

React JS

Laurence
X  NATIS



Compétence demandée :
**Comprendre les tâches
asynchrones**

1. Les traitements **asynchrones**
2. Les **callbacks** et **promises**
3. Les **REST API**

1. L'ASYNCHRONE

Les traitements **asynchrones** permettent
de faire des **traitements parallèles**.

2. CALLBACKS & PROMISES

```
function validerPrixBonbon(prix, successCallback, failureCallback) {  
  if (prix < 4.5) {  
    successCallback("On peut l'acheter !");  
  } else {  
    failureCallback("Trop cher pour un bonbon");  
  }  
}  
  
function successCallback(résultat) {  
  console.log("L'opération a réussi avec le message : " + résultat);  
}  
  
function failureCallback(erreur) {  
  console.error("L'opération a échoué avec le message : " + erreur);  
}  
  
let prixDuBonbon = 5.0;  
validerPrixBonbon(prixDuBonbon, successCallback, failureCallback);
```

```
function validerPrixBonbon(prix, successCallback, failureCallback) {
  if (prix < 4.5) {
    successCallback("On peut l'acheter !");
  } else {
    failureCallback("Trop cher pour un bonbon");
  }
}

function successCallback(résultat) {
  console.log("L'opération a réussi avec le message : " + résultat);
}

function failureCallback(erreur) {
  console.error("L'opération a échoué avec le message : " + erreur);
}

let prixDuBonbon = 5.0;
validerPrixBonbon(prixDuBonbon, successCallback, failureCallback);
```

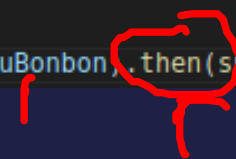


```
function validerPrixBonbon(prix) {
  return new Promise((successCallback, failureCallback) => {
    if (prix < 4.5) {
      successCallback("On peut l'acheter !");
    } else {
      failureCallback("Trop cher pour un bonbon");
    }
  });
}

function successCallback(résultat) {
  console.log("L'opération a réussi avec le message : " + résultat);
}

function failureCallback(erreur) {
  console.error("L'opération a échoué avec le message : " + erreur);
}

let prixDuBonbon = 5.0;
validerPrixBonbon(prixDuBonbon).then(successCallback, failureCallback);
```



Les « promises » introduisent un
paradigme de développement différent :
event-driven development.

3. LES REST API

Les **REST API** permet à plusieurs systèmes de **communiquer entre eux** de manière banalisée (au **format JSON** majoritairement).

Le **CRUD** est constitué des opérations de base de manipulation de données :

- **C** : Create
- **R** : Read/Retrieve
- **U** : Update
- **D** : Delete

```
list() {  
  return fetch('**URL DE L'API**', {  
    method: 'GET',  
    headers: {  
      'Accept': 'application/json',  
      'Content-Type': 'application/json',  
    }  
  }).then((response) => response.json());  
}
```

```
create(data) {  
  return fetch('**URL DE L'API**', {  
    method: 'POST',  
    headers: {  
      'Accept': 'application/json',  
      'Content-Type': 'application/json',  
    },  
    body: JSON.stringify(data),  
  }).then((response) => response.json());  
}
```

```
update(data) {  
  return fetch('**URL DE L'API**', {  
    method: 'PUT',  
    headers: {  
      'Accept': 'application/json',  
      'Content-Type': 'application/json',  
    },  
    body: JSON.stringify(data),  
  }).then((response) => response.json());  
}
```

```
delete(id) {  
  return fetch('**URL DE L'API**', {  
    method: 'DELETE',  
    headers: {  
      'Accept': 'application/json',  
      'Content-Type': 'application/json',  
    }  
  }).then((response) => response.json());  
}
```