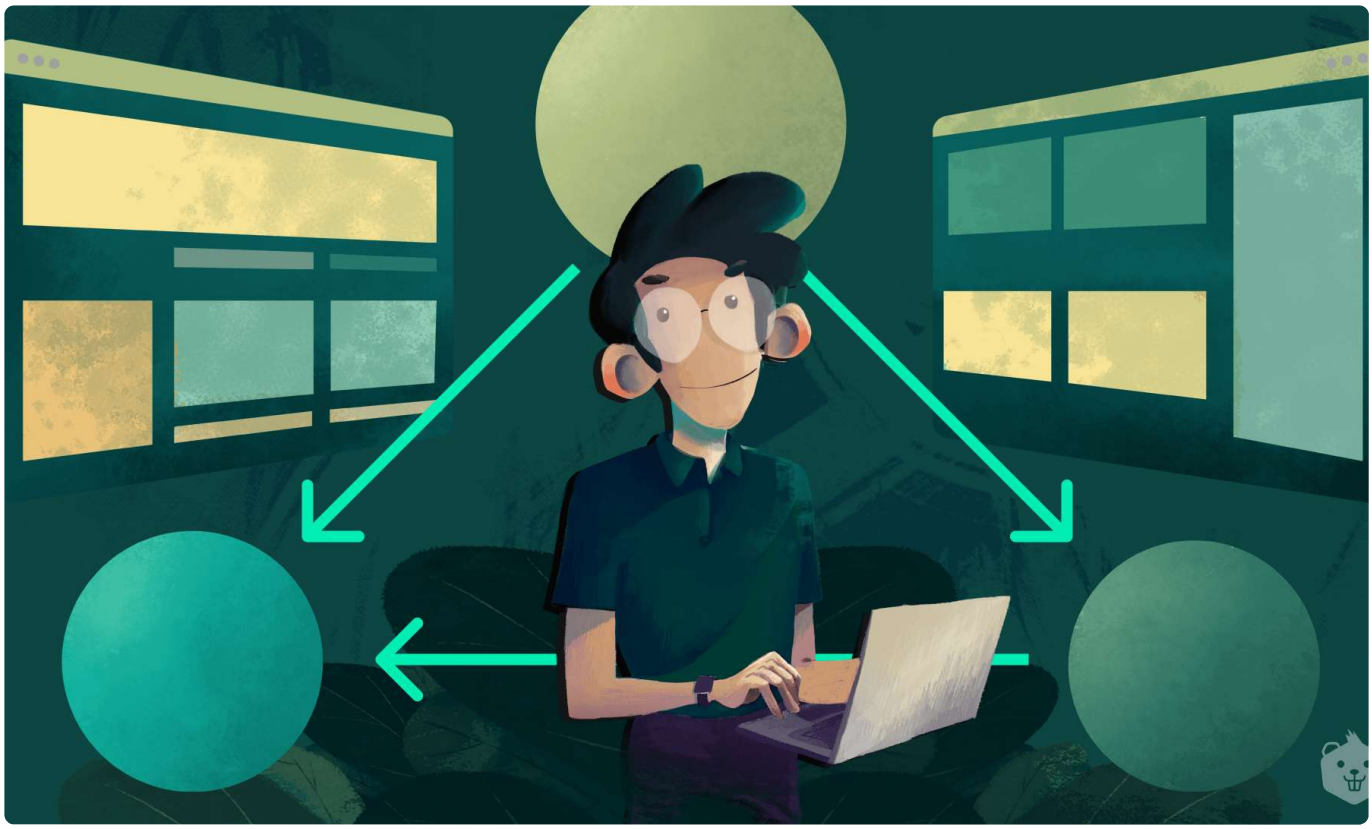
Wednesday, December 06, 2023 (IST)

WEB DEVELOPMENT

Understand MVC Architecture in 5 mins

By Jaidev Singh Bhui - October 06, 2022 - 5 min read



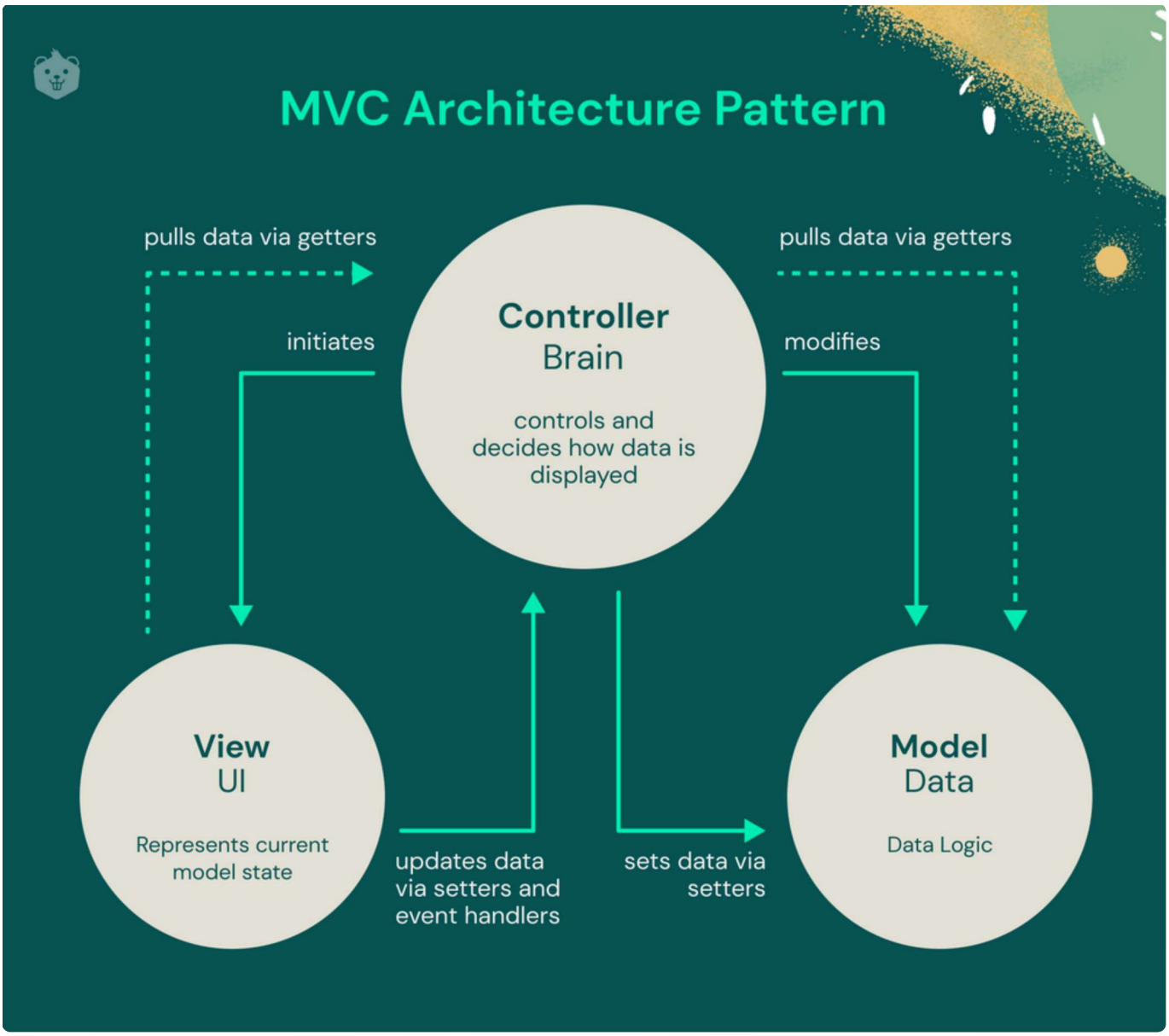
As a rookie web developer, or even an experienced one, you might have heard of the MVC architecture. Let's try to understand what it is, how it is used. and whether it is a design pattern or an architecture.

What is MVC?

MVC is an architectural pattern that is used to divide the application into three components, namely - **Model, View, and Controller**.

Separating the application into these three components makes it easier to scale the application and makes it more extensive and easier to maintain.

Components of an MVC architecture



Model

This component contains all the data and business-related logic, i.e the database of an application.

View

This component is used for the UI logic and how to visually present the data, i.e the front-end of a web application or a mobile application.

Controller

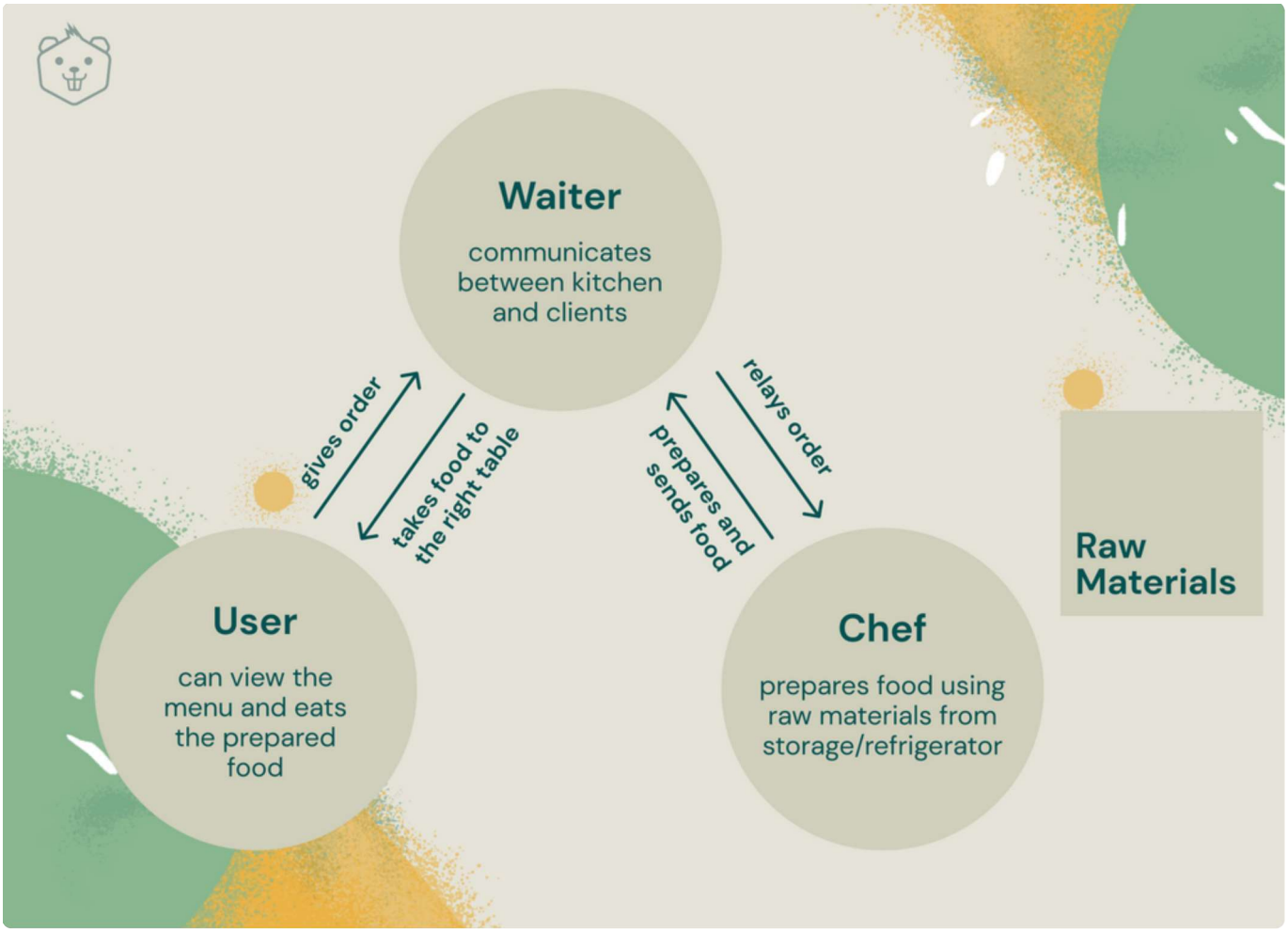
This component acts as an interface between the Model and the View, and handles all the incoming requests, by using the data from the Model and sending it to the appropriate View, to render the desired output.

As you can sense, this architecture can be used in creating large web applications and even mobile applications.

Restaurant analogy

Let's take the example of a restaurant.

When you visit a restaurant, the waiter presents you with a menu. You view the menu and then tell the waiter your order. The waiter notes down the order and passes it to the chef, in the kitchen. The chef uses the raw materials from the refrigerator and cooks the dish. The waiter then takes the dish from the kitchen and then presents it to you, after which you can enjoy your meal.



Real life example of MVC model

Using this short scenario, we can try to visualize how the MVC architecture works.

In this, you are the end user, who visits or uses the application, in this case, the restaurant. The **menu** and the food come under the **View component**, as that's the UI of the restaurant experience.

The chef and the refrigerator can be thought of as the **Model**, as the raw data is being extracted, transformed, and then presented to the user.

And, at last, the **waiter** is the **Controller**, who accepts all the incoming requests and acts as an interface between the View and the Model.

MVC architecture of a food ordering application

Say, you were to build a food ordering application. The user can log into the application, select a store, and view the menu of that particular store. Users can add items to their cart and then finally make the order.

You can divide the application into Model, View, and Controller as:

Model

All the data that is to be presented to the user, like the menu, the food item details, restaurant information, and other data. There can be different objects for each entity, like a Restaurant object, which consists of the Menu object, which can further contain multiple Food Item objects. It handles all the business logic-related data.

View

Everything the user can see and interact with on the application consists of the View. The list or grid of products, the food information layout, the shopping cart, everything UI-related is a part of the View component.

Controller

Whenever the user interacts with the View, for example, when the user clicks on a product to view its information, the View sends the request to the Controller to present the information about the product. The Controller asks the Model for the information about the product, which it extracts from the database. The controller then presents the data to the View and then renders the product information page with all the data. So, a controller's job is to present the right data with the right data to the end-user.

For example, each of the layers can be developed by different teams, like a team of UI and front-end developers can work on the View part, Database engineers can work on the Model part, while back-end engineers can work on the Controller part, all can be done separately. This makes the application loosely coupled and easier to debug in cases of failure.

This architecture can be implemented using different technologies and languages. Most of the majorly used languages and frameworks work on this pattern or have a similar working ideology. Mostly used frameworks are Ruby on Rails for Ruby, Spring for Java, ASP.NET MVC for C#, AngularJS for Javascript.

MVC: Design Pattern or Architecture?

In software engineering, an architectural pattern is a term used for reusable solutions for commonly occurring problems concerned with subsystems of an application. It defines the high-level planning to be done on the whole application level and the workflow.

Whereas, design patterns are the solutions for commonly occurring problems at the component or algorithmic level. This is more about the lower-level implementation of a particular component.

Like, if you were to build a city, on a high level, or an architectural level, you would decide which area would be residential, which would be commercial, just get a broader picture. Now, to dig deeper into the residential area, we would have to design and decide stuff like what kind of houses would be there, how many floors to be constructed, how many bedrooms and bathrooms to be built in a particular house. This is the lower-level implementation, the design part.

Similarly, the Model-View-Controller pattern gives an overarching view of the application at a higher level, with all these different layers and components, and

What is the advantage of MVC?

1. Loosely coupled
2. Highly cohesive
3. Rapid Development
4. Easier to test and debug

Let's look at each of the following advantages in more detail.

Loosely coupled

As explained before, each of the three components can be developed and worked on separately by different teams. This is possible because the functioning of each of the components is independent of each other. In software engineering, the degree of inter-dependence of a component or a module on other modules for its proper working is called coupling. The less the interdependence between the components, the looser the coupling between them.

Highly cohesive

Another concept is cohesion, which defines how closely all the code in a particular module supports a central purpose. In MVC, all the components are highly cohesive, which makes them highly maintainable and reduces modular complexity.

Rapid Development

Since there is a separation of concerns, different teams can work independently on the different components of the architecture, and the application can be developed at a rapid pace.

Easier to test and debug

All the different teams and departments can run tests on their own components, which is easier than running every piece together. Loosely coupled components can be developed independently and whenever there is a failure in the system, it is easier to find and fix the point of failure.

Perfect for large teams

The tasks can be broken down into smaller components, and be distributed in a large team, thus utilizing the resources efficiently.

Limitations of MVC

Although this architecture excels in many places, it does have its limitations, and the limitation is not architecture-related but more implementation-based.

The hardest part of the MVC pattern is that it can get very complex to understand this pattern and implement it into an application.

Also, the developers using this architecture are expected to be skilled in multiple technologies, to fully develop and deliver the project.

Final thoughts

As we have seen, the MVC pattern is an architectural pattern that is widely used to develop web applications, because of all the features and advantages it has.

However, it can get a bit complex to understand and implement in our own applications, but isn't it worth it to overcome the complexities and enjoy all the benefits it has to offer? Let's discuss in the comments section :)

Written by Jaidev Singh Bhui

MORE POST BY JAIDEV SINGH BHUI →

You might also like

WEB DEVELOPMENT

What is JSON - The Only Guide You Need To Understand JSON

February 26, 2023 - 12 min read

WEB DEVELOPMENT

What exactly is an API - Explained in simple terms

February 22, 2023 - 11 min read

WEB DEVELOPMENT

9 Steps to Become a Full Stack Developer in 2023

January 18, 2023 - 18 min read

WEB DEVELOPMENT

4 Most Important Skills To Become A Backend Developer

October 01, 2022 - 11 min read

WEB DEVELOPMENT

What to Learn: React or Angular?

WEB DEVELOPMENT

Learn Load Balancer Using HAProxy

August 09, 2022 - 12 min read

OLDER POST

4 Most Important Skills To Become A Backend Developer

October 01, 2022 - 11 min read

NEWER POST

10 Important String Methods In Java You Must Know

October 08, 2022 - 14 min read

What do you think?

21 Responses


Upvote


Love


Enlightened

0 Comments Login ▾

G Start the discussion...

LOG IN WITH OR SIGN UP WITH DISQUS ?

Name

♡ Share Best Newest Oldest

Be the first to comment.

Subscribe Privacy Do Not Sell My Data

LATEST POSTS

How to Switch from a non-IT to IT role? | Career Cafe

November 20, 2023 - 3 min read

Better salary, a bigger role or a prestigious company - What to prioritize? | Career Cafe

November 09, 2023 - 3 min read

Billing info update failed.

Is it possible for anyone to get into MAANG Companies? | Career Cafe

November 06, 2023 - 3 min read

Career Cafe - Here's what's brewing at Crio.Do!

November 02, 2023 - 1 min read

Empowering the Future: Celebrating Developers of India

August 11, 2023 - 2 min read

CATEGORIES

AUTOMATION TESTING CAREER GUIDANCE CRIO COMMUNITY

DATA STRUCTURES AND ALGORITHMS DATABASES DEVELOPER ESSENTIALS

IN THE NEWS MINI PROJECTS PROGRAMMING LANGUAGES WEB DEVELOPMENT

WHY CRIO

STAY CONNECTED

NAVIGATION

Mini Projects

Data Structures and Algorithms

Developer Essentials

Web Development

Databases

Programming Languages

Crio Community

In the News

Why Crio

Subscribe to newsletter

Stay up to date! Get all the latest posts delivered straight to your inbox.

Your email address

Subscribe