Framework agnostic client side state management

Antonello Pasella - www.pasella.it

Flux

Flux

RxJS

Flux Observables

RxJS

Flux Observables

RxJS Immutables

Redux Flux Observables

RxJS Immutables

Redux Flux Observables

RxJS AntaniJS Immutables

Come siamo arrivati fin qui?

Redux Flux Observables

RxJS AntaniJS Immutables

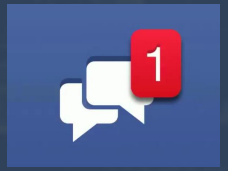
Cronologia

tutto facile

Cronologia

Gennaio 2014

Facebook scopre un bug nella sua chat



tutto facile

Cronologia

Gennaio 2014

Facebook scopre un bug nella sua chat

tutto facile

Maggio 2014

Facebook presenta Flux

Cronologia

Gennaio 2014

Facebook scopre un bug nella sua chat

tutto facile

casino totale

Maggio 2014

Facebook presenta Flux

Cronologia

Gennaio 2014

Facebook scopre un bug nella sua chat

tutto facile | casino totale

Maggio 2014

Facebook presenta Flux

Coincidenza? Non credo

Fatti

La UI mostra sempre più informazioni da più sorgenti

Websocket, notifiche push etc

Fatti

La UI mostra sempre più informazioni da più sorgenti

Websocket, notifiche push etc

Le applicazioni di uso comune abituano gli utenti a una elevata complessità

Modalità offline, sincronizzazione multi device

Fatti

La UI mostra sempre più informazioni da più sorgenti

Websocket, notifiche push etc

Le applicazioni di uso comune abituano gli utenti a una elevata complessità

Modalità offline, sincronizzazione multi device

I clienti vogliono per i loro utenti la stessa user experience ma con budget da pezzenti

“Tanto è tutto automatico, no?” (Er Piotta)

Problemi

Lo stato applicativo
e la sua interazione
con la UI sono
diventati sempre
più complessi

Problemi

Lo stato applicativo e la sua interazione con la UI sono diventati sempre più complessi

La manipolazione dello stato deve essere deterministica e chiara

Problemi

Lo stato applicativo e la sua interazione con la UI sono diventati sempre più complessi

La manipolazione dello stato deve essere deterministica e chiara

“Occorre fornire più dati grezzi al client”

(Jing Chen, Facebook)

Application state

Tipi:

- server
- persistent
- client
- transient
- local UI

Fonte: https://blog.nrwl.io/managing-state-in-angular-applications-22b75ef5625f

Application state

Tipi:

- server
- persistent
- client
- transient
- local UI

Sincronizzazione:

- server
- client
- url
- storage
- memory

Fonte: https://blog.nrwl.io/managing-state-in-angular-applications-22b75ef5625f

Server state

“E’ memorizzato sul server”

(Capitan Ovvio)

Server state

“E’ memorizzato sul server”

(Capitan Ovvio)

Normalmente il puntamento corrente viene gestito nella URL

Server state

“E’ memorizzato sul server”

(Capitan Ovvio)

Normalmente il puntamento corrente viene gestito nella URL

Il client non deve necessariamente avere accesso a tutti i dati

Persistent state

E’ la porzione di server state a cui ha accesso il client

Persistent state

E’ la porzione di server state a cui ha accesso il client

Concettualmente è sincronizzato con il server state

Persistent state

E’ la porzione di server state a cui ha accesso il client

Concettualmente è sincronizzato con il server state

Potrebbe differire dal server state in caso di gestione offline o aggiornamenti ottimistici

Client state

“Non è memorizzato sul server”

(Capitan Ovvio, ovviamente)

Client state

“Non è memorizzato sul server”

(Capitan Ovvio, ovviamente)

Esempi tipici sono i filtri sugli elenchi, il numero di pagina corrente etc

Client state

“Non è memorizzato sul server”

(Capitan Ovvio, ovviamente)

Esempi tipici sono i filtri sugli elenchi, il numero di pagina corrente etc

Hanno senso solo sul client corrente e molto spesso sono legati all’utente loggato

Transient client state

Contiene informazioni memorizzate sul client ma non riflesse sulla URL

Transient client state

Contiene informazioni memorizzate sul client ma non riflesse sulla URL

Esempio: posizione del video su Youtube

Transient client state

Contiene informazioni memorizzate sul client ma non riflesse sulla URL

Esempio: posizione del video su Youtube

Permettono di arricchire l'esperienza utente pur non manipolando gli altri stati

UI client state

Contiene
informazioni locali
su alcuni
componenti

UI client state

Contiene informazioni locali su alcuni componenti

Esempio: sidebar aperta/chiusa

UI client state

Contiene informazioni locali su alcuni componenti

Esempio: sidebar aperta/chiusa

Permettono di ripristinare l'aspetto della UI

Principi base

Single source of truth

Tutti i dati sono contenuti in un unico store.

Spesso è un POJO

Principi base

Single source of truth

Tutti i dati sono contenuti in un unico store.

Spesso è un POJO

Lo stato è in sola lettura

ImmutableJS&C permettono di rispettare facilmente questo principio

Principi base

Single source of truth

Tutti i dati sono contenuti in un unico store.

Spesso è un POJO

Lo stato è in sola lettura

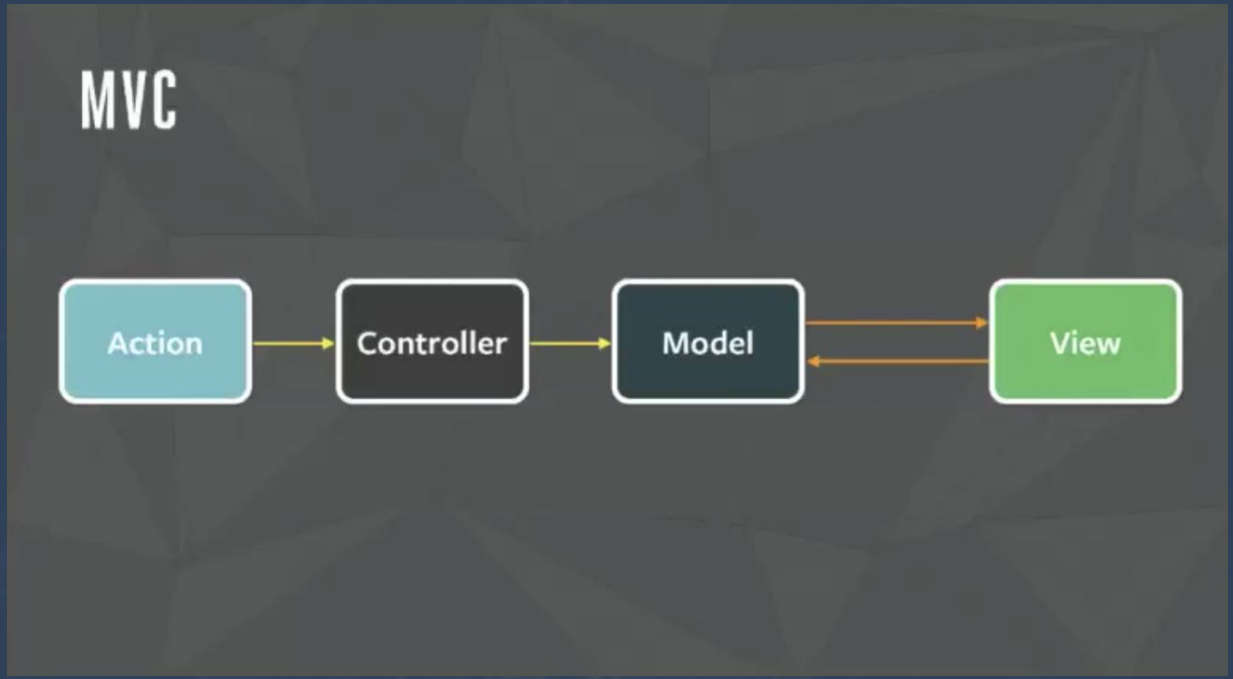
ImmutableJS&C permettono di rispettare facilmente questo principio

Le modifiche sono effettuate tramite funzioni attivate da azioni

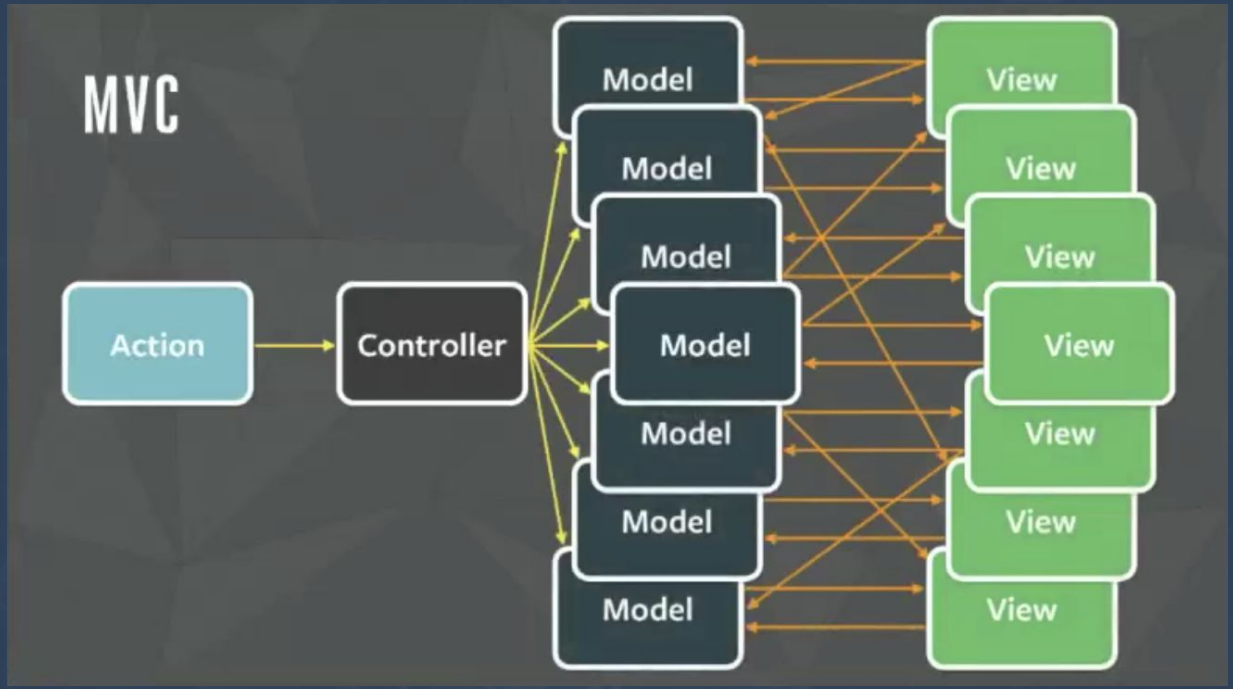
Extra

Lo strato di gestione deve essere completamente separato dal resto dell'applicazione ed interagire esclusivamente tramite eventi in ingresso allo store e una architettura pub/sub per monitorare i cambiamenti

Il design che vorrei



Il design che ho



Redux *e i suoi fratelli*

Elementi

Actions

Descrivono l'informazione che deve essere gestita dallo store

Elementi

Actions

Descrivono l'informazione che deve essere gestita dallo store

Action handlers

Intercettano le azioni e creano un nuovo stato applicativo

Anche noti come reducer

Elementi

Actions

Descrivono l'informazione che deve essere gestita dallo store

Es: {type: NEW_MOVE, cell:3 }

Action handlers

Intercettano le azioni e creano un nuovo stato applicativo

Anche noti come reducer

Store

Si occupa di mettere insieme azioni e reducer

Un esempio minimale

Tic-Tac-Toe

Azioni

```
export function move(index, symbol) {
  return {
      type: 'GAME_MOVE',
      index,
      symbol
  };
}
```

```
export function resetGame() {
  return {
      type: 'RESET_GAME'
  };
}
```

State

```
{
  board: ['', '', '','', '', '','', '', ''],
  currentPlayer: 'O',
  winner: 'n'
};
```

Reducer

```
export function tictactoe(state = initialState, action) {
  switch (action.type) {
      case 'GAME_MOVE':
          if (!state.board[action.index] && state.winner === 'n') {
              const newBoard = _.cloneDeep(state.board);
              newBoard[action.index] = action.symbol;
              return {
                  ...state,
                  board: newBoard,
                  currentPlayer: action.symbol === 'O' ? 'X' : 'O',
                  winner: checkWinner(newBoard)
              };
          }
          return state;
      case 'RESET_GAME':
          return {
              ...initialState
          };
      default:
          return state;
  }
}
```

```
export function tictactoe(state = initialState, action) {
  switch (action.type) {
      case 'GAME_MOVE':
          if (!state.board[action.index] && state.winner === 'n') {
              const newBoard = _.cloneDeep(state.board);
              newBoard[action.index] = action.symbol;
              return {
                  ...state,
                  board: newBoard,
                  currentPlayer: action.symbol === 'O' ? 'X' : 'O',
                  winner: checkWinner(newBoard)
              };
          }
          return state;
      case 'RESET_GAME':
          return {
              ...initialState
          };
      default:
          return state;
  }
}
```

REDUCER

(sì, è una normalissima funzione)

```
export function tictactoe(state = initialState, action) {
  switch (action.type) {
      case 'GAME_MOVE':
          if (!state.board[action.index] && state.winner === 'n') {
              const newBoard = _.cloneDeep(state.board);
              newBoard[action.index] = action.symbol;
              return {
                  ...state,
                  board: newBoard,
                  currentPlayer: action.symbol === 'O' ? 'X' : 'O',
                  winner: checkWinner(newBoard)
              };
          }
          return state;
      case 'RESET_GAME':
          return {
              ...initialState
          };
      default:
          return state;
  }
}
```

MOSSA

```
export function tictactoe(state = initialState, action) {
  switch (action.type) {
      case 'GAME_MOVE':
          if (!state.board[action.index] && state.winner ==
              const newBoard = _.cloneDeep(state.board);
              newBoard[action.index] = action.symbol;
              return {
                  ...state,
                  board: newBoard,
                  currentPlayer: action.symbol === 'O' ? 'X' : 'O',
                  winner: checkWinner(newBoard)
              };
          }
          return state;
      case 'RESET_GAME':
          return {
              ...initialState
          };
      default:
          return state;
  }
}
```

Lo stato è read only*

**Per essere onesti, dovrebbe esserlo ma non lo è.*
*Per evitare errori meglio affidarsi a **Immutable** o simili*

```
export function tictactoe(state = initialState, action) {
  switch (action.type) {
      case 'GAME_MOVE':
          if (!state.board[action.index] && state.winner === '') {
              const newBoard = _.cloneDeep(state.board);
              newBoard[action.index] = action.symbol;
              return {
                  ...state,
                  board: newBoard,
                  currentPlayer: action.symbol === 'O' ? 'X' : 'O',
                  winner: checkWinner(newBoard)
              };
          }
          return state;
      case 'RESET_GAME':
          return {
              ...initialState
          };
      default:
          return state;
  }
}
```

Registro la mossa

```
export function tictactoe(state = initialState, action) {
  switch (action.type) {
      case 'GAME_MOVE':
          if (!state.board[action.index] && state.winner === 'n') {
              const newBoard = _.cloneDeep(state.board);
              newBoard[action.index] = action.symbol;
              return {
                  ...state,
                  board: newBoard,
                  currentPlayer: action.symbol === 'O' ? 'X' : 'O',
                  winner: checkWinner(newBoard)
              };
          }
          return state;
      case 'RESET_GAME':
          return {
              ...initialState
          };
      default:
          return state;
  }
}
```

Restituisco il nuovo stato

```
export function tictactoe(state = initialState, action) {
  switch (action.type) {
      case 'GAME_MOVE':
          if (!state.board[action.index] && state.winner === 'n') {
              const newBoard = _.cloneDeep(state.board);
              newBoard[action.index] = action.symbol;
              return {
                  ...state,
                  board: newBoard,
                  currentPlayer: action.symbol === 'O' ? 'X' : 'O',
                  winner: checkWinner(newBoard)
              };
          }
          return state;
      case 'RESET_GAME':
          return {
              ...initialState
          };
      default:
          return state;
  }
}
```

Spread operator (ES6, 2015)

*E' possibile trovare maggiori informazioni sullo "spread operator" (...) su **MDN***

Gestione eventi

```
// handler click sulla cella
gameMove(index) {
  this.store.dispatch({
    type: 'GAME_MOVE',
    index: index,
    symbol: this.store.getState().currentPlayer
  });
}
```

Gestione cambio stato

```
this.store.subscribe( (state) => {
  this.update();
  if (this.store.getState().winner !== 'n') {
    this.win();
  }
});
```

Giochiamo!

Tic-Tac-Toe

Thanks! Grazie!

Spero che questi suggerimenti ti aiutino
a creare applicazioni migliori.

https://github.com/antonellopasella/webconf-2018-tictactoe

https://antonellopasella.github.io/webconf-2018-tictactoe/

Per qualsiasi richiesta visita il sito pasella.it oppure
manda una email a antonello@pasella.it