Arquitectura MVC amb TypeScript

```
<!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="dist/bundle.js"></script>
 </body>
</html>
```

Model (`todo.model.ts`)

La primera classe que es construeix en aquest exemple és el model de laplicació, `todo.model.ts`, 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 l'aplicació). Els atributs són del tipus següent:

```
- id: string;
- text: string;
- complete: boolean;
```

Fixa't que has de definir també el valor per defecte dels atributs al constructor. A continuació, es mostra la classe a .js prèvia a transformar-la a TypeScript

```
/**
 * @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)
    );
  }
}
```

Servei (`todo.service.ts`)

Les operacions que es realitzen sobre els TODOs són dutes a terme al servei. En aquest cas concret utilitzarem un *array* per emmagatzemar tots els TODOs (és a dir, l'*array* `todos` haurà de contenir objectes de la classe `Todo`) i construirem els 4 mètodes associats a llegir, modificar, crear i eliminar TODOs.

A continuació es mostra la classe `TodoService` que hauràs de transformar a TypeScript (Tingues en compte que els callback són paràmetres del tipus **Function**).

```
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.ts`)

La classe corresponent a la vista és lencarregada de gestionar el DOM de laplicació.

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.

Per convertir aquest codi en TypeScript heu de tenir en compte que els elements html tenen una interfície específica HTMLElement

(https://developer.mozilla.org/es/docs/Web/API/HTMLElement).

El fitxer .js que cal transformar a TypeScript és el següent.

```
/**
 * @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.ts`)

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 fer una acció concreta a la vista).

Al controlador s'han unit (*binding*) les accions de la vista amb les del servei. El fitxer `controller.js` que ha de ser transformat a TypeScript és el següent:

```
/**
 * @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.ts`)

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

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