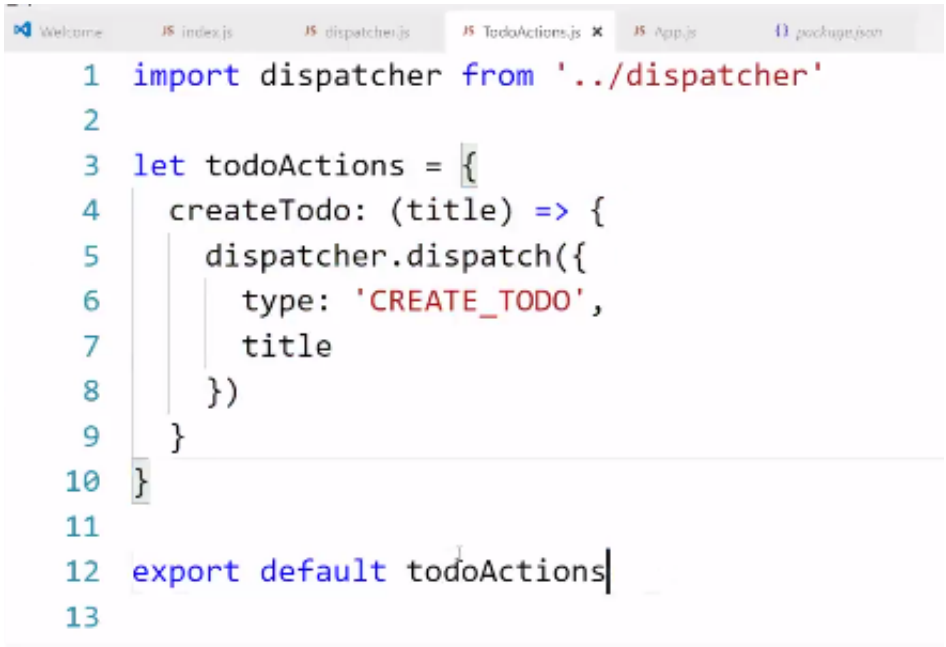
вторник, 4.07.2017

Кратък план за създаване на *flux* архитектура

1. Създаваме си **dispatchera**(*който отговаря за доставянето на action-те към сторовете*)

2. Създаваме си **action**-ите



```
  }

}

export default toDoActions

```

3. Създаваме си **store**-те Във всеки **store** трябва да се импортне **EventEmitter** , за да може когато има промяна някъде конкретния **store** да може да съобщи на компонентите, които го следят че има нови неща(Тоест, за да могат да хвърлят **events(събития)**). Освен **EventEmitter**-а , даден store трябва да има метод **handleaction(action)** , който според **action**-а(действието да вика определен метод, който да го обработи).

1. Импортваме си **EventEmitter**, за да може нашият **Store** да хвърля **евенти(събития)**, когато е настъпила някаква промяна. В конструктора може да има, но може и да няма някакви инициализиращи стойности.

TodoList.js | ToDo.js | ToDoActions.js | ToDoStore.js

```
import { EventEmitter } from 'events'
import dispatcher from '../dispatcher'

class ToDoStore extends EventEmitter {
 constructor () {
  super()

  this.todos = [
    { id: 1, title: 'Go shopping', completed: false },
    { id: 2, title: 'Go walking', completed: false }
  ]
 }
```

2. Описваме **методите** и всички други неща, с които **store-ът** може да работи. В случая това са `createToDo`, `completeToDo`

ТУК Е ВАЖЕН РЕДЪТ *`this.emmit('change')`* , КОЙТО ПРИ ВСЯКА ПРОМЯНА ЕМИТВА СЪБИТИЕ, ЧЕ НЕЩО СЕ Е ПРОМЕНИЛО.

```
 getAll () {
  return new Promise((resolve, reject) => {
   resolve(this.todos.slice(0))
  })
 }

 createToDo (title) {
  const id = this.todos.length + 1
  this.todos.push({
   id,
   title,
   completed: false
```

```
26      })
27
28      this.emit('change')
29    }
30
31    completeToDo (id) {
32      const todo = this.todos.find(todo => todo.id === id)
33      todo.completed = true
34      this.emit('change')
35    }
```

Освен тези неща всеки **Store** трябва да съдържа в себе си и **handleAction** метод, който получава **action** и на базата на неговия тип решава какво да прави. В случая ако типът на екшънът е **CREATE_TODO** , ще се извика методът, който описахме малко по-горе **createToDo**..

```
handleAction (action) {
  switch (action.type) {
    case 'CREATE_TODO': {
      this.createToDo(action.title)
      break
    }
    case 'COMPLETE_TODO': {
      this.completeToDo(action.id)
      break
    }

    default: {
      break
    }
  }
}
```

След това този **handleAction** трябва да го регистрираме в **dispatcher-a**

```
55    let todoStore = new ToDoStore()
56
57    dispatcher.register(todoStore.handleAction.bind(todoStore))
58
59    export default todoStore
60
```