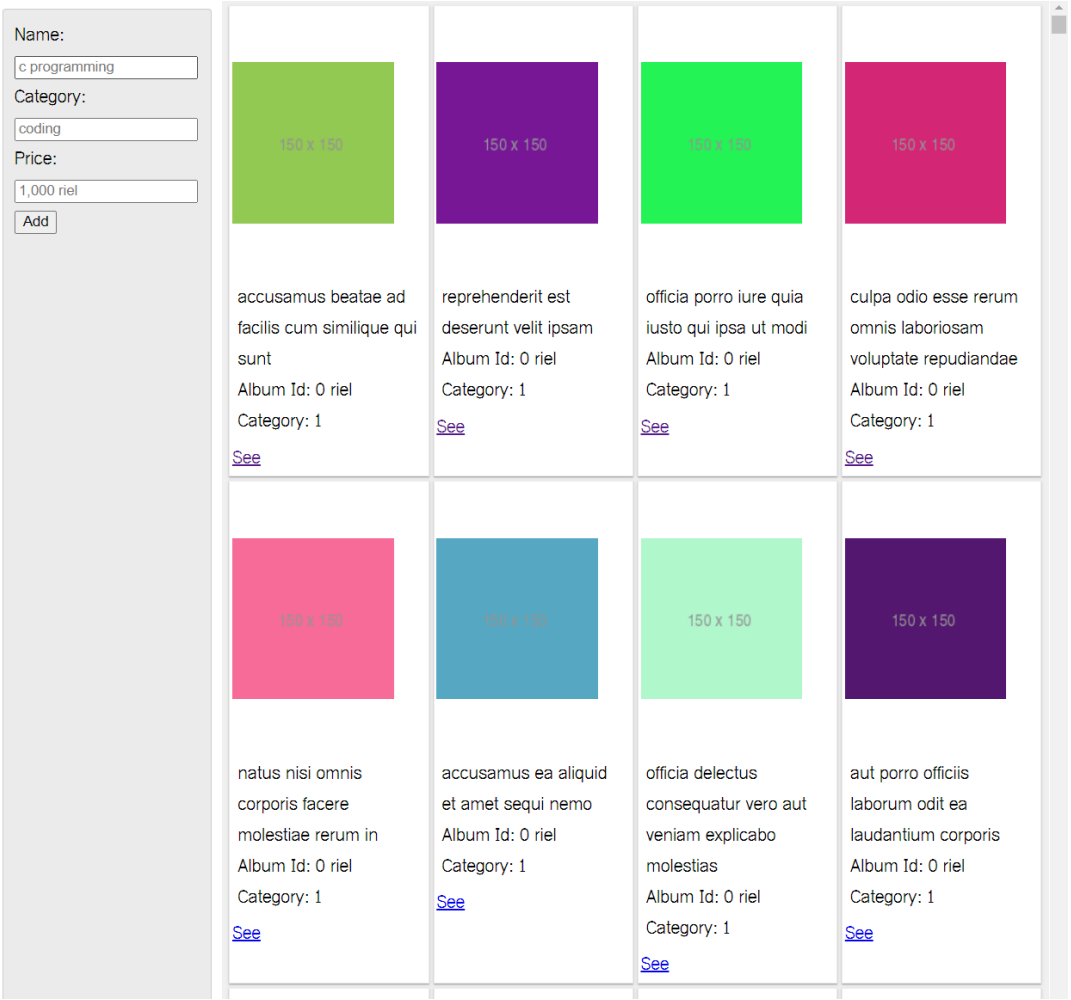
TP-07

JavaScript

(NodeJS, Typescript)

NodeJS

EX1: Create a simple NodeJS project to handle Bookstore app



- Run on NodeJS
- Access pages by following URL
  - http://localhost:3000/
  - http://localhost:3000/detail

Typescript

EX2: Write a Typescript library for Khmer DateTime. The library can be imported to use in JavaScript module.

```
const { KhmerDate } = require('./lib')


const date = new KhmerDate(new Date('2022-02-15T17:30:55.839Z'))

console.log(date.getDate());
```

1min <	មុននេះបន្ដិច
1hour <	...នាទីមុន
24hours <	...ម៉ោងមុន
7day <	...ថ្ងៃមុន
1week <	...សប្ដាហ៍មុន
1month <	...ខែមុន
1month >	

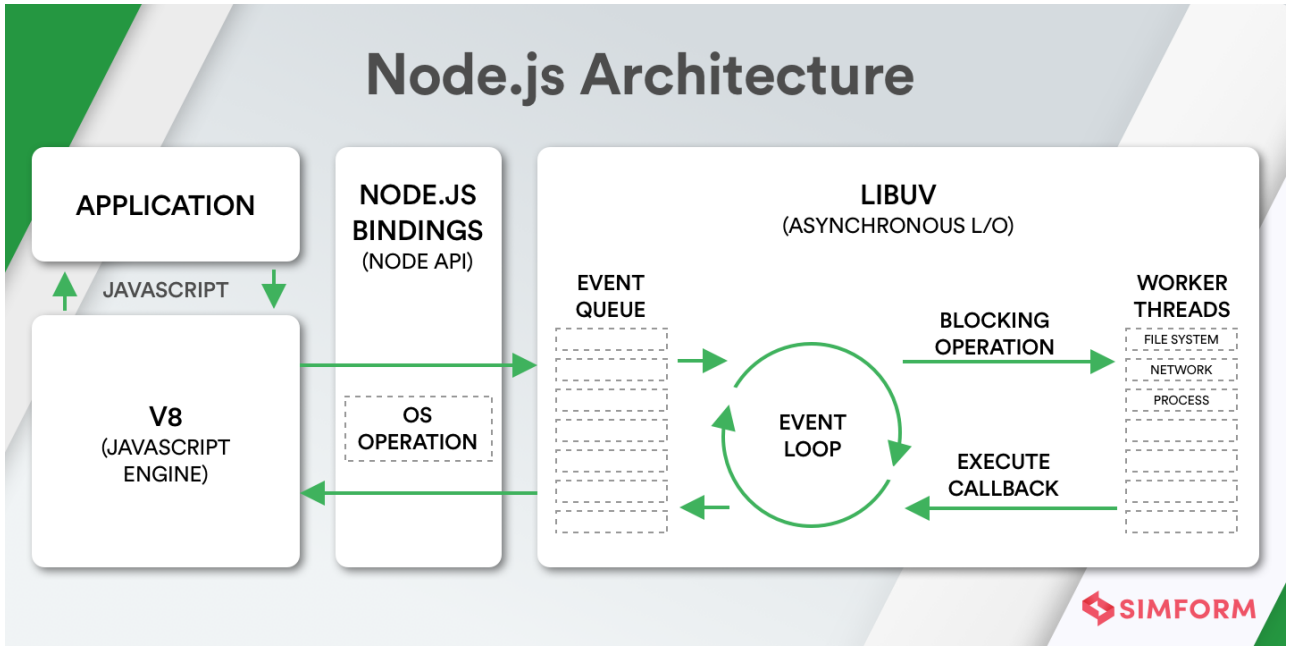
☞ The number must be converted to Khmer

Getting to understand

“NodeJS” & “TypeScript”

NodeJS

Node.js is an open-source, cross-platform, back-end JavaScript runtime environment that runs on the V8 engine and executes JavaScript code ***outside a web browser***.



- ***Asynchronous/Non-blocking thread execution*** – *Every API of the Node.js library is non-blocking. While waiting for a response for something outside the execution chain, the next tasks in the stack are continuously executed.*
- ***Event-driven*** – *A server built with Node.js uses a notification mechanism called "Events" to receive and track responses of previous API requests. Event Loop allows Node.js to execute all the non-blocking operations.*
- ***Cross-platform compatibility*** – *Node.js is compatible with various platforms like Windows, Linux, Mac OS X, Unix, and mobile platforms.*

https://www.simform.com/blog/what-is-node-js/

TypeScript

Basic:

```
const a: string = "foo";
const b = 1;
const c = false;
const d = [1, 2, 3];
const e = ["a", "b", "c"];
const f = { id: 1 };
const g = null;
const h = undefined;
```

```
const aTyped: string = 'foo'
const bTyped: number = 1
const cTyped: boolean = false
```

```
const dTyped: number[] = [1, 2, 3]
// or
const dTyped : Array<number> = [1, 2, 3]
```

```
const eTyped: Array<string> = ["a", "b", "c"];
const fTyped: Object = { id: 1 };
// or better
const fTyped: { id: number } = { id: 1 };
const gTyped: null = null
```

```
type ExpectedInput = 1 | 2 | 3

const doSomething = (input: ExpectedInput) => {
  switch (input) {
    case 1:
      return 'Level 1'
    case 2:
      return 'Level 2'
    case 3:
      return 'Level 3'
  }
}

doSomething(0) // error: This type is incompatible with the expected par
doSomething(1) // ok
```

```
let aVar: string = "foo";

aVar = 'bar'
aVar = 1  // Error!
```

TypeScript

Any vs Unknown

```
const double = (input: unknown) => {
  if (typeof input === 'string') {
    return input + ' - ' + input
  }
  if (Array.isArray(input)) {
    return input.concat(input)
  }
  return input
}

const result = double('foo') // ok
```

```
const length = (input: any) => {
  if (typeof input === "string") {
    return input.length;
  }

  if (Array.isArray(input)) {
    return input.length;
  }

  return 0;
};

length("foo");
length([1, 2, 3, 4]);
length(1); // no Error!
```

TypeScript

Optional Values

```
const optionalLength = (input?: string | Array<any>) => {
  if (typeof input === "string") {
    return input.length;
  }

  if (Array.isArray(input)) {
    return input.length;
  }

  return false;
};

optionalLength();
optionalLength(undefined);
optionalLength([1, 2, 3, 4]);
optionalLength("foo");
```

```
optionalLength(1) // Error!
```

```
optionalLength(null); // error! We need to be explicit about null
```

TypeScript

Functions

```
let add = (a: number, b: number): number => {
  return a + a;
};

add(2, 2);
add(2, "a"); // Error!
const addResult: number = add(2, 2);
```

```
const addResultError : string = add(1, 2); // Error!
```

Array

```
const aArray : Array<number> = [1, 2, 3]
const aArrayShortHand : number[] = [1, 2, 3]
```

```
const aOptionalArray: Array<number | null | undefined> = [
  1,
  null,
  2,
  undefined
];
const aOptionalArrayShortHand: (number | null | undefined)[] = [
  1,
  null,
  2,
  undefined
];
```

```
const bArray: Array<number> = [1, 2, 3];
bArray.push(4);
bArray.push("foo"); // Error!
```

TypeScript

Objects

```
const aObject: Object = { id: 1, name: "foo" };
const bObject: { id: number } = { id: 1, name: "foo" }; // !Error
```

- Type

```
type E = { id: number; name: string; points?: number };
const eObject: E = { id: 1, name: "foo" };
```

```
type F = {id: number, name: string}
const fObject : F = {id: 1, name: 'foo', points: 100} // Error!
```

```
const aMap: { [key: number]: string } = {};
aMap[1] = "foo";
aMap["a"] = "foo"; // Error!
aMap[1] = 1; // Error!

const otherMap: { [key: string]: number } = {};
otherMap["foo"] = 1;
otherMap[1] = 2; // No Error!
otherMap["bar"] = "foo"; // Error!
```

TypeScript

Class

```
class Foo {
  state = { val: 0 };
  update(val: number) {
    this.state = { val };
  }
  getVal() {
    return this.state.val;
  }
}

const foobar: Foo = new Foo();
```

```
class Foo {
  state: { val: number } = { val: 0 };
  update(val: number): void {
    this.state = { val };
  }
  getVal(): number {
    return this.state.val;
  }
}

const foobar: Foo = new Foo();

foobar.update(3);
foobar.update("foo"); // Error!

const fooResult: number = foobar.getVal();
const fooResultError: string = foobar.getVal(); // Error!
```

TypeScript

Interfaces

```
interface Updateable<T> {
  state: { val: T };
  update(a: T): void;
}

class InterfaceExample implements Updateable<boolean> {
  state = { val: false };
  constructor(val: boolean) {
    this.state = { val };
  }
  update(val: boolean) {
    this.state = { val };
  }
  getValue() {
    return this.state.val;
  }
}

const exampleInstance = new InterfaceExample(true);
const exampleInstanceResultOk: boolean = exampleInstance.getValue();
const exampleInstanceResultError: number = exampleInstance.getValue(); /
```

I want more about TS

👉 https://www.typescriptlang.org/docs/handbook/

Good luck 👌