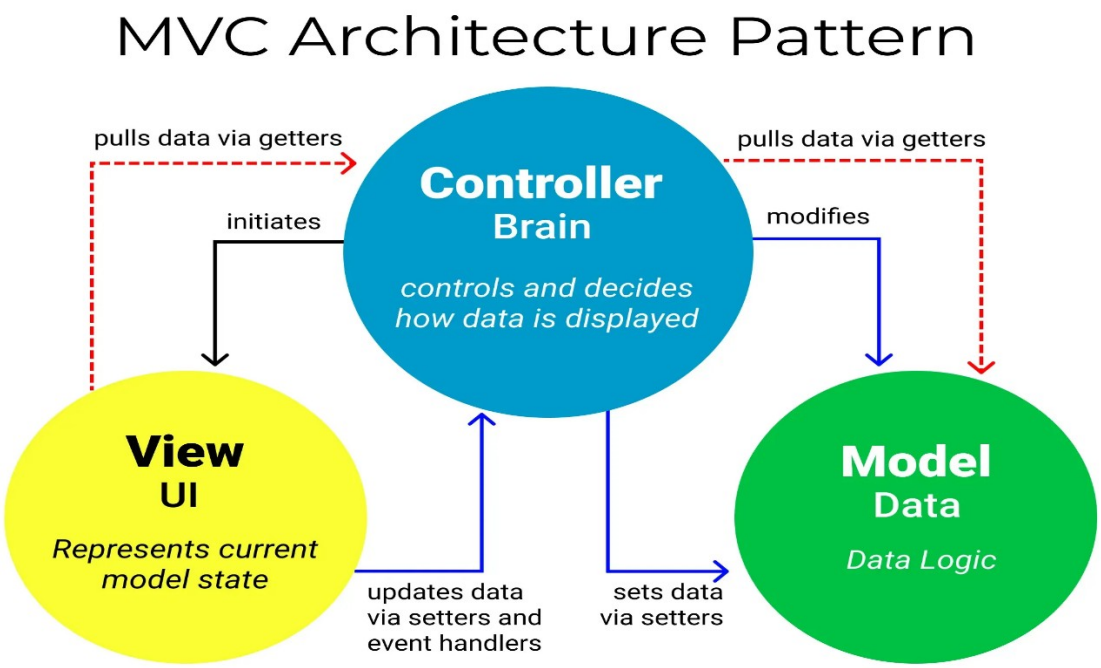
Nhóm 4
Mô hình MVC

MVC (Model-View-Controller) là một mẫu kiến trúc phần mềm được sử dụng rộng rãi trong việc phát triển ứng dụng để tạo ra các giao diện người dùng trực quan và có khả năng tương tác cao. MVC chia một ứng dụng thành 3 phần chính và mỗi phần có một vai trò riêng biệt:

1. Model đại diện cho dữ liệu và quy tắc nghiệp vụ của ứng dụng.
2. View chịu trách nhiệm hiển thị dữ liệu cho người dùng một cách trực quan và tương tác.
3. Controller đóng vai trò là cầu nối giữa Model và View, xử lý các yêu cầu từ người dùng và cập nhật giao diện tương ứng.



Thành phần của MVC

Mô hình MVC gồm 3 thành phần bên trong không thể thiếu khi áp dụng mô hình này:

- Model là lớp đại diện cho dữ liệu của ứng dụng. Nó có thể là một cơ sở dữ liệu, một file cấu hình hoặc một đối tượng phức tạp. Model chịu trách nhiệm lưu trữ, truy xuất và cập nhật dữ liệu.

- View là lớp giao diện người dùng. Nó hiển thị dữ liệu từ Model cho người dùng và cho phép người dùng tương tác với ứng dụng. View thường được xây dựng bằng các ngôn ngữ template như HTML, JSP hoặc React.
- Controller là lớp điều khiển luồng của ứng dụng. Nó nhận các yêu cầu từ người dùng, cập nhật Model và chọn View phù hợp để hiển thị.

Để hiểu rõ hơn về 3 thành phần này, bạn xem qua về mối quan hệ giữa cả 3 như sau:

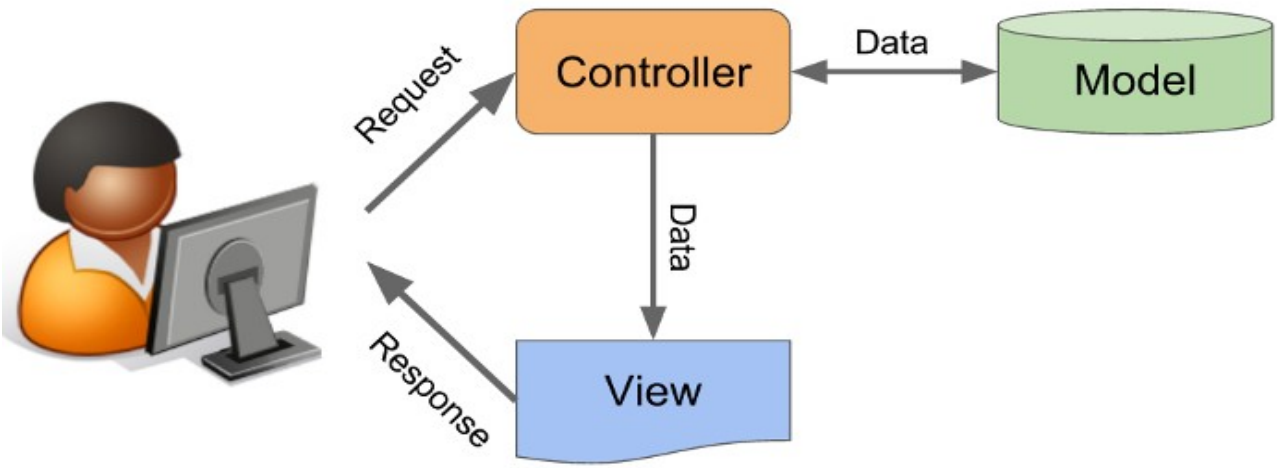
- **Controller:** Cập nhật Model dựa trên các sự kiện từ View và chọn View phù hợp để hiển thị.
- **Model:** Cung cấp dữ liệu cho View và Controller.
- **View:** Hiển thị dữ liệu từ Model và gửi các sự kiện (event) đến Controller.

Luồng xử lý trong MVC

Luồng xử lý trong của mô hình MVC, bạn có thể hình dung cụ thể và chi tiết qua từng bước dưới đây:

- Khi một yêu cầu của từ máy khách (Client) gửi đến Server. Thì bị Controller trong MVC chặn lại để xem đó là URL request hay sự kiện.
- Sau đó, **Controller** xử lý **input** của user rồi giao tiếp với **Model** trong MVC.
- Model chuẩn bị data và gửi lại cho Controller.
- Cuối cùng, khi xử lý xong yêu cầu thì Controller gửi dữ liệu trở lại View và hiển thị cho người dùng trên trình duyệt.

View và Model sẽ được xử lý bởi Controller



Ở đây, **View không giao tiếp trực tiếp với Model**. Sự tương tác giữa **View và Model** sẽ chỉ được xử lý bởi **Controller.**

Ưu và nhược điểm của MVC

Ưu điểm mô hình MVC

- Đầu tiên, nhắc tới ưu điểm mô hình MVC thì đó là băng thông (Bandwidth nhẹ vì không sử dụng viewstate nên khá tiết kiệm băng thông. Việc giảm băng thông giúp website hoạt động ổn định hơn.
- Kiểm tra đơn giản và dễ dàng, kiểm tra lỗi phần mềm trước khi bàn giao lại cho người dùng.
- Một lợi thế chính của MVC là nó tách biệt các phần Model, Controller và View với nhau.

- Sử dụng mô hình MVC chức năng Controller có vai trò quan trọng và tối ưu trên các nền tảng ngôn ngữ khác nhau
- Ta có thể dễ dàng duy trì ứng dụng vì chúng được tách biệt với nhau.
- Có thể chia nhiều developer làm việc cùng một lúc. Công việc của các developer sẽ không ảnh hưởng đến nhau.
- Hỗ trợ TTD (test-driven development). Chúng ta có thể tạo một ứng dụng với unit test và viết các won test case.
- Phiên bản mới nhất của MVC hỗ trợ trợ thiết kế responsive website mặc định và các mẫu cho mobile. Chúng ta có thể tạo công cụ View của riêng mình với cú pháp đơn giản hơn nhiều so với công cụ truyền thống.

Nhược điểm mô hình MVC

Bên cạnh những ưu điểm MVC mang lại thì nó cũng có một số nhược điểm cần khắc phục.

MVC đa phần phù hợp với công ty chuyên về website hoặc các dự án lớn thì mô hình này phù hợp hơn so với với các dự án nhỏ, lẻ vì khá là cồng kềnh và mất thời gian.

Nhược điểm mô hình MVC không hỗ trợ Preview như ASP.NET

- Không thể **Preview** các trang như ASP.NET.
- Khó triển khai.

Vì sao nên sử dụng mô hình MVC?

- Mô hình MVC đã trở thành một tiêu chuẩn trong phát triển web hiện đại. Với cấu trúc phân chia rõ ràng giữa các thành phần mang đến nhiều lợi ích cho người sử dụng. Trong phần này, chúng ta sẽ tìm hiểu về tại sao nên áp dụng mô hình MVC vào dự án.

Quy trình phát triển nhanh hơn

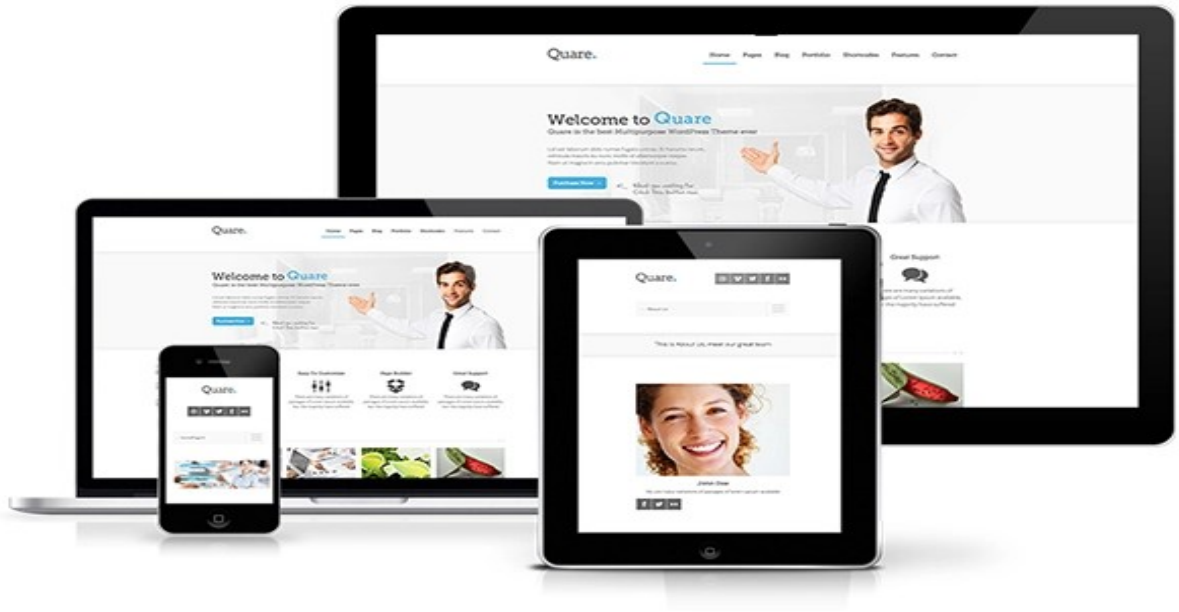
- **MVC** hỗ trợ phát việc phát triển nhanh chóng và song song. Nếu một **mô hình MVC** được dùng để phát triển bất kỳ ứng dụng web cụ thể nào, một lập trình viên có thể làm việc trên **View** và một developer khác có thể làm việc với **Controller** để tạo logic nghiệp vụ cho ứng dụng web đó.
- Do đó, **ứng dụng mô hình MVC** có thể được hoàn thành nhanh hơn ba lần so với các ứng dụng mô hình khác.

Khả năng cung cấp nhiều chế độ view

- Trong **mô hình MVC**, bạn có thể tạo nhiều View cho **chỉ một mô hình**. Ngày nay, nhu cầu có thêm nhiều cách mới để truy cập ứng dụng và đang ngày càng tăng. Do đó, việc sử dụng MVC để phát triển chắc chắn là một giải pháp tuyệt vời.
- Hơn nữa, với phương pháp này, việc nhân bản code rất hạn chế. Vì nó tách biệt dữ liệu và logic nghiệp vụ khỏi màn hình.

Các sửa đổi không ảnh hưởng đến toàn bộ mô hình

- Đối với bất kỳ ứng dụng web nào, người dùng có **xu hướng thay đổi** thường xuyên. Bạn có thể quan sát thông qua những thay đổi thường xuyên về màu sắc, font chữ, bố cục màn hình. Hay là thêm hỗ trợ thiết bị mới cho điện thoại hay máy tính bảng…



Sửa đổi giao diện, font, hình ảnh, .. không ảnh hưởng đến toàn bộ mô hình MVC

Việc thêm một kiểu view mới trong MVC rất đơn giản. Vì phần Model không phụ thuộc vào phần View. Do đó, bất kỳ thay đổi nào trong Model sẽ không ảnh hưởng đến toàn bộ kiến trúc.

MVC Model trả về dữ liệu mà không cần định dạng

MVC pattern có thể trả về dữ liệu mà không cần áp dụng bất kỳ định dạng nào. Do đó, các thành phần giống nhau có thể được sử dụng với bất kỳ giao diện nào.

Ví dụ: tất cả loại dữ liệu đều có thể được định dạng bằng HTML. Ngoài ra, nó cũng có thể được định dạng bằng **Macromedia Flash** hay **Dream Viewer.**

Hỗ trợ kỹ thuật Asynchronous

Kiến trúc MVC có thể được tích hợp với cả **JavaScript Framework**. Có nghĩa là, các ứng dụng MVC có thể hoạt động ngay cả với các file PDF, trình duyệt riêng cho web hay các widget trên desktop.

Ngoài ra, MVC cũng hỗ trợ kỹ thuật Asynchronous, giúp các developer phát triển các ứng dụng có thể load rất nhanh.

Nền tảng MVC thân thiện với SEO

Nền tảng MVC hỗ trợ phát triển các trang web thân thiện với SEO. Bằng nền tảng này, bạn có thể dễ dàng phát triển các URL thân thiện với SEO để tạo ra nhiều lượt truy cập hơn.

Nền tảng MVC thân thiện với SEO

Những ngôn ngữ như JavaScript hay jQuery có thể được tích hợp với MVC. Từ đó phát triển nhiều ứng dụng web giàu tính năng, đặc biệt là với **mô hình MVC trong Java**.

Ứng dụng mô hình MVC vào lập trình như thế nào?

Ngôn ngữ lập trình và framework mà bạn dùng phụ thuộc nhiều hơn vào mục đích nghề nghiệp. Nhưng lập trình MVC dưới dạng kiến trúc sẽ luôn là một lựa chọn khả thi để phát triển nghề nghiệp của bạn.

Ví dụ, mọi người đang dần chuyển từ Dotnet MVC sang Dotnet Core. Nhưng hiện nay, vẫn còn nhu cầu về Django cũng sử dụng MVC.

Các kỹ năng cần thiết khi sử dụng mô hình kiến trúc

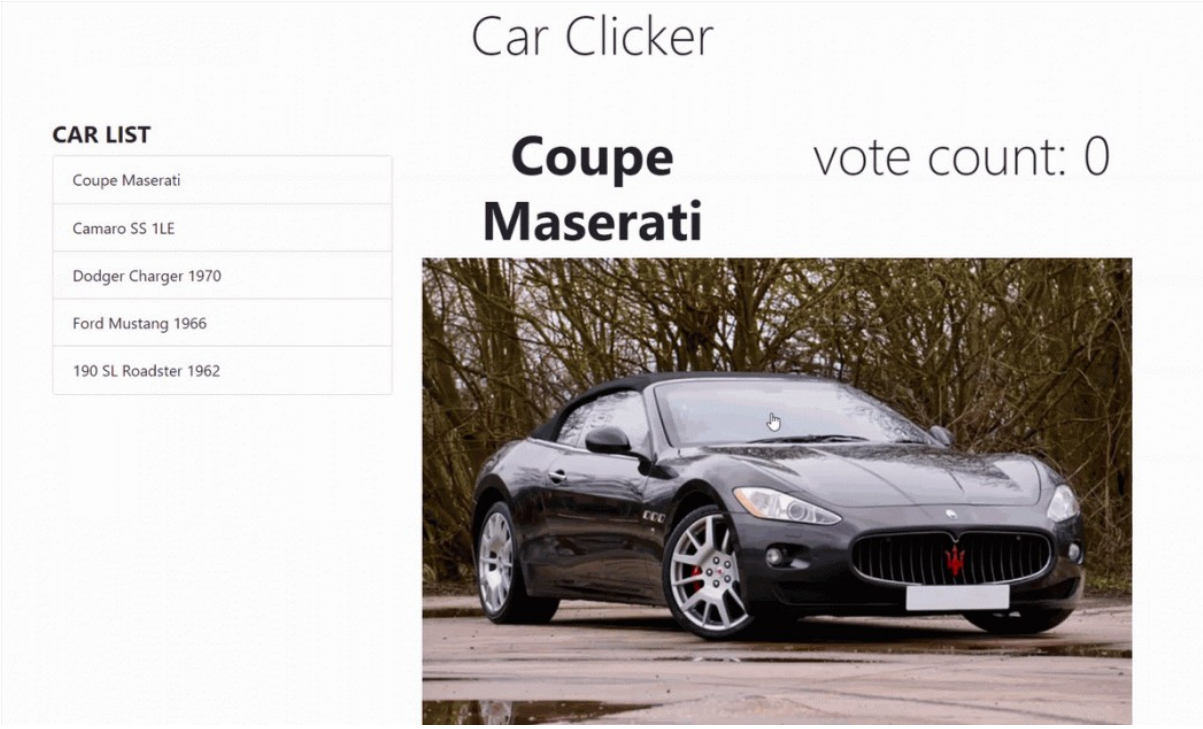
Khi bạn đã hiểu **MVC là gì**, thì nó giúp ích cho bạn có được một kiến thức cần thiết và nó là một trong các kỹ năng cần thiết khi bạn làm lập trình. Để sử dụng tốt **mô hình kiến trúc** này cần có các kỹ năng và kiến thức sau:

- Hiểu rõ về **mô hình kiến trúc phần mềm** (MVC).
- Hiểu cách sử dụng framework.
- Có kiến thức cơ bản về lập trình hướng đối tượng.
- Có khả năng logic và hiển thị nội dung, đảm bảo được rằng Model và View độc lập với nhau.

Cách sử dụng MVC

Ví dụ minh họa: Ứng dụng My Car Clicker là một biến thể của Cat Clicker nổi tiếng. Dưới đây là một số khác biệt chính trong ứng dụng này:

- Nhiều mẫu xe hơi được liệt kê.
- Có nhiều bộ đếm click chuột.
- Nó chỉ hiển thị chiếc xe đã chọn.



Model (dữ liệu)

Nhiệm vụ chính của thành phần này chỉ đơn giản là quản lý dữ liệu. Model sẽ chịu trách nhiệm quản lý dữ liệu từ cơ sở dữ liệu, API hay JSON. Trong ứng dụng Car Clicker, Model chứa các đối tượng (ô tô) với tất cả thông tin (dữ liệu) cần thiết cho ứng dụng.

Nó cũng quản lý chiếc xe hiện tại đang được hiển thị với biến ban đầu được đặt là null.

```
const model = {
    currentCar: null,
    cars: [
        {
            clickCount: 0,
            name: 'Coupe Maserati',
            imgSrc: 'img/black-convertible-coupe.jpg',
        },
        {
            clickCount: 0,
            name: 'Camaro SS 1LE',
            imgSrc: 'img/chevrolet-camaro.jpg',
        },
        {
            clickCount: 0,
            name: 'Dodger Charger 1970',
            imgSrc: 'img/dodge-charger.jpg',
        },
        {
            clickCount: 0,
            name: 'Ford Mustang 1966',
            imgSrc: 'img/ford-mustang.jpg',
        },
        {
            clickCount: 0,
            name: '190 SL Roadster 1962',
            imgSrc: 'img/mercedes-benz.jpg',
        },
    ],
};
```

View (Giao diện người dùng)

Nhiệm vụ của **View** là quyết định xem người dùng sẽ nhìn thấy gì trên màn hình của họ?

Ứng dụng Car Clicker có hai chế độ xem: carListView và CarView. Cả hai chế độ xem đều có 2 chức năng quan trọng là xác định những gì mỗi chế độ xem muốn khởi tạo và hiển thị. Chức năng này sẽ quyết định câu hỏi là người dùng sẽ nhìn gì và như thế nào trên màn hình.

- **carListView**

```
const carListView = {
    init() {
        // store the DOM element for easy access later
        this.carListElem = document.getElementById('car-list');

        // render this view (update the DOM elements with the right values)
        this.render();
    },

    render() {
        let car;
        let elem;
        let i;
        // get the cars to be render from the controller
        const cars = controller.getCars();

        // to make sure the list is empty before rendering
        this.carListElem.innerHTML = '';

        // loop over the cars array
        for(let i = 0; i < cars.length; i++) {
            // this is the car we've currently looping over
            car = cars[i];

            // make a new car list item and set its text
            elem = document.createElement('li');
            elem.className = 'list-group-item d-flex justify-content-between lh-condensed';
            elem.style.cursor = 'pointer';
            elem.textContent = car.name;
            elem.addEventListener(
                'click',
                (function(carCopy) {
                    return function() {
                        controller.setCurrentCar(carCopy);
                        carView.render();
                    };
                })(car)
            );
            // finally, add the element to the list
            this.carListElem.appendChild(elem);
```

- **CarView**

```
const carView = {
    init() {
        // store pointers to the DOM elements for easy access later
        this.carElem = document.getElementById('car');
        this.carNameElem = document.getElementById('car-name');
        this.carImageElem = document.getElementById('car-img');
        this.countElem = document.getElementById('car-count');
        this.elCount = document.getElementById('elCount');


        // on click, increment the current car's counter
        this.carImageElem.addEventListener('click', this.handleClick);

        // render this view (update the DOM elements with the right values)
        this.render();
    },

    handleClick() {
        return controller.incrementCounter();
    },

    render() {
        // update the DOM elements with values from the current car
        const currentCar = controller.getCurrentCar();
        this.countElem.textContent = currentCar.clickCount;
        this.carNameElem.textContent = currentCar.name;
        this.carImageElem.src = currentCar.imgSrc;
        this.carImageElem.style.cursor = 'pointer';
    },
};
```

Controller

Chức năng của controller là lấy, sửa đổi và cung cấp dữ liệu cho người dùng. Về cơ bản, controller là liên kết giữa **View** và **Model**.

Thông qua các hàm **getter** và **setter**, **controller** lấy dữ liệu từ **model** và khởi tạo **view**. Nếu có bất kỳ cập nhật nào từ view, nó sẽ sửa đổi dữ liệu bằng hàm setter.

```
const controller = {
    init() {
        // set the current car to the first one in the list
        model.currentCar = model.cars[0];

        // tell the views to initialize
        carListView.init();
        carView.init();
    },

    getCurrentCar() {
        return model.currentCar;
    },

    getCars() {
        return model.cars;
    },

    // set the currently selected car to the object that's passed in
    setCurrentCar(car) {
        model.currentCar = car;
    },

    // increment the counter for the currently-selected car
    incrementCounter() {
        model.currentCar.clickCount++;
        carView.render();
    },
};

// Let's goooo!
controller.init();
```