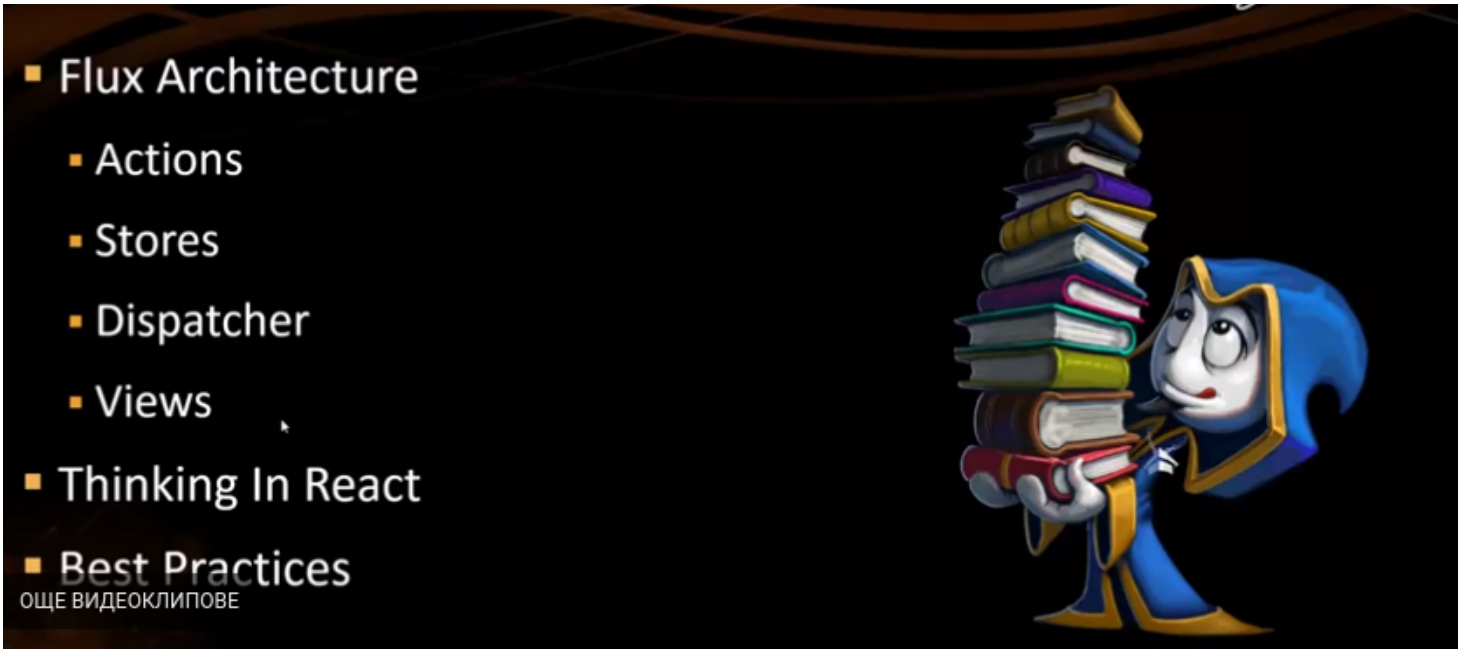
сряда, 28.06.2017

Thinking in React, Architecture & Best Practices

Архитектура Flux
Самата архитектура се състои от показаните по-долу на снимката(**Actions**, **Stores**, **Dispatcher**, **Views**).
Важното е да се знае, че това са компоненти на структурата, а не **React компоненти**



> ***Actions***
> *- Action(Екшъни) описват какъв вид манипулации върху данните имаме в нашето приложение. Примерно в приложение ToDo, action-и могат да бъдат - добави ново ToDo, маркирай ToDo като completed или изтрий ToDo.* ***ТЕ САМО ОПИСВАТ КАКВИ ДАННИ И КАКВО СЕ ПРОМЕНЯ , НО НЕ ПРОМЕНЯТ НИЩО.***

> ***Stores***
> *- Stores - тяхната задача е да дават данни и по никакъв начин да* ***НЕ ПОЗВОЛЯВАТ ТЕЗИ ДАННИ ДА БЪДАТ ПРОМЕНЯНИ ПРЕЗ ТЯХ.***

> ***Dispatcher***
> *- Dispatcher - позволява ни да прехвърляме* ***actions*** *към* ***stores****.* ***Идеята*** *е когато примерно създадем createToDo action и искаме да се изпълни, това означава че Dispatchera ще предаде на този store, който отговаря за това нещо и съответно store ще ъпдейтне сам данните.*

> ***Views***
> *- Views - това са реакт компоненти html-чета*

Идеята е всяко едно от тези да отговаря за по едно нещо. **Action**-ът описва какво трябва да се случи и кои са данните, за да се случи. **Dispatcher**-ът е този , който прехвърля тези action-и наляво надясно по сторовете. **Stores** се грижи данните да бъдат ъпдейтвани и манипулирани, но по никакъв начин да не позволява някой да го прави отвън директно и view-тата са html.

Flux не е фреймуорк, а по-скоро някакво решение , което ни е дадено от фейсбуук за често срещан проблем когато нашето приложение расте.

Идеята при Flux е:

> - ***данните да се намират на <u>едно място</u> и съответно <u>компонентите да комуникират с него</u>,*** *но по никакъв начи да не подават нагоре и надолу някакви неща към други компоненти.*

> - ***позволява ни да променим начинът, по който компонентите вземат и*** *променят данни. Сетването(поставяне на нови данни) не става директно , а чрез съответния action. Казва се - извикваме action-ът ToDo и съответно този, който се е регистрирал, че може да обработва createToDo action, автоматично ще ъпдейтне данните. Вземането на информация става по-лесно, защото тези сторове(stores) дават данните, но по никакъв начин не позволяват на компонентът директно да ги променя.*

Actions(Екшъни)

> - ***това са обекти, които създаваме когато искаме нещо да се промени в*** *нашето приложение.*

> - ***всеки екшън има type пропърти, за да опише какъв е точно екшънът.***

```
let action = {
    type: "CREATE_TODO",
    title: "Go shopping"
}
```

Stores

> - ***това са тези обекти, които получават actions и на базата на тях*** *решават какво да правят. Примерно ако имаме ToDo Store, той получава някакъв екшън , примерно createToDo, updateToDo, completeToDo и тн... и този Store вижда какъв е типът(type) на action-ът и преценява какво да се изпълни и как да се променят данните на базата на този екшън. Сторовете се занимават да държат данните в някакъв state.*

> - ***Те се занимават да държат данните в някакъв state.***

> - ***Имат методи за вземане на данните, но по никакъв начин да ги*** *променят. Идеята е сторовете да нямат възможност някой от външния свят да променя данните чрез тях. През тези сторове не би трябвало да има възможност да се променя информацията директно. Трябва да се извикат екшъни , които да се изпълнят. <u>***Тоест ако искам нещо да променям по данните - трябва да създам екшън, да dispatch-на до store-а и опиша на стора, когато получи такъв екшън какво да се случва***</u>.*

Dispatcher

> - ***идеята му е през него да се разпределят всички екшъни по сторовете и*** *все едно им казва "тези неща трябва да бъдат изпълнени в моето приложение".*

> - ***обикновено когато даден store ъпдейтне някакви данни, той хвърля*** *някакъв event и казва "данните се промениха". И вече компонентите, които зависят от тези данни са се записали(subscribe-нали) към този event(събитие)*

> - ***общо взето идеята е , че не можем да dispatch-ваме екшъни без да*** *преминем през този т. нар. dispatcher.*

Всъщност се случва следното:
Някъде в кода се изпълнява някакъв **action**, примерно **createToDo**. Той има **type**(тип) **CREATE_TODO**(***това е просто пример***) и някакви данни. Това нещо отива в **dispatcher**-а и последният казва "Добре.. аз това нещо ще го доставя до store-а", **store**-вете получават този екшън и всеки от тях преценява дали трябва нещо да промени. И съответно се записва това нещо в store-а и той съответно казва "Аз ъпдейтнах моите данни" и съответно даден компонент, който се е записал(subscribe-нал) за тях казва "А супер има нови данни. Аз ще ъпдейтна състоянието(state) на компонента". В следващият момент, когато потребителят реши нещо да промени или да цъкне на completeToDO, през view-то казваме, че този екшън се изпълнява. Последният съдържа информация, която completeToDo извършва и отива отново в dispatcher-а и последният казва "добре.. искам всички сторове да получат този complete to do и който знае какво да прави с него , да го изпълни".

Този процес е показан на следващата картинка като започва от екшъна на долния ред.