Què és l'arquitectura MVC?

MVC és una arquitectura amb 3 capes/parts:

- **Models –** Gestiona les dades duna aplicació. Els models seran anèmics (mancaran de funcionalitats) ja que es derivaran als serveis.
- **Views –** Una representació visual dels models.
- **Controller –** Enllaços entre els serveis i les vistes.

A continuació explicarem l'aplicació `Ejer3 – TODO`. Aquesta conté un fitxer `index.html` que actuarà de vista. A més, té els següents fitxers JavaScript:

- `todo.model.js` – Els atributs (el model) duna tasca.
- `todo.controller.js` – L'encarregat d'unir al servei i la vista.
- `todo.service.js` – Gestiona totes les operacions sobre els TODOs.
- `todo.views.js` – Encarregat de refrescar i canviar la pantalla de visualització.

Tota l'aplicació es renderitzarà en un simple node de html anomenat `root` que serà manejat a través de JavaScript.

```
<!DOCTYPE html>
<html lang="en">
 <head>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <meta http-equiv="X-UA-Compatible" content="ie=edge"/>
  <title>Todo App</title>
  <link rel="stylesheet" href="style.css" />
 </head>
 <body>
  <div id="root"></div>
  <script src="models/todo.model.js"></script>
  <script src="services/todo.service.js"></script>
  <script src="controllers/todo.controller.js"></script>
  <script src="views/todo.views.js"></script>
  <script src="app.js"></script>
 </body>
</html>
```

Fixa't que s'han enllaçat els diferents fitxers JavaScript que es requereixen: `todo.model`, `todo.service`, `todo.controller`, `todo.views` y el llançador `app.js`. Aquesta tasca serà automatitzada gràcies als frameworks.

A més, s'ha escrit un petit CSS anomenat `style.css` per donar un aspecte visual acceptable a l'aplicació que es desenvoluparà.

Model (`todo.model.js`)

La primera classe que es construeix en aquest exemple és el model de l'aplicació, `todo.model.js`, la qual consisteix en els atributs de la classe, i un mètode privat que genera ID aleatòries (aquestes id's podrien venir del servidor des d'una base de dades en un punt més avançat de laplicació).

```
/**
 * @class Model
 *
 * Manages the data of the application.
 */
class Todo {
  constructor({ text, complete } = { complete: false }) {
    this.id = this.uuidv4();
    this.text = text;
    this.complete = complete;
  }

  uuidv4() {
    return ([1e7] + -1e3 + -4e3 + -8e3 + -1e11).replace(/[018]/g, c =>
      (
        c ^
        (crypto.getRandomValues(new Uint8Array(1))[0] & (15 >> (c / 4)))
      ).toString(16)
    );
  }
}
```

Servici (`todo.service.js`)

Les operacions que es realitzen sobre els TODO són dutes a terme en el servei. El servei és el que permet que els models siguin anèmics, ja que tota la càrrega de lògica s'hi troba. En aquest cas concret utilitzarem un array per emmagatzemar tots els TODO i construirem els 4 mètodes associats a llegir, modificar, crear i eliminar TODO. Heu d'observar que el servei fa ús del model, instant els objectes que s'extreuen de `LocalStorage` a la classe `Todo`. Això és així perquè `LocalStorage` només emmagatzema dades i no prototips de

les dades emmagatzemades. El mateix passa amb les dades que viatgen del backend al frontend, aquestes no tenen instanciades les classes.

```
class TodoService {
 constructor() {
  this.todos = (JSON.parse(localStorage.getItem("todos")) || []).map(
   todo => new Todo(todo)
  );
 }
 bindTodoListChanged(callback) {
  this.onTodoListChanged = callback;
 }
 _commit(todos) {
  this.onTodoListChanged(todos);
  localStorage.setItem("todos", JSON.stringify(todos));
 }
 addTodo(text) {
  this.todos.push(new Todo({ text }));

  this._commit(this.todos);
 }
 editTodo(id, updatedText) {
  this.todos = this.todos.map(todo =>
   todo.id === id
    ? new Todo({
       ...todo,
       text: updatedText
      })
    : todo
  );
  this._commit(this.todos);
 }
 deleteTodo(_id) {
  this.todos = this.todos.filter(({ id }) => id !== _id);

  this._commit(this.todos);
 }

 toggleTodo(_id) {
  this.todos = this.todos.map(todo =>
   todo.id === _id ? new Todo({ ...todo, complete: !todo.complete }) : todo
```

```
    );

    this._commit(this.todos);
  }
}
```

Vista (`todo.views.js`)

La classe corresponent a la vista és l'encarregada de gestionar el DOM de l'aplicació. Com es va veure a l'assignatura de programació de JavaScript, el DOM (*Document Object Model*) és una API des de la qual es pot modificar completament l'aspecte visual d'una pàgina web, que permet crear, eliminar o refrescar nodes d'HTML. Encara que aquesta tasca és molt tediosa, és fonamental el maneig del DOM per part dels desenvolupadors web. Una de les principals característiques de tots els frameworks és que oculten el DOM i en faciliten el maneig per als desenvolupadors.

Si observes el fitxer relatiu a la vista, `todo.views.js`, s'han seleccionat elements de l'arbre del DOM (pàgina web) per assignar esdeveniments, les accions de clic al botó esborrar, sobre el text, o el checkbox. Si observeu el codi (p.ex. línia 102), podreu veure que les accions o *handles* són funcions passades com a paràmetres a la vista. D'aquesta manera, s'ha aconseguit abstreure completament la vista de les accions que cal fer. Les funcions de la vista són anomenades des del controlador.

```
/**
 * @class View
 *
 * Visual representation of the model.
 */
class TodoView {
  constructor() {
    this.app = this.getElement("#root");
    this.form = this.createElement("form");
    this.input = this.createElement("input");
    this.input.type = "text";
    this.input.placeholder = "Add todo";
    this.input.name = "todo";
    this.submitButton = this.createElement("button");
    this.submitButton.textContent = "Submit";
    this.form.append(this.input, this.submitButton);
    this.title = this.createElement("h1");
```

```
    this.title.textContent = "Todos";
    this.todoList = this.createElement("ul", "todo-list");
    this.app.append(this.title, this.form, this.todoList);

    this._temporaryTodoText = "";
    this._initLocalListeners();
  }

  get _todoText() {
    return this.input.value;
  }

  _resetInput() {
    this.input.value = "";
  }

  createElement(tag, className) { … }

  getElement(selector) { ... }

  displayTodos(todos) { … }
  _initLocalListeners() { … }

  bindAddTodo(handler) { … }

  bindDeleteTodo(handler) { … }

  bindEditTodo(handler) { … }

  bindToggleTodo(handler) { … }
}
```

Controlador (`todo.controller.js`)

Finalment, la classe que uneix totes les parts és el controlador, on es reben les instàncies de servei i vista per crear els manejadors (és a dir, les funcions encarregades de realitzar una acció quan es prem un botó o realitzar una acció concreta a la vista).

Al controlador s'han unit (binding) les accions de la vista amb les del servei.

```
/**
 * @class Controller
 *
 * Links the user input and the view output.
 *
 * @param model
 * @param view
 */
class TodoController {
  constructor(service, view) {
    this.service = service;
    this.view = view;

    // Explicit this binding
    this.service.bindTodoListChanged(this.onTodoListChanged);
    this.view.bindAddTodo(this.handleAddTodo);
    this.view.bindEditTodo(this.handleEditTodo);
    this.view.bindDeleteTodo(this.handleDeleteTodo);
    this.view.bindToggleTodo(this.handleToggleTodo);

    // Display initial todos
    this.onTodoListChanged(this.service.todos);
  }
  onTodoListChanged = todos => {
    this.view.displayTodos(todos);
  };

  handleAddTodo = todoText => {
    this.service.addTodo(todoText);
  };

  handleEditTodo = (id, todoText) => {
    this.service.editTodo(id, todoText);
  };

  handleDeleteTodo = id => {
    this.service.deleteTodo(id);
  };

  handleToggleTodo = id => {
    this.service.toggleTodo(id);
```

```
  };
}
```

Llançadora (`app.js`)

L'últim pas que queda per fer funcionar l'aplicació és "llançar-la". Per això tenim el fitxer `app.js` que actua de llançador de tota la lògica a través de la instanciació de `Controller`.

```
const app = new TodoController(new TodoService(), new TodoView());
```

Si necessiteu més detall sobre el funcionament de l'exemple, disposeu de més explicacions de com està realitzat a l'article que va publicar l'autora: https://www.taniarascia.com/javascript-mvc-todo-app/

També aquest enllaç us pot resultar d'interès per aprofundir en MVC, inclou un altre exemple pràctic.

[Nat Apuntes] Arquitectura MVC usant VanillaJS