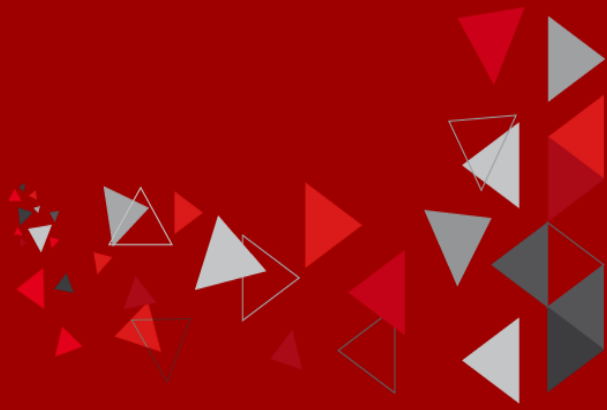
esprit
Se former autrement
HONORIS UNITED UNIVERSITIES

Composants de classe

United Nations Educational, Scientific and Cultural Organization
UNESCO Chair "Project-based learning" ESPRIT School of engineering, Tunisia
EUR-ACE® Délivrée par la Commission des Titres d'Ingénieur
CONFÉRENCE DES GRANDES ÉCOLES
cdio CONCEIVE DESIGN IMPLEMENT OPERATE

- Composant React
- Cycle de vie d’un composant React



Composant de Classe

Composant de Classe

- Les composants de classes sont ,comme son nom l'indique ,qui héritent de la class **React.Component.**

- Les composants de class disposent d'une méthode **render()** qui retourne le rendu du composant.

```
class Welcome extends React.Component {
  render() {
    return <h1>Bonjour, {this.props.name}</h1>;
  }
}
```

Composant de Classe

- Dans un composant de classe, on peut définir plusieurs méthodes qui sont propres à ce dernier.
- Ces méthodes peuvent changer le state ou pas selon le besoin.
- On peut distinguer deux types de méthodes:
  - Les méthodes attachées (binded) à la classe.
  - Les méthodes non attachées.

Composant de Classe

Exemple :

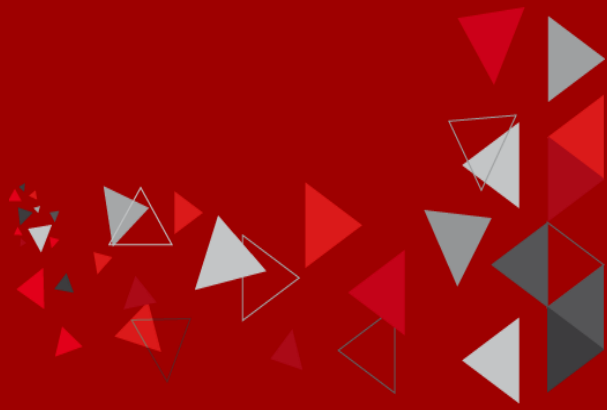
```
export default class Product extends Component {
  constructor(props) {
    super(props);
    this.state = { likes: 0 };
    //méthode attachée
    this.addLikes = this.addLikes.bind(this);
  }
  addLikes() {
    this.setState((oldState) => ({
      likes: oldState.likes + 1,
    }));
  }
  //méthode non attachée
  displayMessage() {
    console.log("hello it's me !!!");
  }

  render() {
    return (
      <div>
        <p>Like : {this.state.likes}</p>
        <button onClick={this.addLikes}>Like</button>
        <button onClick={this.displayMessage}>Click me</button>
      </div>
    );
  }
}
```

Composant de Classe

- React nous donne accès grâce au **“class component”** à plusieurs méthodes prédéfinies tout au long du cycle de vie d’un composant.

- Ces méthodes de cycles de vie nous permettent de modifier certains aspects du composant.

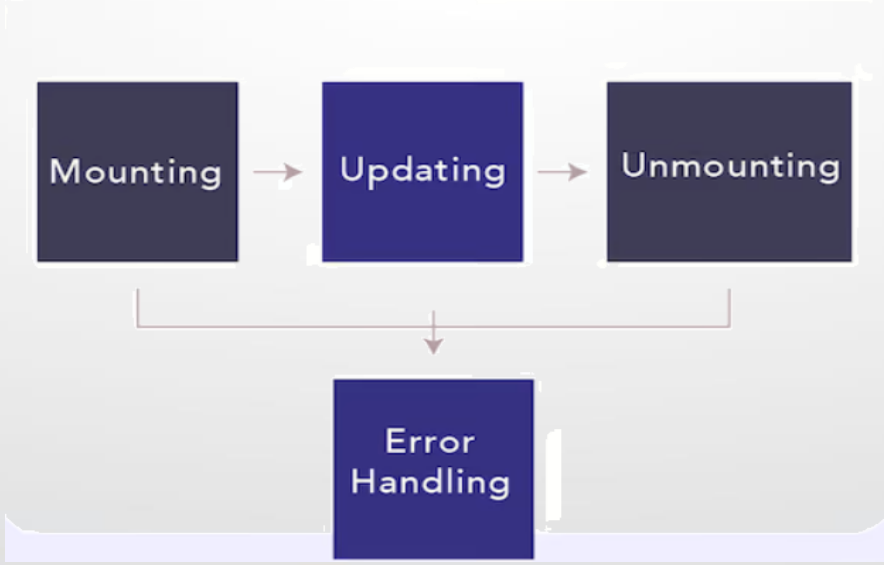


Cycle de vie d’un composant React

Cycle de vie d'un composant React

On peut diviser le cycle de vie d'un composant React en 4 grandes phases :

1. **Mounting** (l'étape d'ajout des nœuds au DOM)
2. **Updating** (l'étape de modification / re-rendering des nœuds du DOM)
3. **Unmounting** (l'étape de suppression des nœuds du DOM)
4. **Error Handling** (l'étape de vérification d'erreur)



Cycle de vie d'un composant React

Méthodes du « Mounting » : Montage

Lors de cette phase, React va appeler ces méthodes dans l'ordre:

1. **Constructor()**
2. static **getDerivedStateFromProps()**
3. **Render()**
4. **componentDidMount()**

Cycle de vie d'un composant React

Méthodes du « Mounting » : Montage

Exemple :

```
export default class Product extends Component {
  constructor(props) {
    super(props);
    console.log("je suis le constructeur");
  }

  static getDerivedStateFromProps(props, state) {
    console.log(props.name);
    console.log("getting props and init states !!!");
    return null;
  }
  componentDidMount() {
    console.log("component did mount !!!");
  }


  render() {
    return (
      <div>
        {console.log("rendering")}
        <p>Like : {this.state.likes}</p>
        </div>
    );
  }
}
```

1 est appelée lorsque le composant est lancé, et c'est où nous allons configurer l'état initial et d'autres valeurs initiales.

2 C'est l'endroit naturel pour définir l'objet d'état **state** basé sur les props initiales.

4 La méthode est appelée après le rendu du composant.

3 Le rendu du composant dans notre page html.

Cycle de vie d'un composant React

Méthodes du « Updating » : Mise à jour

Une mise à jour du composant peut être le résultat d'un changement sur l'une des variables props ou state.
Les méthodes ci-dessous seront appelées dans l'ordre suivant:

1. static **getDerivedStateFromProps()**
2. **shouldComponentUpdate()**
3. **render()**
4. **getSnapshotBeforeUpdate()**
5. **componentDidUpdate()**

Cycle de vie d'un composant React

Méthodes du « Updating » : Mise à jour

Exemple :

```
export default class Product extends Component {
  constructor(props) {
    super(props);
    this.addLikes = this.addLikes.bind(this);
    this.state = { likes: 0, name: "init" };
  }

  static getDerivedStateFromProps(props, state) {
    return null;
  }
  shouldComponentUpdate(props, state) {
    if (state.likes > 2) {
      console.log(state.likes);
      return false;
    } else {
      return true;
    }
  }

  getSnapshotBeforeUpdate(prevProps, prevState) {
    console.log(prevProps);
    console.log(prevState);

    return { prevProps, prevState };
  }
```

1 La première méthode appelée lorsqu'un composant est mis à jour. C'est toujours l'endroit naturel pour définir l'objet d'état **state** basé sur les props initial.

2 La méthode où vous pouvez renvoyer une valeur booléenne qui spécifie si React doit continuer le rendu ou non.

4 Vous avez accès aux **props** et **state** avant la mise à jour pour vérifier les valeurs.

Cycle de vie d'un composant React

Méthodes du « Updating » : Mise à jour

```
.....................
  componentDidUpdate() {
    console.log("current state " + this.state);
  }
  addLikes() {
    this.setState((oldState) => ({
      likes: oldState.likes + 1,
      name: "setState",
    }));
    if (false) {
    }
    console.log(this.state.name);
  }

  render() {
    return (
      <div>
        {console.log("rendering")}
        <p>Like : {this.state.likes}</p>
        <p>Like : {this.state.name}</p>
        <button onClick={this.addLikes}>Like</button>
      </div>
    );
  }
}
```

5

La méthode est appelée après la mise à jour du composant.

3

render() est appelée lorsqu'un composant est mis à jour.

Cycle de vie d'un composant React

Méthodes du « Unmounting » : Démontage

Cette méthode va être appelée lorsqu'un composant va être supprimée du DOM.

1. **componentWillUnmount**()

Cycle de vie d'un composant React

Méthodes du « Unmounting » : Démontage

```
componentWillUnmount() {
  console.log("number " + this.props.id + " is dead");
}
```

La méthode est appelée lorsque le composant va être supprimé du DOM.

Exemple :

```
export default class Product extends Component {
  constructor(props) {
    super(props);
    this.addLikes = this.addLikes.bind(this);
    this.removeLikes = this.removeLikes.bind(this);
    this.state = { likes: 0 };
  }
  removeLikes() {
    this.setState((oldState) => ({
      likes: oldState.likes + -1,
    }));
  }

  addLikes() {
    this.setState((oldState) => ({
      likes: oldState.likes + 1,
    }));
  }

  render() {
    return (
      <div>
        <button onClick={this.addLikes}>ADD</button>
        {this.state.likes != 0 && (
          <button onClick={this.removeLikes}>REMOVE</button>
        )}
        {[...Array(this.state.likes)].map((value, index) => (
          <Cell key={index} id={index} />
        ))}
      </div>
    );
  }
}

class Cell extends Component {
  constructor(props) {
    super(props);
  }
  componentWillUnmount() {
    console.log("number " + this.props.id + " is dead");
  }

  render() {
    return <h1>Cell {this.props.id}</h1>;
  }
}
```

Cycle de vie d'un composant React

Méthodes du « Error Handling » : Gestion d'erreurs

S'il y'a une erreur dans l'une des méthodes du cycle de vie du composant ou dans ses fils, les méthodes suivantes seront appelées:

1. Static **getDerivedStateFromError**()
2. **componentDidCatch**()

Merci de votre attention