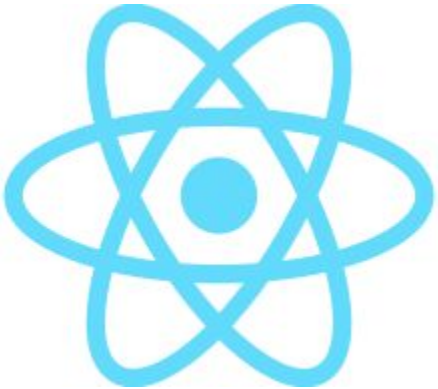
Dièse-Dev 2022

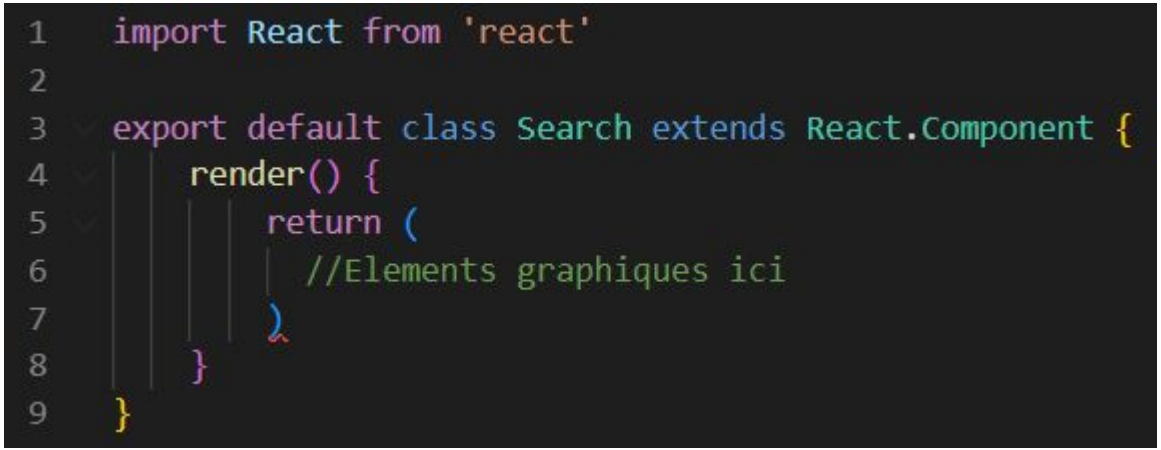
React Native - Cours 2

Architecture et style d’un component custom



Créer un component

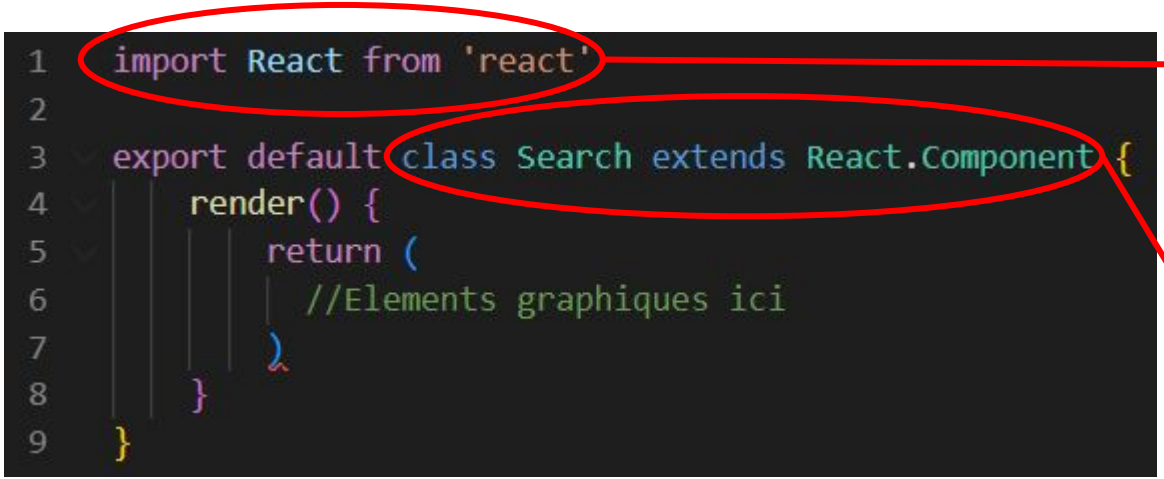
- Convention : un fichier .js par component, le tout dans un dossier “Components” à la racine du projet
- On vas créer un component Search (barre de recherche):
  - Créer le dossier Components à la racine du projet
  - Créer fichier Search.js et le remplir comme cela :



```
import React from 'react'

export default class Search extends React.Component {
    render() {
        return (
          //Elements graphiques ici
        )
    }
}
```

Structure d’un component



Modules importés par le component (ce dont il a besoin)

On définie une classe de type React.Component (donc un component)

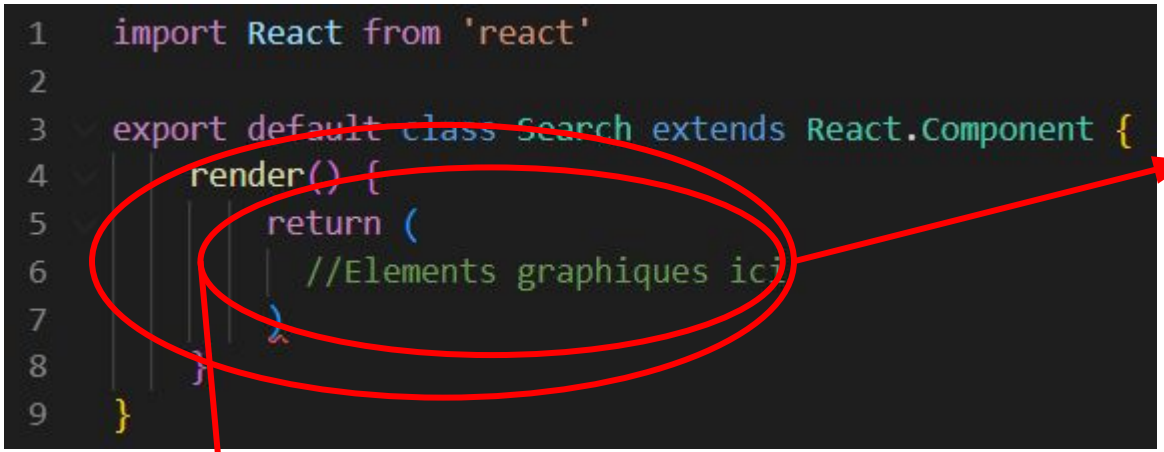
Structure d’un component (2)

```
1  import React from 'react'
2
3  export default class Search extends React.Component {
4      render() {
5          return (
6              //Elements graphiques ici
7          );
8      }
9  }
```

X

Met X comme l’export par défaut du fichier Search.js.
On peut alors utiliser X dans un autre component en faisant “import X from <CheminToFichierSearch>” (cf plus tard)
Ici X est le component Search

Structure d’un component (3)



Fonction render commune à tous les components : Affiche le component

return renvoi les éléments graphiques

On peut calculer des choses entre le render et le return ! (Cf plus tard)

Afficher un component dans render

- On passe à un autre langage : JSX
- Surcouche du JS qui permet de manipuler la hiérarchie des components à la manière du HTML
- Afficher une vue: utilise le component de base View:

  ```
  <View>
    …
  </View>
  ```

  (Avec … le contenue de la vue)
- Afficher une image : utilise le component de base Image :

  `<Image … />` (Avec ... des infos sur l'image)

  Image ne vas pas contenir d'autres components donc se ferme directement avec /> (pas de </Image>)

Parenthèse sur le JSX

- Sans le JSX : <View> … </View> devient :
  `React.createElement('View', null, … )`
- Babel est le compilateur JS qui vas faire passer tout le JSX en fonctions moches (merci à lui)

Remplir le render

- Importer les component dont on a besoin (Ici View, TextInput et Button)
- Retourner une View composée d'un TextInput (pour que l'utilisateur tape un texte) et un Button (pour lancer une fonction).

```
1  import React from 'react'
2  import { StyleSheet, View, TextInput, Button } from 'react-native'
3
4  export default class Search extends React.Component {
5    render() {
6      return (
7        <View style={styles.view}>
8          <TextInput placeholder='Titre du film'/>
9          <Button title='Rechercher' onPress={() => {}}/>
10       </View>
11     )
12   }
13 }
```

Propriétés d’un component

- Sont des paramètres du component et servent à le personnaliser
- Différentes propriétés pour chaques components (cf la doc des components React Native)
- Dans notre cas:
  - TextInput : “**placeholder**” prend une string qui est le texte affiché dans le TextInput en gris avant que l’utilisateur tape qqchose dedans
  - Button : “**title**” prend une string qui est affichée sur le boutton et “**onPress**” prend une fonction qui est la fonction lancée quand l’utilisateur appuie sur le boutton.

Importer un component

- Notre component principal est App.js (à la racine du projet), on vas afficher notre Search:
  - Importer Search dans App.js
  - Mettre Search dans le render de App.js

```
import React from 'react'
import Search from './Components/Search'

export default class App extends React.Component {
  render() {
    return (
      <Search/>
    )
  }
}
```

Rendu

- Rendu différent entre android et ios: React Native utilise bien les composants natifs de chaque OS

- Notre component Search est affiché tout en haut, derrière la barre des tâches. On veut gérer sa position, sa taille, etc…
  -> On vas gérer le style de nos components

Style d’un component

- Chaque component de base admet une propriété “**style**” qui sert à définir son style.
- Styliser son component permet de changer une couleur, définir une taille, ajouter des marges, modifier l'alignement d'éléments entre eux, ajouter des bordures, etc…
- Peut se définir comme cela : `<View style={{marginTop:20}}>`
  Mais ça devient vite le bordel quand on met beaucoup de style
- On utilise plutôt une feuille de style

Feuille de style

- Une feuille de style (StyleSheet) permet de mettre tous les styles utilisé par un component dans une constante, généralement mise en bas du fichier (hors de la définition du component)
- API faite par React Native existe, rajoutez “StyleSheet” dans les imports de react-native
- Définie comme cela:

```
const styles = StyleSheet.create({
  view:{
    marginTop:20
  }
})
```

```
<View style={styles.view}>
```

Style d’un component (3)

- On rajoute aussi du style au TextInput (ne pas oublier de le rajouter dans le component TextInput avec styles={...}) :

```
7           <View style={styles.view}>
8             <TextInput placeholder='Titre du film' style={styles.textinput}/>
9             <Button title='Rechercher' onPress={() => {}}/>
10          </View>
11        )
12      }
13    }
14
15    const styles = StyleSheet.create({
16      view:{
17        marginTop:20
18      },
19
20      textinput: {
21        marginLeft: 5,
22        marginRight: 5,
23        height: 50,
24        borderColor: '#000000',
25        borderWidth: 1,
26        paddingLeft: 5
27      }
28    })
```

Flexbox

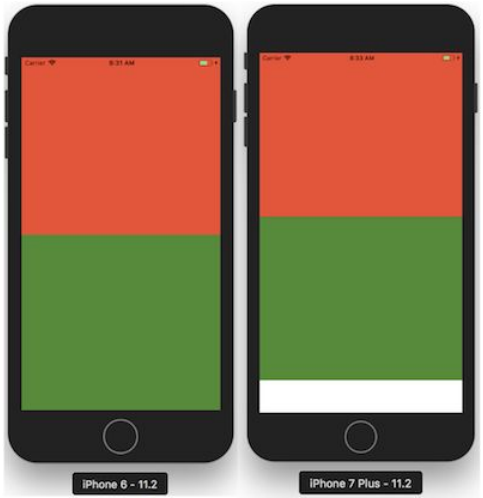
- Mode de mise en page utilisé pour le web
- Permet de gérer les tailles et positions des éléments dynamiquement
- On construit les vues comme des boites flexibles
- Permet d’adapter les tailles en fonction des dimensions de l’écran

Flexbox (2)

- Exemple d'utilité :
  On veut couper une vue verticalement en deux sur un écran de 670px

```
render() {
    return (
      <View style={{ backgroundColor: 'yellow' }}>
        <View style={{ height: 335, backgroundColor: 'red' }}></View>
        <View style={{ height: 335, backgroundColor: 'green' }}></View>
      </View>
    )
}
```

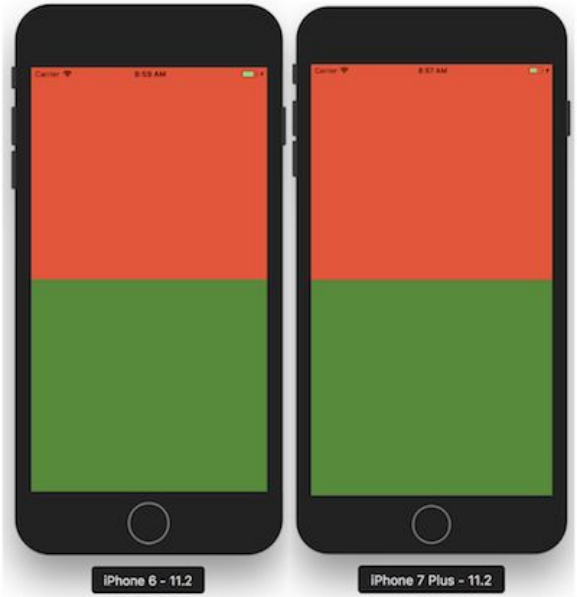
- Rendu ok sur notre téléphone mais nul sur un autre



Style flex

- Soluce: utiliser flex dans le style des éléments.
  flex détermine quelle part de l'écran les composant vont prendre.
  flex: 1, le composant prend toute la place disponible accordé par son parent

```
render() {
    return (
      <View style={{ flex: 1, backgroundColor: 'yellow' }}>
        <View style={{ flex: 1, backgroundColor: 'red' }}></View>
        <View style={{ flex: 1, backgroundColor: 'green' }}></View>
      </View>
    )
}
```



Style flex (2)

- La vue parente prend tout l'espace disponible et ses deux enfants veulent prendre toute la place disponible accordée par leur parent. React Native interprète que les deux enfants doivent se répartir équitablement l'espace
- Si le premier enfant est à flex: 1 et le deuxième a flex: 2, alors le deuxième occuperas les deux tiers de l'espace disponible accordée par leur parent
- Si on enlève flex:1 au parent -> écran blanc, pourquoi ?

```
render() {
    return (
      <View style={{ backgroundColor: 'yellow' }}>
        <View style={{ flex: 1, backgroundColor: 'red' }}></View>
        <View style={{ flex: 1, backgroundColor: 'green' }}></View>
      </View>
    )
}
```

Pourquoi écran blanc ?

- Si on donne aucune valeur de flex, vaut 0 par défaut
- Comme on ne donne aucune taille au parent, React Native vas déterminer sa taille avec la taille de ses composant
- Mais la taille des composants enfants est déterminée par rapport à la taille accordée par le parent via le style flex
- Impossible pour React Native de déterminer les tailles des components
  -> Ils ne seront pas affichés

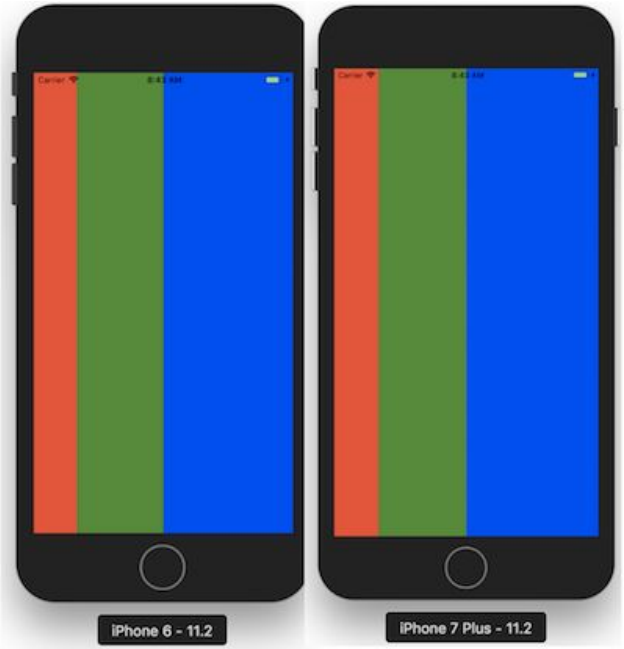
- Il faut garder à l'esprit que si on met une valeur de flex à un component, son parent doit lui aussi avoir une valeur de flex non nulle ou une taille définie (avec height)

Pourquoi écran blanc ? (2)

- En général : on met le flex de la vue principale d’un component à 1 pour éviter les problèmes (sauf sur les component à taille définie)
- Les explications théoriques des flex sont un peu compliqués a comprendre, la pratique expliqueras mieux

Gérer l’alignement

- Par défaut les différents components s’aligne verticalement, on peut changer ça en rajoutant “flex-direction : ‘row’ “



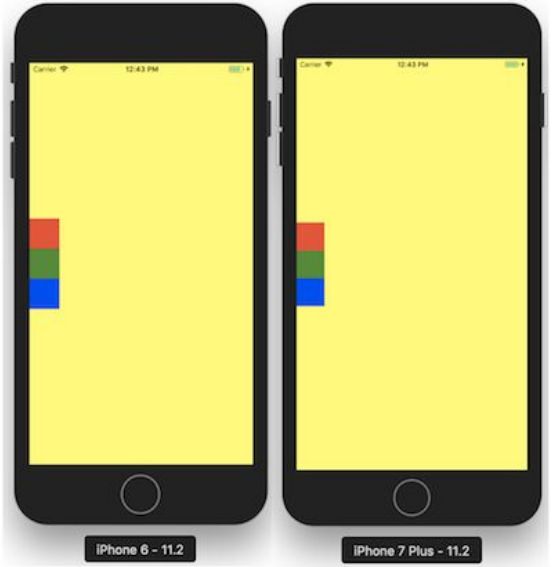

```
render() {
    return (
      <View style={{ flex: 1, flexDirection: 'row', backgroundColor: 'yellow' }}>
        <View style={{ flex: 1, backgroundColor: 'red' }}></View>
        <View style={{ flex: 2, backgroundColor: 'green' }}></View>
        <View style={{ flex: 3, backgroundColor: 'blue' }}></View>
      </View>
    )
}
```

On note que l’on ne voit pas le jaune de la vue parente, c’est normal car ses enfants sont affichés “par dessus”

Gérer l’alignement (2)

- Dans le cas où les enfants ne prennent pas toute la place, on peut gérer leur alignement avec “justify-content” :
  - “center”/“flex-start”/“flex-end”: met tous les enfants au milieu/début/fin
  - “space-between” : répartit les components sur tout l’espace
  - …. cf la doc

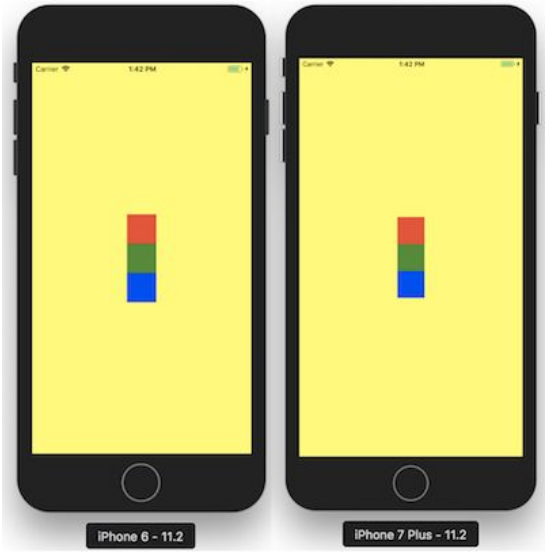
```
render() {
    return (
      <View style={{ flex: 1, justifyContent: 'center', backgroundColor: 'yellow' }}>
        <View style={{ height: 50, width: 50, backgroundColor: 'red' }}></View>
        <View style={{ height: 50, width: 50, backgroundColor: 'green' }}></View>
        <View style={{ height: 50, width: 50, backgroundColor: 'blue' }}></View>
      </View>
    )
}
```



Gérer l’alignement (3)

- justify-content gère l’alignement sur l’axe principal (flex-direction), pour gérer l’alignement sur l’axe secondaire : utiliser “alignItems”
- Peut mixer tout ça de pleins de façons différentes

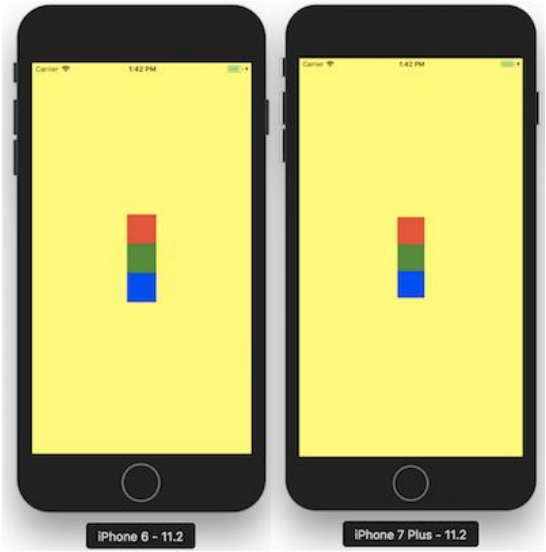
```
render() {
    return (
      <View style={{ flex: 1, justifyContent: 'center', alignItems: 'center', backgroundColor:
'yellow' }}>
        <View style={{ height: 50, width: 50, backgroundColor: 'red' }}></View>
        <View style={{ height: 50, width: 50, backgroundColor: 'green' }}></View>
        <View style={{ height: 50, width: 50, backgroundColor: 'blue' }}></View>
      </View>
    )
}
```



Gérer l’alignement (3)

- justify-content gère l’alignement sur l’axe principal (flex-direction), pour gérer l’alignement sur l’axe secondaire : utiliser “alignItems”
- Peut mixer tout ça de pleins de façons différentes

```
render() {
    return (
      <View style={{ flex: 1, justifyContent: 'center', alignItems: 'center', backgroundColor:
'yellow' }}>
        <View style={{ height: 50, width: 50, backgroundColor: 'red' }}></View>
        <View style={{ height: 50, width: 50, backgroundColor: 'green' }}></View>
        <View style={{ height: 50, width: 50, backgroundColor: 'blue' }}></View>
      </View>
    )
}
```



Le style s’apprend avec la pratique

- J’ai présenté que la base théorique de quelques éléments de style
- Je vais pas tout présenter, c’est déjà assez chiant comme ça
- Vous apprendrez le reste en pratiquant : si vous voulez mettre un texte en italique : “React Native italic text” sur google et vous lisez la doc des components et de leur style ou vous trouvez qqun qui a posé la question sur stack overflow

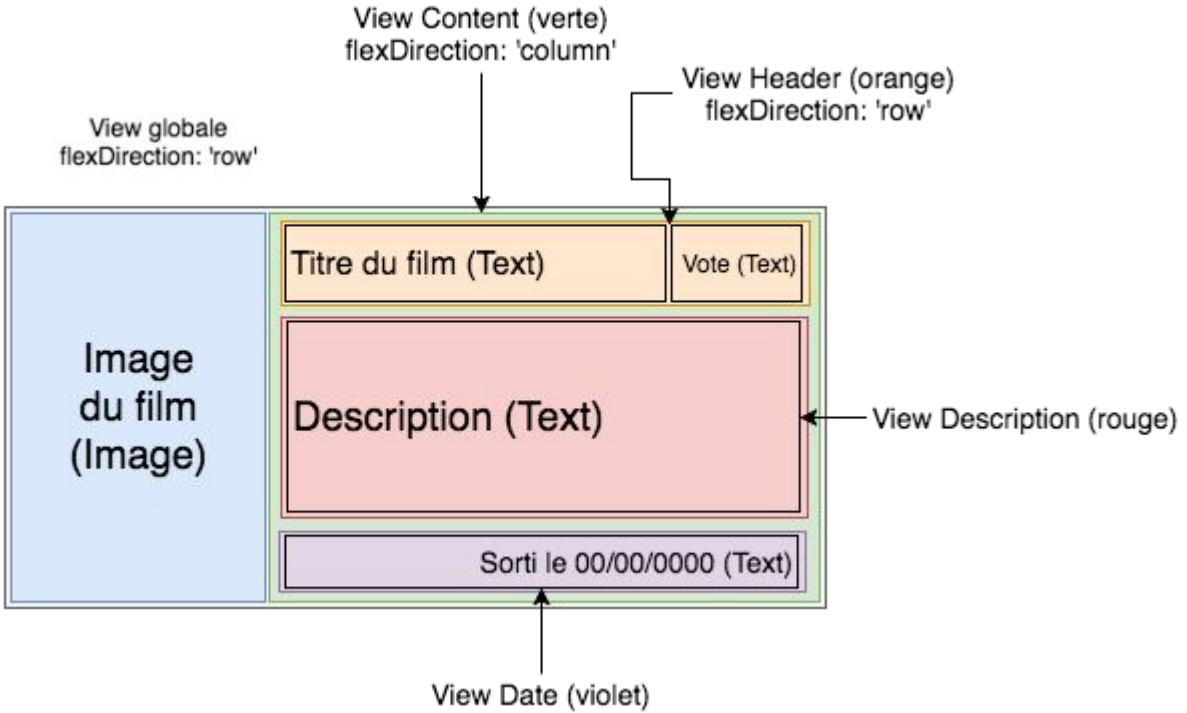
Pratique

- On veut afficher une liste de film dans notre app. La liste sera composé d’éléments comme ceci :

- On vas faire un template que l’on vas utiliser plus tard de cette forme :

Pratique (2)

- Créez un component FilmItem.js qui affichera le résultat voulu
- Petite aide sur la structure :



Tips: mettez la couleur de fond en rouge si vous savez plus très bien quelle taille fait votre component
backgroundcolor: ‘red’ dans la feuille de style

Résultat

- App.js : (Pour bien voir notre component)

```
import React from 'react'
import Search from './Components/Search' //Pas utilisé
import FilmItem from './Components/FilmItem'
import { View, StyleSheet } from 'react-native'

export default class App extends React.Component {
  render() {
    return(
      <View style={styles.mainContainer}>
        <FilmItem/>
      </View>
    )
  }
}

const styles = StyleSheet.create({
  mainContainer:{
    flex:1,
    alignItems:'center',
    justifyContent:'center',
  }
})
```

Résultat (2)

- Résultat visuel avant le code :

- Avec:
  - Bleu clair : background de la vue globale (pas visible)
  - Gris : vue header
  - Orange: texte titre film
  - Violet: texte vote film
  - Jaune: texte description
  - Rouge : texte date

Résultat (3)

- Structure des components:

- Pas besoin de mettre des vue autour de description et date en fait (une vue sert en général à contenir plusieurs components)

(Le code seras sur git, rdv sur le serv discord de dièse-dev)

```jsx
import React from 'react'
import { StyleSheet, View, Text, Image } from 'react-native'

export default class FilmItem extends React.Component {
  render() {
    return (
      <View style={styles.mainContainer}>

        <Image style={styles.image} source={{uri:"lien http"}}/>

        <View style={styles.contentContainer}>

          <View style={styles.headerContainer}>
            <Text style={styles.titleText}>Titre du film</Text>
            <Text style={styles.voteText}>Vote</Text>
          </View>

          <Text style={styles.descriptionText}>Description</Text>
          <Text style={styles.dateText}>Sorti le 00/00/0000</Text>

        </View>

      </View>
    )
  }
}
```

Résultat (4)

- Feuille de style:

```
28  const styles = StyleSheet.create({
29    mainContainer: {
30      height: 200,
31      width:'90%',
32      flexDirection: 'row',
33      backgroundColor:'lightblue'
34    },
35
36    image: {
37      flex:1,
38    },
39    contentContainer: {
40      flex: 2,
41    },
43    headerContainer: {
44      flexDirection: 'row',
45      justifyContent:'space-between',
46      backgroundColor:'grey'
47    },
48    titleText: {
49      fontWeight: 'bold',
50      fontSize: 20,
51      backgroundColor:'orange'
52    },
53    voteText: {
54      alignSelf:'flex-end',
55      fontWeight: 'bold',
56      fontSize: 20,
57      backgroundColor:'violet'
58    },
60    descriptionText: {
61      flex: 1,
62      backgroundColor:'yellow',
63      fontStyle: 'italic',
64    },
65
66    dateText: {
67      textAlign: 'right',
68      fontSize: 14,
69      backgroundColor:'red'
70    }
71  })
```

Conclusion et prochain cours

- On a appris à créer et styliser une architecture de components de base complexe (un gros FilmItem.js)
- Dans le prochain cours on vas apprendre à communiquer des informations entre plusieurs components (Ex: l'utilisateur tape une info dans Search, comment modifier les éléments de FilmItem en accord avec sa recherche ?)
- Fin du prochain cours: l'utilisateur pourras taper un texte et l'application afficheras (grâce à la base de données des films de IMDb) une liste de FilmItem de film correspondant à la recherche

- Bonnes vacances !