1. Mục tiêu

- Vận dụng được các đối tượng cơ bản trong Javascript như: Array, String, Date,
- Javascript json, …
- Sử dụng ckeditor

2. Tóm tắt lý thuyết

Khai báo đối tượng:

```
[var] tên_biến = new Tên_đối_tượng([giá trị khởi tạo]);
```

Sử dụng đối tượng:

- Truy xuất thuộc tính của đối tượng: `tên_đối_tượng.tên_thuộc_tính;`
- Gọi phương thức của đối tượng: `tên_đối_tượng.tên_phương_thức([các giá trị khởi tạo];`

2.1 Object là gì?

Trong JavaScript, một object là một thực thể độc lập, với thuộc tính và kiểu. Ví dụ cái tách (cái ly) là một object có các thuộc tính như: màu sắc, thiết kế, trọng lượng, chất liệu tạo ra nó,... Tương tự như vậy, JavaScript Objects cũng có những thuộc tính định nghĩa nên đặc tính của nó.

Đối tượng trong Javascript là một tập hợp các cặp ***[khóa - giá trị]***, tương tự như bản đồ, từ điển, hay hash-table trong ngôn ngữ lập trình khác.

- Tên thuộc tính là một giá trị duy nhất có thể là một chuỗi và trỏ đến một giá trị
- Giá trị thuộc tính có thể là bất kỳ giá trị nào, bao gồm các đối tượng khác hoặc các hàm, được liên kết với tên/khóa

2.2 Tạo đối tượng.

Ta có 3 cách để khai báo đối tượng trong Javascript:

- *Sử dụng từ khóa `{}`*
- *Sử dụng từ khóa new `Object()`*
- *Sử dụng phương thức `static`*

Ví dụ:

```
// literal
const dog = { }

// constructor
const cat = new Object();

// static method
const horse = Object.create({ })
```

2.3 Get and Set Properties

Như ví dụ trên chúng ta đã có 1 đối tượng trống, chúng ta cần thêm các thuộc tính vào cho nó bằng cách sử dụng các trình truy cập (accessors).

Tên thuộc tính hợp lệ bao gồm: chữ cái, số, kí tự,... có thể ép thành một chuỗi, nhưng không được sử dụng các từ dành riêng như function, var, return, .v.v.

```
get = object.property;
object.property = set;
```

⇨ Kể từ ES6, chúng ta có một cách viết tắt thuận tiện để thiết lập các thuộc tính:

```
let hello;
let world;

// // Old way
// const obj = {
//     hello: hello,
//     world: world
// }

// Modern way
const obj = {
    hello,
    world,
}
```

Sử dụng một biến hoặc biểu thức làm tên thuộc tính bằng cách đặt nó trong dấu ngoặc [ ] - đây được gọi là thuộc tính được tính toán.

```
const x = 'khoa';

const obj = {
  [x]: 7
}

obj.khoa // 7
```

Thuộc tính của đối tượng có thể được xóa với từ khóa delete.

```
delete obj.hello;
delete obj.world;
```

2.4 Phương thức trong đối tượng

Một đối tượng ngoài các thuộc tính, nó còn chứa hàm gọi là phương thức.

Một phương thức là một hàm liên kết với một object, hay có thể nói phương thức là một thuộc tính của object có giá trị là một hàm.

Phương thức được định nghĩa giống như cách định nghĩa hàm, ngoài trừ việc chúng phải được gán như là thuộc tính của một object.

```
//Hàm khởi tạo đối tượng
function person(name, age) {
    this.name = name;
    this.age = age;
    this.changeName = function (name) {
      this.name = name;
    }
  }

  //Tạo đối tượng
  var p = new person("Khoa", 19);

  p.changeName("Vân");
  //Giờ p.name bằng "Vân"
```

Các phương thức cũng có thể được định nghĩa bên ngoài hàm khởi tạo.

```
function person(name, age) {
    this.name= name;
    this.age = age;
    this.yearOfBirth = bornYear; //Gán phương thức bên ngoài
  }

  //Hàm bên ngoài hàm tạo, hàm này được gán vào đối tượng qua hàm tạo ở trên
  function bornYear() {
    return 2020 - this.age;
  }

  var p = new person("Khoa", 19);
  document.write(p.yearOfBirth());
  // Outputs 2001
```

2.5 Setter & Getter

Một thuộc tính của đối tượng còn được thiết lập là hàm setter hoặc getter. Nếu là setter nó chỉ được gọi qua toán tử gán giá trị cho nó, nếu là getter thì chỉ được gọi khi truy cập lấy giá trị thuộc tính.

Hàm setter định nghĩa bằng cách cho thêm set, hàm getter định nghĩa bằng cách cho thêm get.

```
var obj = {
    age: 0,

    set ageInfo(age) {                              //Định nghĩa setter
        console.log('setter - ' + age);
        this.age = age;
    },

    get ageInfo() {                                 //Định nghĩa getter
        console.log('getter');
        return "Thông tin tuổi: " + this.age;
    }
};

obj.ageInfo = 25;           //Gán -> Tự động gọi settter
alert(obj.ageInfo);         //Không phải gán -> Tự động gọi getter
```

⇨ Trong trường hợp muốn định nghĩa setter / getter trong hàm tạo đối tượng thì ta cần định nghĩa theo nguyên tắc thêm một thuộc tính vào đối tượng đã có với lệnh `Object.defineProperty`

```
//Một đối tượng đã có tên ob, thêm cho nó setter, getter có tên namepro
Object.defineProperty(ob, 'namepro', {
    set: function(x) {
        //code setter ở đây
      },
    get: function() {
          //code getter ở đây
      }

});
```

Từ 2 ví dụ trên, ta định nghĩa lại trong hàm tạo:

```
function person(age) {
    this.age = 0;
    Object.defineProperty(this, 'ageInfo', {
        set : function (age) {
            console.log('setter - ' + age);
            this.age = age;
        },
        get : function () {
            console.log('getter');
            return "Thông tin tuổi: " + this.age;
        }
    });
 }

var obj = new person(0);

obj.ageInfo = 25;
alert(obj.ageInfo);
```

2.6 Sử dụng this để tham chiếu đối tượng

- Trong JavaScript, từ khóa this đại diện cho một object. Object đại diện cho chủ thể trong ngữ cảnh và nó phụ thuộc vào lúc run-time chứ không phải lúc khởi tạo. this đóng vai trò là một con trỏ, trỏ đến chính object gọi hàm đó.

```
const person = {
    id: 1,
    firstName: "Nguyen",
    lastName: "Khoa",
    getFullName: function() {
        console.log(this.firstName + " " + this.lastName);
    }
};

person.getFullName();
//-->Output: Nguyen Khoa
```

- Trường hợp tiếp theo ta khai báo biến global và function global. Vì thế toàn bộ các biến và các function đều được nằm trong một object lớn có tên là window.

```
let firstName = "Nguyen", lastName = "Trung";
// biến firstName và lastName lúc này thuộc object window nhé
function getFullName() {
    console.log(this.firstName + " " + this.lastName);
}

window.getFullName();
//-->Output: Nguyen Trung
getFullName();
//-->Output: Nguyen Trung
```

- ***window.getFullName***() - this ở đây trỏ đến object gọi function getFullName đó là object window
- ***getFullName***() - Việc gọi function như vầy thì cũng do object window gọi đến function getFullName() mà thôi.

- Về bản chất thì một function bất kỳ đều có property, giống như object vậy. Khi gọi function, sẽ có property this chứa item của object đang gọi đến function đó.

```
"use strict"

const person = {
    id: 1,
    name: "Nguyen Van A",
    gender: "male",
    age: 18
};

function actionObj(){
    console.log(this.person);
};

actionObj();
```

⇨ Ở đây this trong một function (cụ thể là actionObj()) nó sẽ chứa các item của object mà gọi function actionObj(). Và ta cũng cần this để có thể truy cập ngược lại vào các property và method của object mà gọi fuction actionObj() trên.

- Lưu ý về this cần nhớ:
  - Từ khóa this đại diện cho ngữ cảnh hay ***context*** của nơi mà function dùng nó để gọi.
  - Từ khóa this có hai loại ***context*** như sau: ***Object*** chứa các phương thức được gọi hoặc là ***Global***.

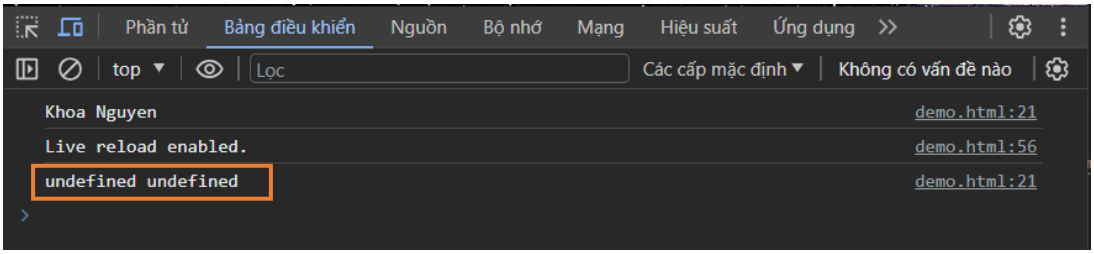
- **Các trường hợp dễ nhầm lẫn với "this"**

  - *Function truyền vào như một callback function.*

Ví dụ người dùng muốn click vào một button và sẽ lấy được full name của user. Việc này khá đơn giản, chỉ việc truyền function getFullName() vào như một callback cho function onClick().

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>"this" in JavaScript</title>
</head>

<body>
    <button>show full name</button>

    <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.4.1/jquery.min.js"></script>
    <script type="text/javascript">
        var person = {
            id: 1,
            firstName: 'Khoa',
            lastName: 'Nguyen',
            getFullName: function () {
                console.log(this.firstName + ' ' + this.lastName);
            }
        };
        person.getFullName();

        $('button').click(person.getFullName);
    </script>
</body>

</html>
```

Tuy nhiên chỉ có gọi function person.getFullName() là chạy đúng còn khi ta click button show full name thì cho kết quả lại là ***undefined undefined.***



Lý do vì this ở đây chính là của button ta click vào, do đó this không có trường ***firstName*** và ***lastName***. Để fix lỗi trên ta chuyển thành dạng ***anonymus function*** hoặc dùng function bind để xác định tham số this cho function truyền vào là được.

```
<script type="text/javascript">
    var person = {
        id: 1,
        firstName: 'Khoa',
        lastName: 'Nguyen',
        getFullName: function () {
            console.log(this.firstName + ' ' + this.lastName);
        }
    };
    person.getFullName();

    // Dùng anonymous function
    $('button').click(function(){ person.getFullName()});
</script>
```

Hoặc

```
<script type="text/javascript">
    var person = {
        id: 1,
        firstName: 'Khoa',
        lastName: 'Nguyen',
        getFullName: function () {
            console.log(this.firstName + ' ' + this.lastName);
        }
    };
    person.getFullName();

    // Dùng bind
    $('button').click(person.getFullName.bind(person));
</script>
```

- *Sử dụng "this" trong anonymous function.*

Ví dụ ta có một object person có chứa danh sách các best friend của "Nguyen Khoa", ta muốn viết một function để hiện ra các tên của danh sách đó, ta sẽ viết như sau:

```
const person = {
    name: "Nguyen Khoa",
    bestFriends: ["Huynh Lap", "An Nhien", "Nguyen Du", "Tran Van Duc"],
    showBestFriend: function() {
        for(let i = 0; i < this.bestFriends.length; i++) {
            console.log(this.name + " has a friend name: " + this.bestFriends[i]);
        }
    },
    showBestFriendThis: function() {
        this.bestFriends.forEach(function(fr) {
            console.log(this.name + " has a friend name: " + fr);
        });
    }
};

person.showBestFriend();
//-->Output: Nguyen Khoa has a best friend name: Huynh Lap
//           Nguyen Khoa has a best friend name: An Nhien
//           Nguyen Khoa has a best friend name: Nguyen Du
//           Nguyen Khoa has a best friend name: Tran Van Duc

person.showBestFriendThis();
//-->Output: has best a friend name: Huynh Lap
//           has best a friend name: An Nhien
//           has best a friend name: Nguyen Du
//           has best a friend name: Tran Van Duc
```

⇨ Ta thấy function showBestFriend() dùng for thường thì chạy đúng như mong muốn, nhưng function showBestFriendThis() dùng forEach và truyền vào một ***anonymous function***, tuy nhiên nó cho ra kết quả không như mong muốn.

⇨ Bởi vì this lúc này là this của ***object window***, do đó mà trường name bị ***undefined***. Để fix trường hợp này ta tạo một biến để gán giá trị this vào và truy xuất đến giá trị đó trong ***anonymous function*** là được.

```
const person = {
    name: "Nguyen Khoa",
    bestFriends: ["Huynh Lap", "An Nhien", "Nguyen Du", "Tran Van Duc"],

    showBestFriendThis: function() {
        let objPersion = this; //Fix như vầy
        this.bestFriends.forEach(function(fr) {
            console.log(objPersion.name + " has a best friend name: " + fr);
        });
    }
};

person.showBestFriendThis();
//-->Output: Nguyen Khoa has a best friend name: Huynh Lap
//           Nguyen Khoa has a best friend name: An Nhien
//           Nguyen Khoa has a best friend name: Nguyen Du
//           Nguyen Khoa has a best friend name: Tran Van Duc
```

- *Function được gán vào một biến.*

Trường hợp khi ta gán một function vào một biến và ta nghĩ nó sẽ chạy bình thường, và gọi nó ra sử dụng. Nhưng kết quả không như mong đợi.

```
const person = {
    id: 1,
    firstName: "Khoa",
    lastName: "Nguyen",
    getFullName: function() {
        console.log(this.firstName + " " + this.lastName);
    }
};

person.getFullName();
//-->Output: Khoa Nguyen

const showFullName = person.getFullName; //Gán function vào biến showFullName
showFullName();
//-->Output: undefined undefined
```

Cách fix là ta dùng bind() như trường hợp ở trên là được.

```
const person = {
    id: 1,
    firstName: "Khoa",
    lastName: "Nguyen",
    getFullName: function() {
        console.log(this.firstName + " " + this.lastName);
    }
};

const showFullName = person.getFullName.bind(person);
showFullName();
//-->Output: Khoa Nguyen
```

2.7 So sánh Objects

Trong JavaScript, **những object là kiểu tham chiếu**. Hai đối tượng tách biệt không bao giờ bằng nhau, thậm chí nếu chúng có cùng những thuộc tính. Chỉ khi nó so sánh với chính nó thì kết quả mới là true.

```
// Hai biến, hai đối tượng riêng biệt có cùng thuộc tính
var fruit = {name: 'apple'};
var fruitbear = {name: 'apple'};

fruit == fruitbear; // return false
fruit === fruitbear; // return false
```

```
// Hai biến, một đối tượng
var fruit = {name: 'apple'};
var fruitbear = fruit;  // gán tham chiếu đối tượng trái cây cho Fruitbear

// ở đây fruit và fruitbear đang trỏ đến cùng một đối tượng
fruit == fruitbear; // return true
fruit === fruitbear; // return true
```

```
fruit.name = 'grape';
console.log(fruitbear);
// yields { name: "grape" } instead of { name: "apple" }
```

3. Các đối tượng cơ bản

3.1 String

Khai báo: `var ten_bien = new String;`

hay `var ten_bien = new String("giá trị khởi tạo");`

Thuộc tính: `length` cho biết tổng số ký tự của chuỗi.

Một số phương thức cơ bản:

- `charAt(i)`: trả về ký tự thứ i trong chuỗi
- `concat(s1, s2,…, sn)`: nối các chuỗi s_1, s_2,…, s_n thành 1 chuỗi
- `indexOf(s, [start])`: trả về vị trí đầu tiên tìm được chuỗi s tính từ start.
- `lastIndexOf(subString, [start])`: trả về vị trí cuối cùng tìm được chuỗi s tính từ start.
- `split("separator", [n])`: tách chuỗi dựa vào ký tự separator, n số phần tử trả về.
- `substr(start, [n])`: trả về chuỗi con bắt đầu từ start và có chiều dài là n
- `substring(start,end)`: trả về chuỗi con bắt đầu từ start đến end-1

3.2 Boolean

Dùng để xử lý giá trị luận lý đúng (true) hay sai (false).

Khai báo:

var ten_bien = new Boolean;

var ten_bien = new Boolean(giá_trị_khởi_tạo);

Phương thức: `toString()` chuyển sang kiểu chuỗi.

3.3 Date

Date là kiểu dữ liệu thời gian.

Khai báo:

var ten_bien = new Date();

var ten_bien = new Date(năm, tháng, ngày); //tháng tính từ 0

Một số phương thức cơ bản: `getDate()`, `getDay()`, `getMonth()`, `getYear()`.

3.4 Math

Đối tượng Math dùng để tính toán.

Các thuộc tính: PI, SQRT1_2, SQRT

Các phương thức: `abs(x)`, `ceil(x)`, `floor(x)`, `max(a,b)`, `min(a,b)`, `pow(x,y)`, `random()`,`round(x)`, `sqrt(x)`.

Sử dụng đối tượng Math:

- Sử dụng thuộc tính của Math: Math.tên_thuộc_tính. Ví dụ như Math.PI
- Gọi phương thức của Math: Math.`tên_phương_thức`(tham số)

3.5 Array

Array dùng để lưu trữ tập các giá trị.

Khai báo:

var ten_bien = new Array();

var ten_bien = new Array(giá trị 1, giá trị 2, …);

var ten_bien = [giá trị 1, giá trị 2, …];

4. Constructor là gì?

Phương thức `constructor` là một phương thức đặc biệt dùng để khởi tạo 1 object và được tạo ở trong `class`.

```
constructor([arguments]) { ... }
```

Ví dụ:

```
class Polygon {
    constructor() {
      this.name = 'Polygon';
    }
  }

  const poly1 = new Polygon();

  console.log(poly1.name);
  // expected output: "Polygon"
```

Chỉ có duy nhất 1 phương thức đặc biệt tên là "`constructor`" ở trong class. Có nhiều hơn 1 phương thức `constructor` ở trong class thì sẽ gây ra lỗi SyntaxError.

Một `constructor` có thể sử dụng từ khóa `super` để gọi đến `constructor` của class cha.

⇨ Nếu ta không chỉ định 1 phương thức `constructor` thì `constructor` mặc định sẽ được sử dụng.

Ví dụ sau sử dụng phương thức `constructor`:

```
class Square extends Polygon {
    constructor(length) {
      // Ở đây, nó gọi hàm tạo của lớp cha với độ dài
      // được cung cấp cho chiều rộng và chiều cao của Polygon
      super(length, length);
      // Lưu ý: Trong các lớp dẫn xuất, super() phải được gọi trước bạn
      // có thể sử dụng 'this'. Việc bỏ phần này ra sẽ gây ra lỗi tham chiếu.
      this.name = 'Square';
    }

    get area() {
      return this.height * this.width;
    }

    set area(value) {
      this.area = value;
    }
  }
```

5. Javascript Json.

JSON viết tắt bởi JavaScript Object Notation. Dịch nôm na là "Kí hiệu object trong JavaScript". Xét ví dụ sau:

Ví dụ trên cũng chính là biểu diễn cho một Object trong JavaScript.

Trong đó, gồm có hai thành phần:

- keys: type, brand, operating system, graphic card
- values: laptop, Sony, Windows 7, NVIDIA

a. Áp dụng

- JSON thường được dùng trong cơ sở dữ liệu NoSQL (ví dụ như MongoDB) và là một chuẩn giao tiếp trên web.
- Khi một client gửi request lên server thì server có thể gửi kết quả trả về dạng JSON (hoặc XML).
- Client bóc tách kết quả đó và dựa vào key để lấy ra thông tin cần thiết.

Việc sử dụng JSON thay vì XML giúp giảm thời gian truy xuất dữ liệu và giảm dung lượng gói tin. Ở ví dụ trên nếu dùng XML để biểu diễn thì nó sẽ như sau:

<type> laptop </type>

<brand> Sony </brand>

<operatin_system> Windows 7 </operating_system>

<graphic_card> NVIDIA </graphic_card>

⇨ Rõ ràng, XML phức tạp hơn JSON ở chỗ XML cần phải có 2 thành phần <**thẻ đóng**> và <**thẻ mở**> để xác định một thuộc tính. Trong khi JSON thì chỉ cần 1 thành phần là **key**.

b. So sách với JavaScript Object

Mặc dù JSON rất giống JavaScript Object nhưng vẫn có một số giới hạn khác như:

- *Key*: luôn luôn phải được đóng gói trong cặp dấu ngoặc kép, không phải ngoặc đơn, cũng không được phép là biến số (variable)

- *Value*: Chỉ được phép là những dữ liệu cơ bản như: numbers, strings, Boolean, array, object, null, không được phép là function, date, undefined hoặc là một biểu thức tính toán.

⇨ Ta có thể gọi đây chính là phiên bản rút gọn của JavaScript Object.

c. Cách sử dụng

JavaScript cung cấp sẵn cho chúng ta hai hàm số là: JSON.stringify và JSON.parse:

- JSON.stringify dùng để convert một JavaScript Object thành JSON string.
- JSON.parse dùng để convert string biểu diễn JSON thành JavaScript Object.

```
var string = JSON.stringify({name: "X", born: 1990});
console.log(string);
// => {"name":"X","born":1990}

var obj = JSON.parse(string);
console.log(obj.name);
// => X
console.log(obj.born);
// => 1990
```

⇨ Điều đó đồng nghĩa với việc nếu ta muốn chỉnh sửa JSON thì có thể convert nó thành Object để sửa đổi các thuộc tính, giá trị. Sau đó convert JavaScript Object đó ngược lại thành JSON.

6. Sử dụng ckeditor

a. CKeditor là gì?

CKEditor (còn gọi là FCKeditor) là một trình soạn thảo mã nguồn mở theo kiểu WYSIWYG (tay làm - mắt thấy) của CKSource. Chương trình này có thể tích hợp vào các web site mà không cần cài đặt. Phiên bản đầu tiên được phát hành năm 2003 và đến nay được rất nhiều người sử dụng.

b. Dowload

Bạn có thể cài CKeditor với Package Managers:

- npm: npm install ckeditor --save
- bower: bower install ckeditor --save

c. Sử dụng CKeditor

Thay thế texarea

Có 2 cách để thay thế phần tử textarea bằng CKeditor: sử dụng javascript hoặc sử dụng class.

- *Sử dụng javascript:* Việc sử dụng javascript để thay thế phần tử textarea thành CKeditor cực kì đơn giản. Ta chỉ cần sử dùng hàm

CKEditor.replace(id) để thay thế. id chính là id của thẻ textarea cần thay thế.

```
<textarea name="editor" class="ckeditor" id="editor"></textarea>
<script src="{!! asset('ckeditor-dev/ckeditor.js') !!}"></script>
<script>
    CKEditor.replace('editor');
</script>
```

Lưu ý: Thẻ script include file **ckeditor.js** phải được đặt trên thẻ chạy lệnh `CKEditor.replace('editor')`.

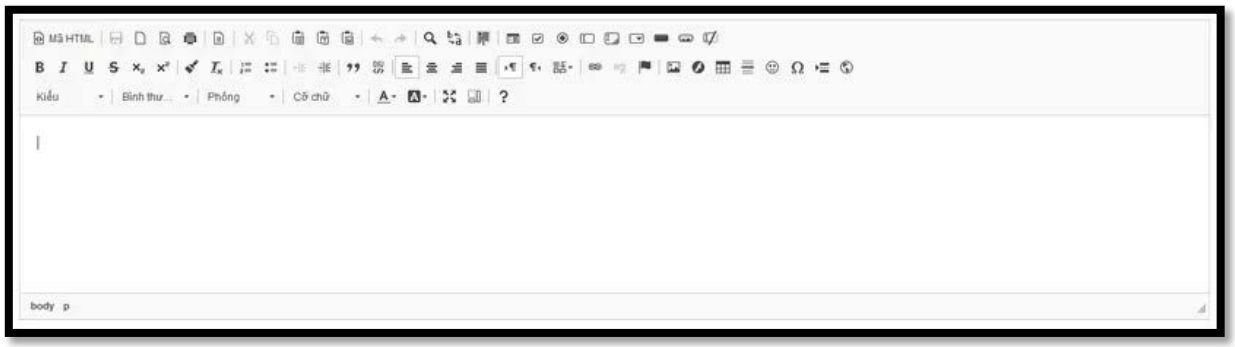
- *Sử dụng class:* Cách này còn đơn giản hơn cả cách trên nữa. Ta chỉ cần thêm class ckeditor vào trong thẻ textarea là có thể thay thế được CKEditor vào rồi.

```
<textarea class="ckeditor" name="editor" cols="80" rows="10"></textarea>
```

Lưu ý:

- Thẻ textarea phải được đặt tên để xác định thì mới thay thế được.
- Đoạn chèn script **ckeditor.js** phải được đặt trên tất cả thẻ textarea và script chèn file **ckeditor.js** cần đặt trong thẻ <head></head>.

Kết quả:



Tùy chọn các plugin cho CKEditor

CKEditor mặc định có rất nhiều công cụ. Để thêm hoặc loại bỏ những công cụ này thì ta phải cấu hình ở file **config.js** có vị trí cùng thư mục với file **ckeditor.js**. Ngoài ra còn có thể định file cấu hình bằng lệnh `CKEDITOR.config.customConfig = 'file cấu hình của mình';`

Tích hợp upload file vào CKEditor

1. **Tạo view**

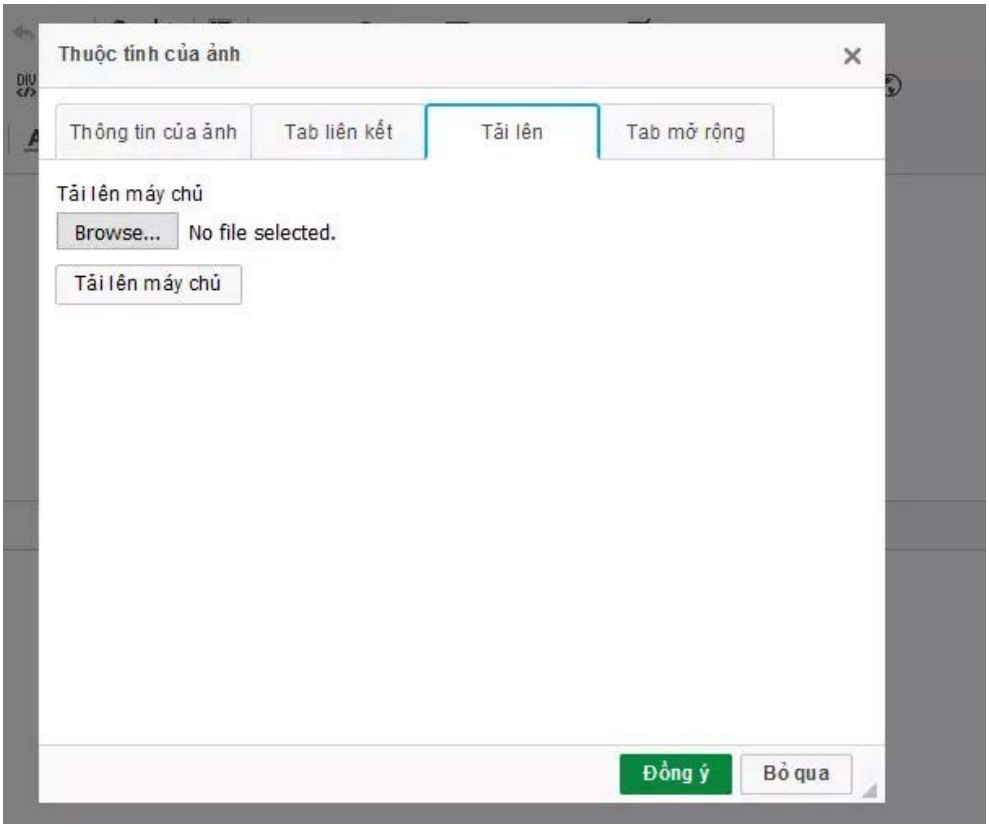
Việc đầu tiên cần làm là tạo 1 file view có cài CKEditor vào đó. Để sử dụng được chức năng upload image trong CK thì ta cần thêm một dòng script cấu hình url:

```
CKEDITOR.config.filebrowserImageUploadUrl = '{!! route('uploadPhoto').'?_token='.csrf_token() !!}';
```

Chúng ta sẽ truyền vào route upload file (đây là route sẽ nhận file và sử lý khi ấn nút submit) và cần thêm 1 thuộc tính _token bằng csrf_token() để không bị lỗi thiếu token.

Toàn bộ code sẽ như sau:

```
<html>
    <head>
        <meta charset="UTF-8">
        <title>Test CKeditor</title>
        <script src="{!! asset('ckeditor-dev/ckeditor.js') !!}"></script>
    </head>
    <body>
        <textarea name="editor" class="ckeditor" id="editor"></textarea>
        <script>
            CKEDITOR.config.filebrowserImageUploadUrl = '{!!
route('uploadPhoto').'?_token='.csrf_token() !!}';
        </script>
    </body>
</html>
```



2. Tạo controller

Trước hết ta cần phải biết CKEditor sẽ gửi request gì thì mới có thể xử lý được. Sau khi bấm nút submit, CKEditor sẽ gửi 1 request với phương thức POST với khá nhiều biến. Ta chỉ cần quan tâm đến 2 biến chính này thôi:

- `upload`: đây chính là file ta upload lên
- `CKEditorFuncNum`: Đây là mã hàm sẽ chạy khi nhận được kết quả trả về.

Ta sẽ sử dụng biến upload để upload ảnh lên flickr. Sau khi upload lên flickr xong ta không return mảng như lúc trước nữa sẽ return 1 view với các tham số như sau:

```
return view('uploadCKEditor', [
    'CKEditorFuncNum' => $request->CKEditorFuncNum,
    'data' => [
        'url' => $url,
        'message' => $message,
    ],
]);
```

Nội dung view:

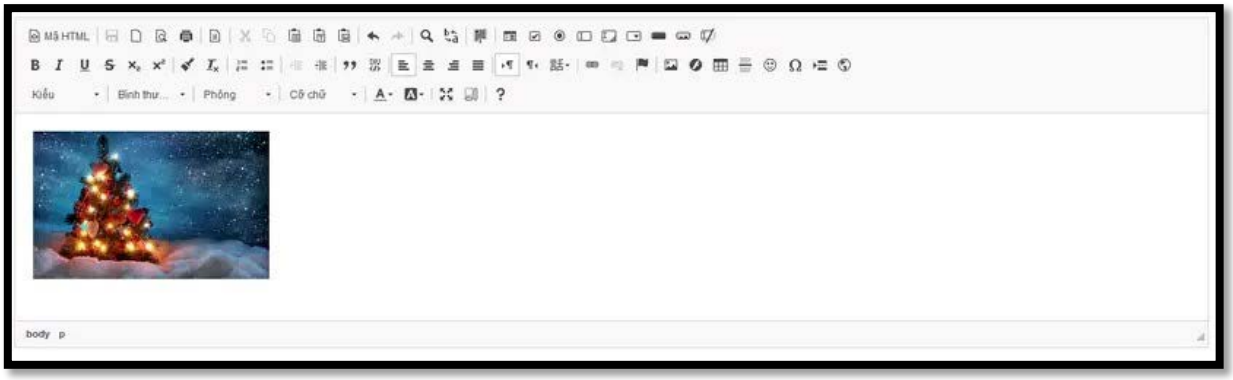
```
<html>
    <head>
        <meta charset="UTF-8">
    </head>
    <body>
        <script type='text/javascript'>
            window.parent.CKEDITOR.tools.callFunction(
                {!! $CKEditorFuncNum !!},
                '{!! $data['url'] !!}',
                '{!! $data['message'] !!}'
            );
        </script>
    </body>
</html>
```

Trong view này ta sẽ chạy một hàm của CKEditor

`window.opener.CKEDITOR.tools.callFunction(funcNum, fileUrl [, data]);`

Các tham số cần truyền vào:

- Mã hàm là $CKEditorFuncNum đã truyền vào từ request.
- fileUrl là url trả về
- data có thể có hoặc không. Nó có thể là hàm hoặc chuỗi. Nếu nó là chuỗi kí tự thì nó sẽ hiện lên màn hình giống như ta gọi hàm alert() của javascript.



7. Thời gian thực hành

120 phút

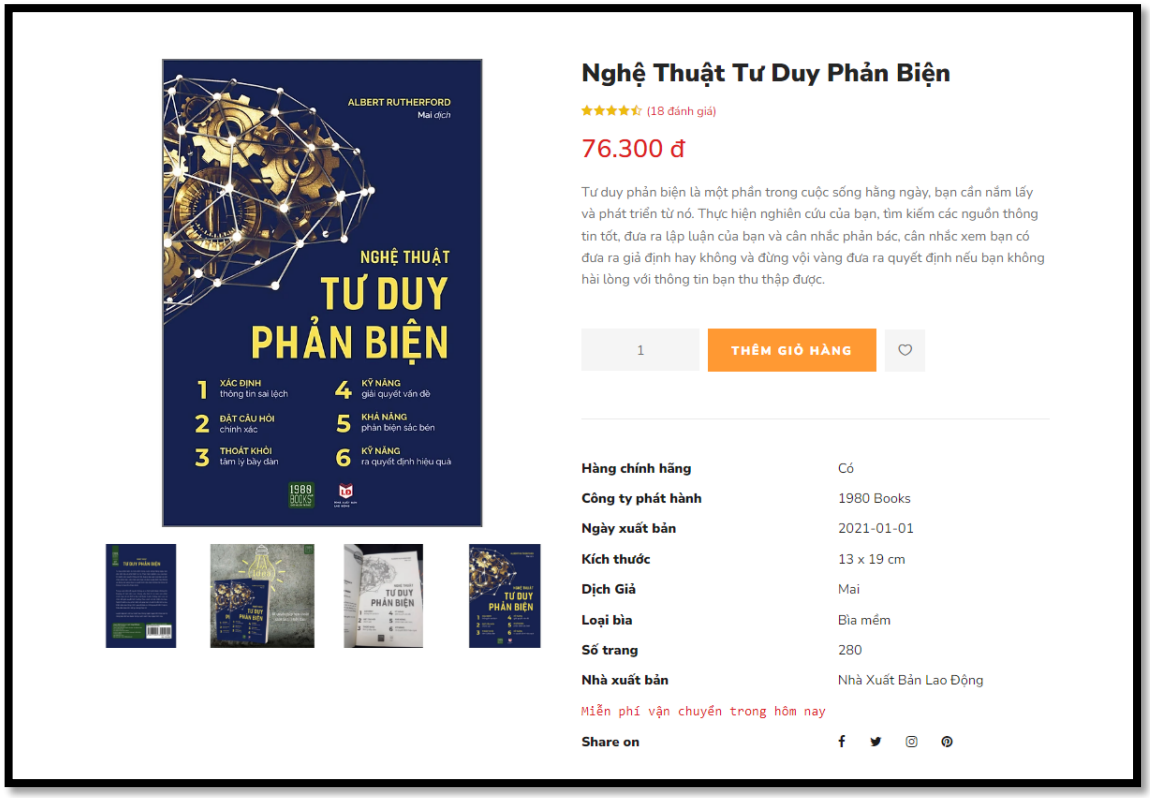
8. Đánh giá

Thang điểm tối đa: 10 điểm/Lab

9. Ví dụ

Chạy **Vi_du_1** và **Vi_du_2** để tìm hiểu về cách hoạt động của slider bằng JS:

Trong **Vi_du_1**, slider dùng để hiển thị các sản phẩm được chạy tự động, có thể sử dụng nút mũi tên trái hoặc phải để xem các sản phẩm trước và sau. Ngoài ra còn có thể dùng chuột kéo qua lại để trượt xem nhanh các sản phẩm rất tiện lợi.



Trong **Vi_du_2**, slider là các hình sản phẩm thu nhỏ (thumbnail) cũng được chạy tự động và có thể click vào để xem hình đầy đủ phía trên. Việc sử dụng slider rất tiện lợi, đem lại cho người dùng có cảm giác thao tác web gọn hơn.

10. Yêu cầu thực hành

Bài 1: (4đ) Sinh viên hãy thiết kế một kiểu slider có nút bấm trái phải. Hình ảnh được cung cấp trong thư mục **img** với yêu cầu sau:

- Có 5 bức hình ảnh, hiển thị 5 chấm tròn ở dưới slide để trỏ tới các ảnh tương ứng;
- Khi bấm nút chuyển slide chỉ chuyển 1 bức hình ảnh trước hoặc sau tương ứng;
- Lưu bài thành *lab08_slider.html*

Bài 2: (3đ) Từ bài *lab08_slider.html* trên hãy viết thêm một chức năng tự động chuyển slide.

- Gợi ý: hàm setInterval được thêm vào để tự động gọi hàm plusSlides(1) mỗi 2 giây, đồng nghĩa với việc chuyển slide tự động.
- Lưu bài thành *lab08_slider_auto.html*

Bài 3: (3đ) Sử dụng Javascript Json, yêu cầu tạo một ứng dụng web đơn giản để hiển thị thông tin từ một đối tượng JSON. Lưu bài thành *lab08_json.html*