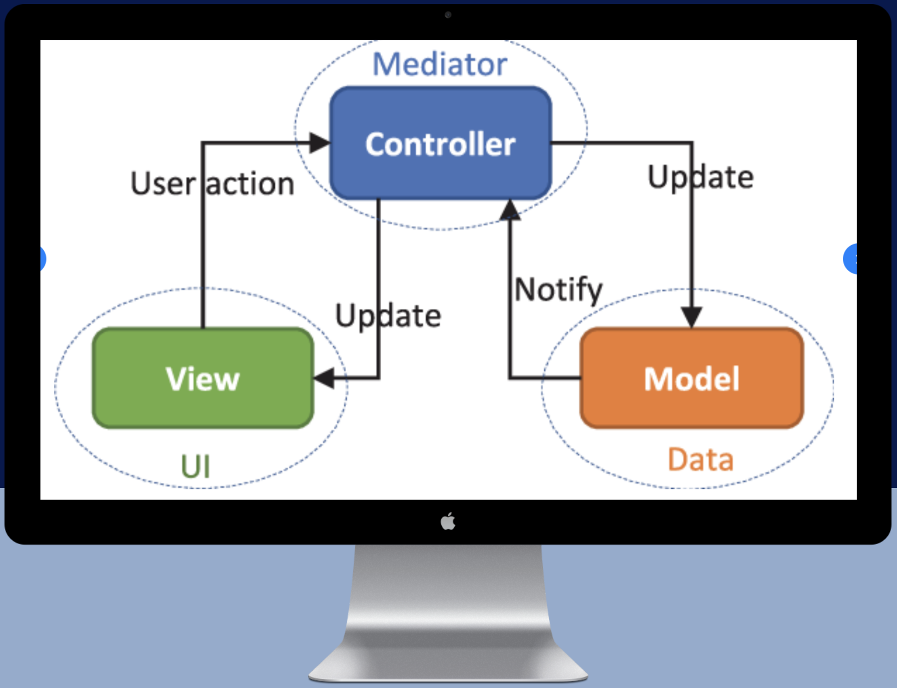


Model View Controller (MVC)

Alberto Cruz Luis

alu0101217734@ull.edu.es

Jeremy Manuel Luis León

alu0101244587@ull.edu.es

Content Table

Content Table

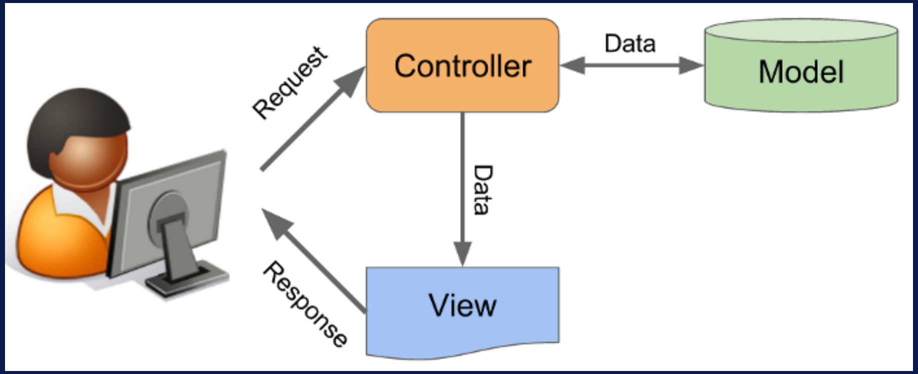
What is Model View Controller?

It is a software architecture pattern, which separates the data and mainly the business logic of an application from its representation and the module in charge of managing events and communications.

The **model** represents the data, and does nothing else.

The **view** displays the model data, and sends user actions to the controller.

The **controller** provides model data to the view, and interprets user actions such as button clicks.

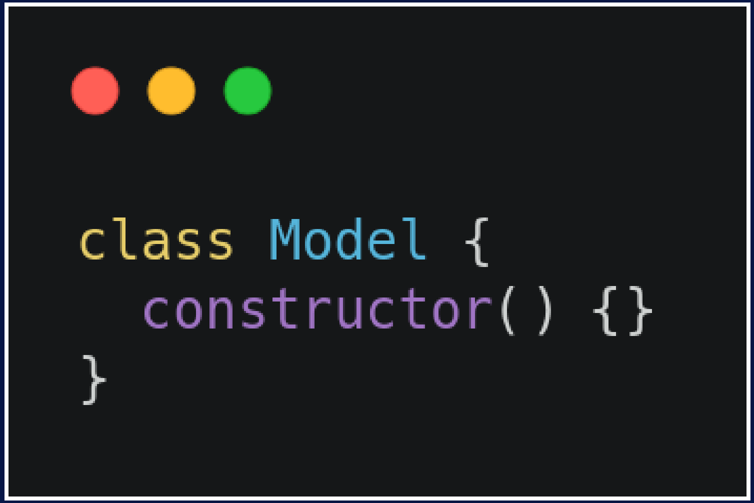


History

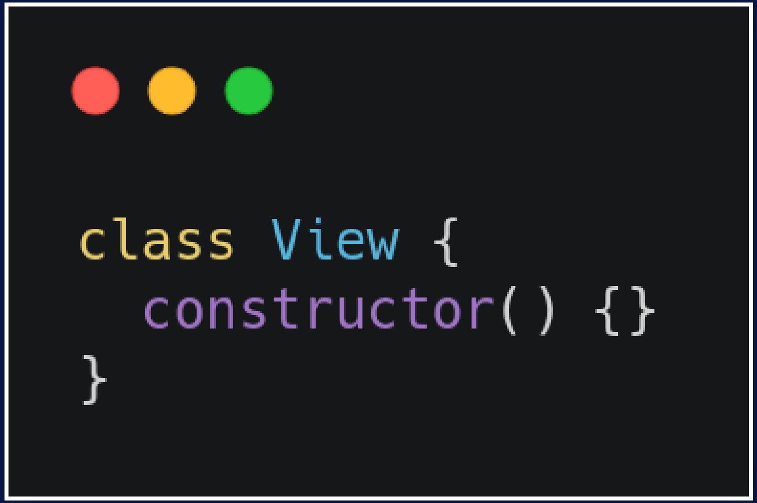
1. Introduced in 1979 by Trygve Reenskaug into Smalltalk-79

2. The MVC pattern has subsequently variants such as hierarchical model–view–controller (HMVC), model–view–adapter (MVA), model–view–presenter (MVP), model–view–viewmodel (MVVM)

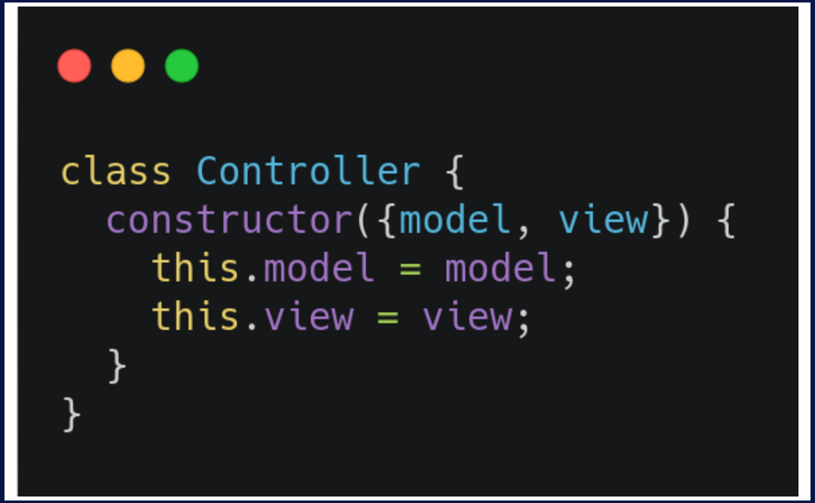
Skeleton for MVC



```
class Model {
  constructor() {}
}
```

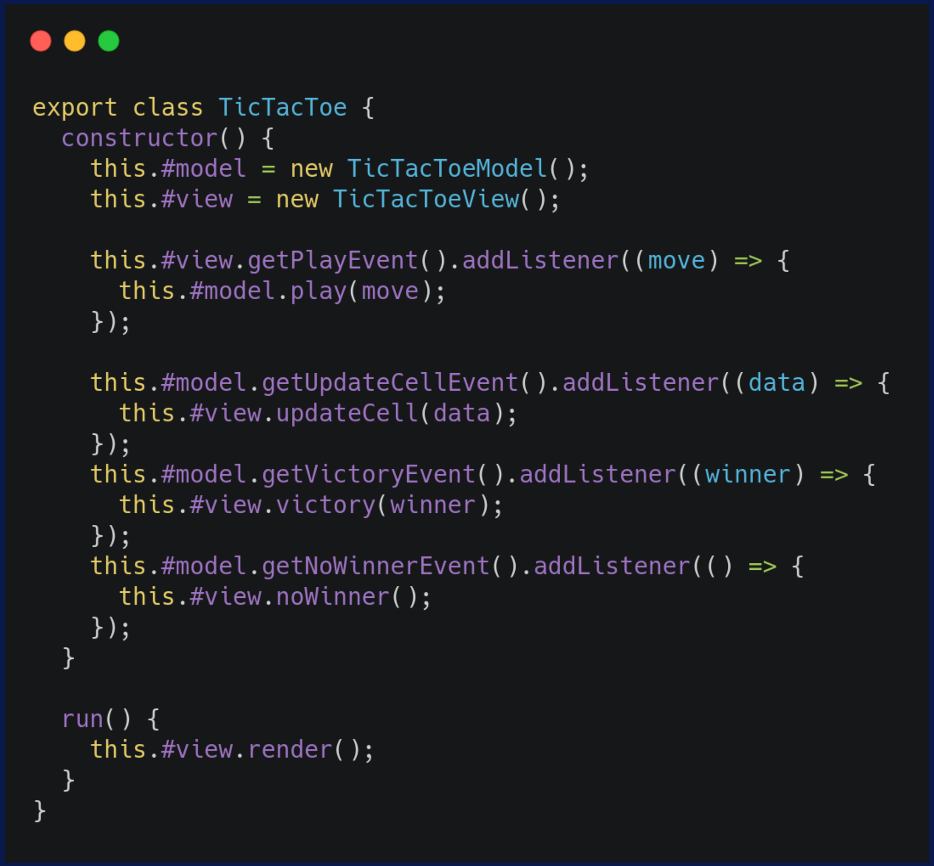


```
class View {
  constructor() {}
}
```

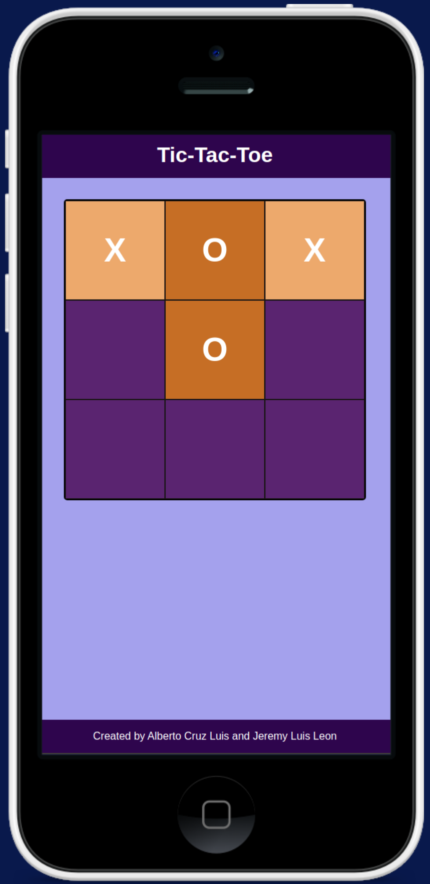


```
class Controller {
  constructor({model, view}) {
    this.model = model;
    this.view = view;
  }
}
```

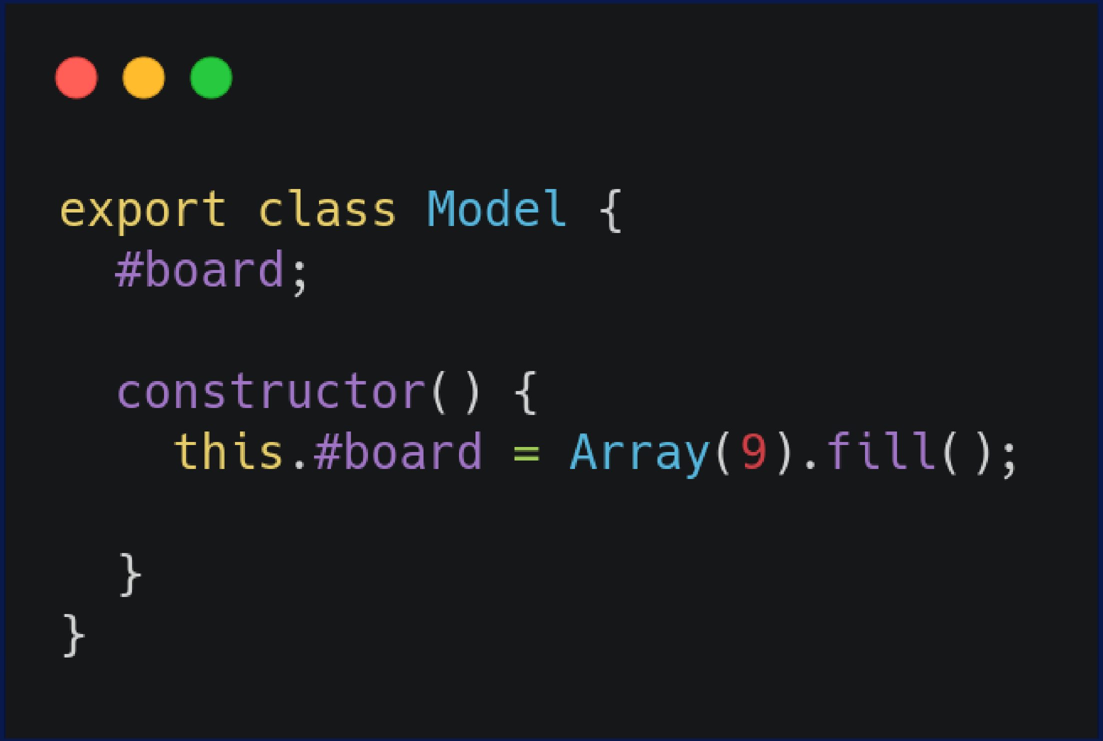
Tic-Tac-Toe in MVC



```
export class TicTacToe {
  constructor() {
    this.#model = new TicTacToeModel();
    this.#view = new TicTacToeView();

    this.#view.getPlayEvent().addListener((move) => {
      this.#model.play(move);
    });

    this.#model.getUpdateCellEvent().addListener((data) => {
      this.#view.updateCell(data);
    });
    this.#model.getVictoryEvent().addListener((winner) => {
      this.#view.victory(winner);
    });
    this.#model.getNoWinnerEvent().addListener(() => {
      this.#view.noWinner();
    });
  }

  run() {
    this.#view.render();
  }
}
```



Model

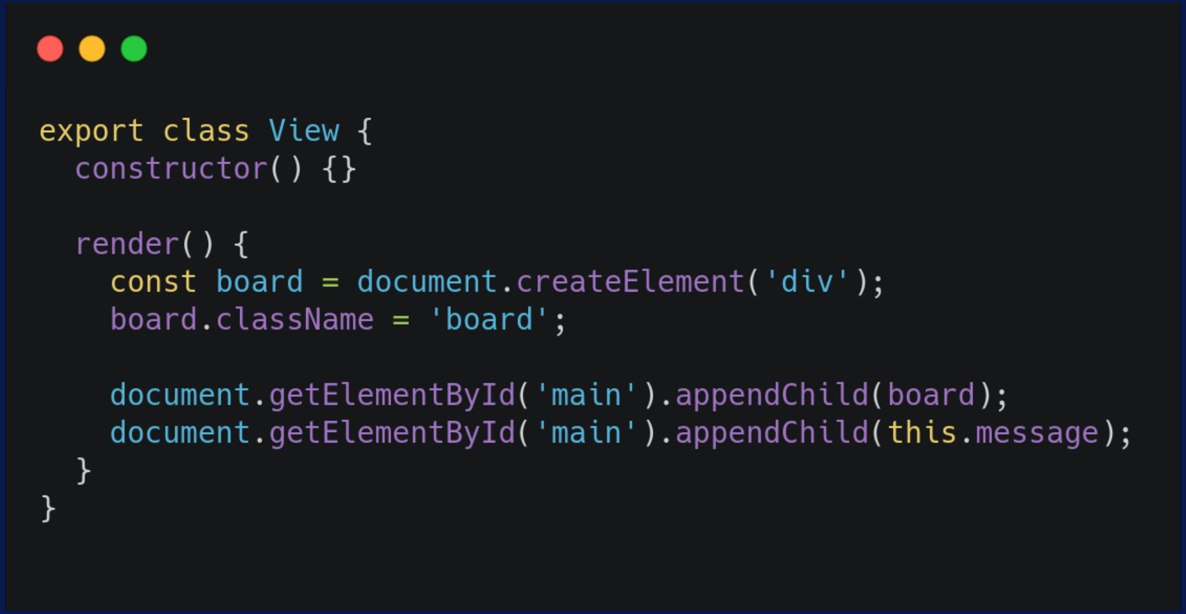


```
export class Model {
  #board;

  constructor() {
    this.#board = Array(9).fill();

  }
}
```

- Is responsible for managing the data of the application
- It responds to request from the view and it also responds to instructions from the controller to update itself
- It is the lowest level of the pattern which is responsible for maintaining data
- The model represents the application core.
- It is also called the domain layer

View

The View, or user interface, is made up of the information that is sent to the client and the interaction mechanisms with it.



```
export class View {
  constructor() {}

  render() {
    const board = document.createElement('div');
    board.className = 'board';

    document.getElementById('main').appendChild(board);
    document.getElementById('main').appendChild(this.message);
  }
}
```

- Receive data from the model and display it to the user.
- It is solely responsible for the graphical interface and how the end user will see the data

Controller

```
export class Controller {
  #model;
  #view;

  constructor() {
    this.#model = new Model();
    this.#view = new View();
  }

  run() {
    this.#view.render();
  }
}
```

- It is the link between the view and the model
- Is responsible for receiving and responding to events

Advantages & Disadvantages

Advantages	Disadvantages
Clear separation between presentation logic and business logic	Hard to understand the architecture
Single responsibility principle	Must have strict rules on methods
Modifications does not affect the entire model	Increased complexity
Fast development	MVC design pattern requires greater dedication in the early stages of development.
Modular implementation	
The developer does not have to worry about requesting that the views be updated	

Alternatives for MVC

PAC
Presentation Abstraction Control

MVA
Model View Adapter

MVP
Model View Presenter

MVVM
Model-View-ViewModel

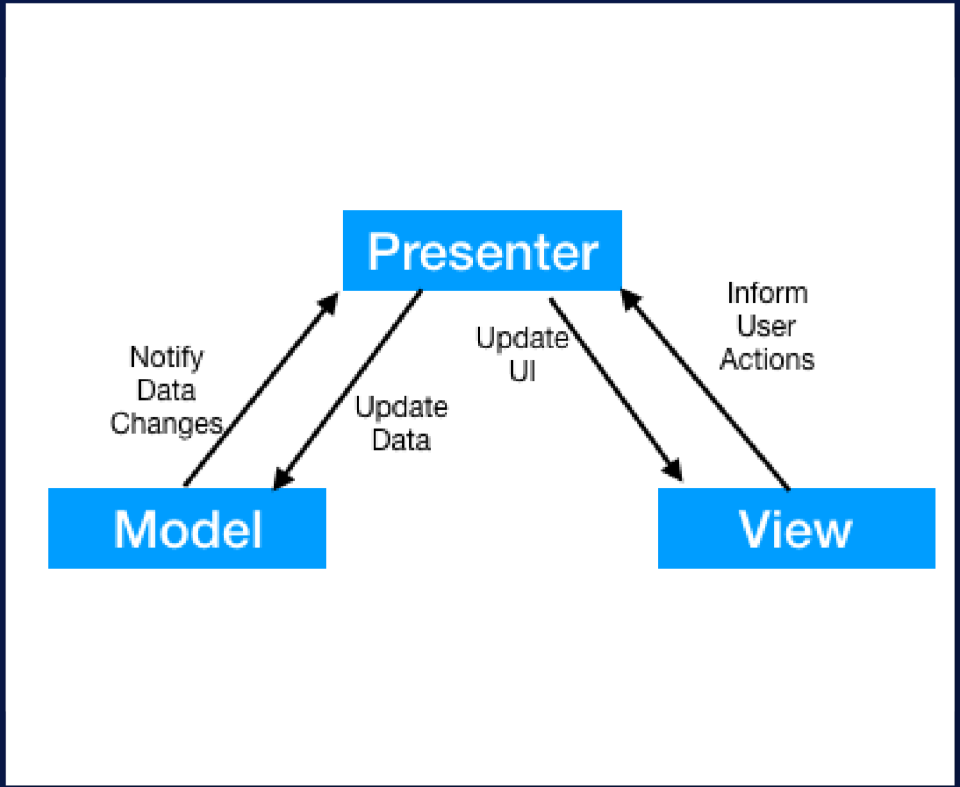
HMVC
Hierarchical Model-View-Controller

Redux

Flux

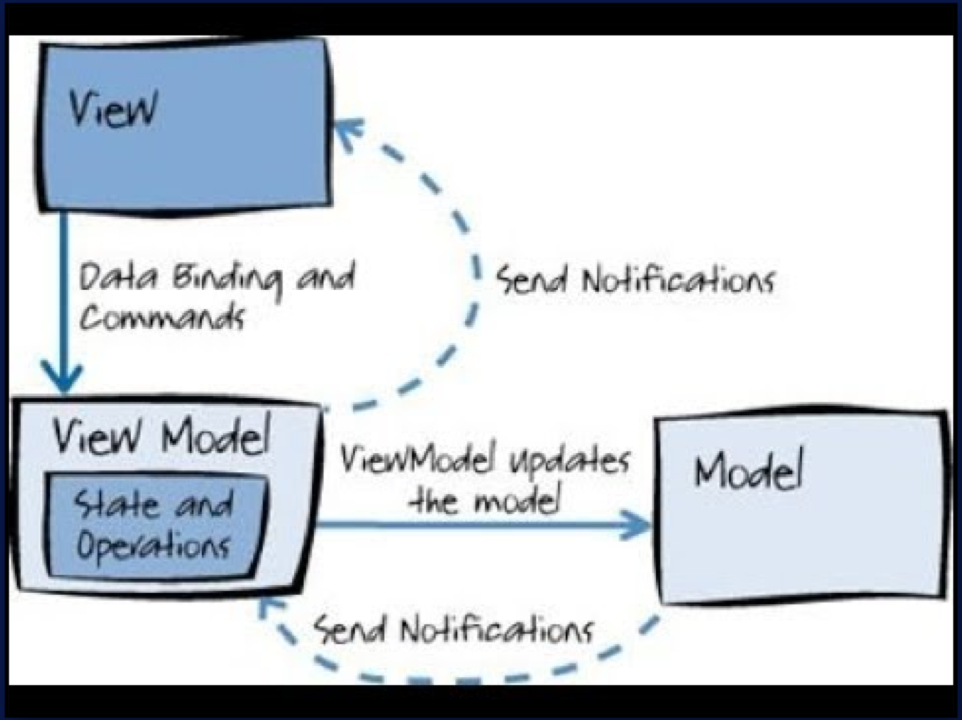
MVP

- **Model:** provides the data that the application requires, and we want to display in the *view*.
- **View:** to display the data from the model, it passes the user actions/commands to the presenter to act upon that data.
- **Presenter:** acts as the middle man between the *model* and the *view*. Retrieves data from the model manipulates it, and returns it to view for display.



MVVM

- **Model:** Model refers either to a domain model, which represents content.
- **View:** As in the model–view–controller (MVC) and model–view–presenter (MVP) patterns, the view is the structure, layout, and appearance of what a user sees on the screen.
- **ViewModel:** Instead of the controller of the MVC pattern, or the presenter of the MVP pattern, MVVM has a binder, which automates communication between the view and its bound properties in the view model.

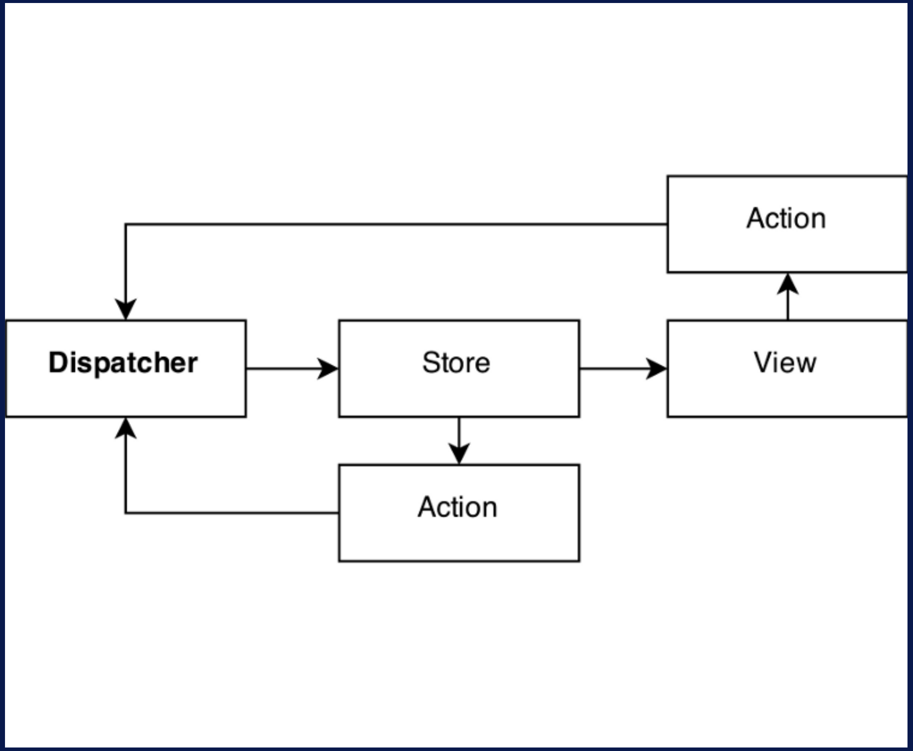


What is Flux?

Just as an MVC pattern is made up of a Model, a View and a Controller, in Flux we have different actors:

- **View:** View would be the web components, whether they are built natively, with Polymer, with Angular, React, ...
- **Store:** Store would be the closest thing to the application model. Save the application data / status and in Flux there may be several.
- **Actions:** An action is simply a JavaScript object that indicates an intention to do something and that carries associated data if necessary.

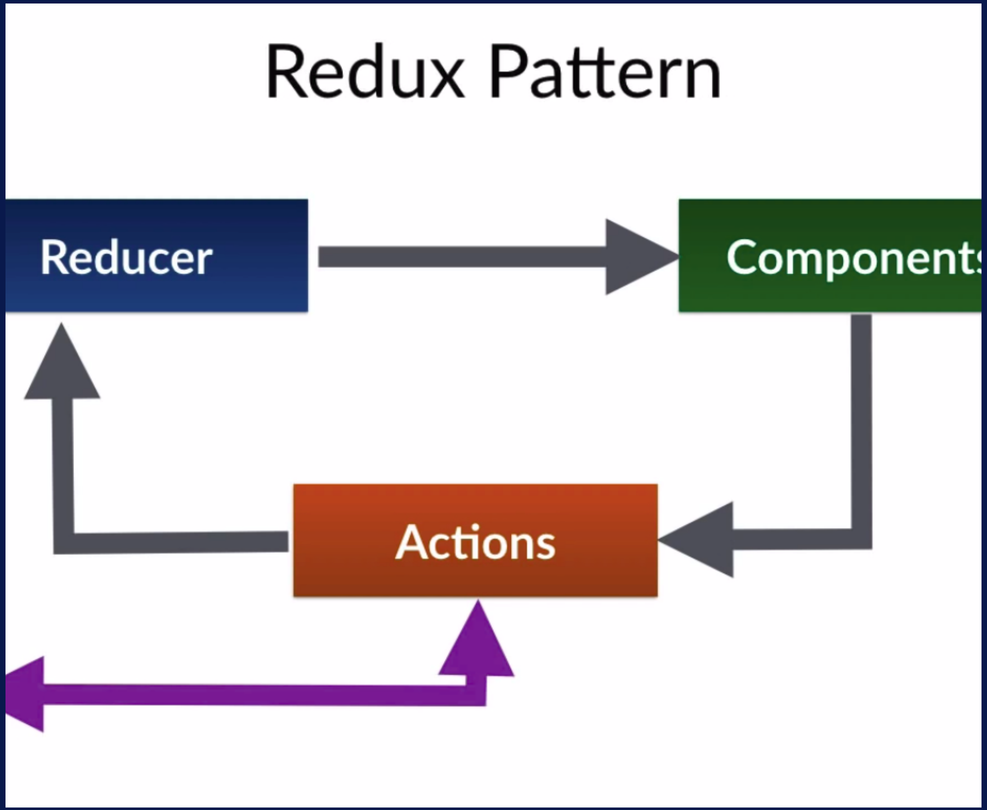
Dispatcher: Actions are sent to a dispatcher that is responsible for triggering it or propagating it to the Store.



What is Redux?

Redux is a data architecture pattern that allows you to handle the state of the application in a predictable way.

Originally started by the React community, as an evolution and improvement of the ideas of Flux, Redux has become a transversal pattern, capable of adapting to any type of library or client-side framework.



Differences between MVC, Flux and Redux

MVC

1. Architectural design pattern for developing UI
2. Follows the bidirectional flow
3. No concept of store
4. Shines well in both client and server-side frameworks

Flux

1. Application architecture designed to build client-side web apps.
2. Follows the unidirectional flow
3. Includes multiple stores
4. Supports client-side framework

Redux

1. Open-source JavaScript library used for creating the UI
2. Follows the unidirectional flow
3. Includes single store
4. Supports client-side framework

Bibliography

Model View Controller Theory

- https://www.taniarascia.com/javascript-mvc-todo-app/
- https://www.taniarascia.com/javascript-mvc-todo-app/
- https://programmerclick.com/article/388054092/

Model View Controller Code Examples

- https://solucionesci.wordpress.com/2016/07/12/patrones-de-diseno-web-alternativas-a-mvc/
- https://www.oreilly.com/library/view/learning-javascript-design/9781449334840/ch10s02.html
- https://www.tutorialspoint.com/design_pattern/mvc_pattern.htm

Alernatives for Model View Controller

- https://carlosazaustre.es/como-funciona-redux-conceptos-basicos
  https://desarrolloweb.com/articulos/que-es-redux.html
  http://blog.enriqueoriol.com/2018/08/que-es-redux.html
- https://www.clariontech.com/blog/mvc-vs-flux-vs-redux-the-real-differences
- https://proandroiddev.com/mvc-mvp-mvvm-clean-viper-redux-mvi-prnsaaspfruicc-building-abstractions-for-the-sake-of-building-18459ab89386

History

- https://sites.google.com/site/aunaris2/programacion/modelo-vista---controlador
- https://blog.nubecolectiva.com/que-es-mvc-modelo-vista-controlador-y-otros-detalles/

Tic-Tac-Toe Based Code

- https://github.com/elshaka/mvc-tictactoe
  https://hackernoon.com/writing-a-simple-mvc-model-view-controller-app-in-vanilla-javascript-u65i34lx

Thanks for your Attention

Alberto Cruz Luis
alu0101217734@ull.edu.es

Jeremy Manuel Luis Leon
alu0101244587@ull.edu.es