Programmation d'applications mobiles 1

Séance de laboratoire 2

INTRODUCTION À REACT.JS

@ Gabriel Y. Gamy

Table of Contents

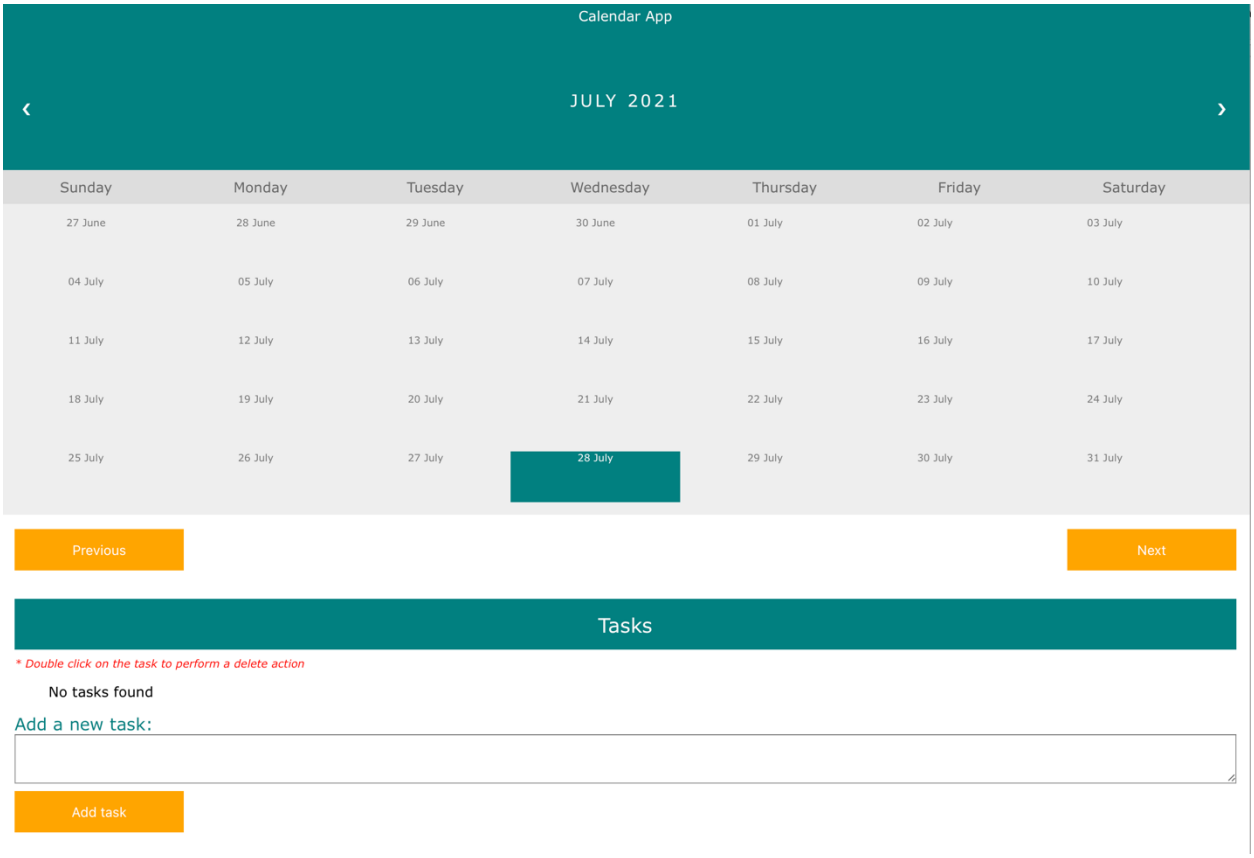
Objectif :

L'objectif de cette séance de laboratoire est:

- De se familiariser avec les concepts de base en React.js
- De comprendre la syntaxe JSX
- D'utiliser les « props » pour communiquer entre composants
- De mettre en pratique la gestion interne des composants avec « state »
- D'utiliser la technique de communication parent-enfant et le « Two Way Binding » des éléments

Calendrier en React.js :

Nous voulons construire un calendrier qui affiche le mois courant et les jours associés.
Voici un aperçu du rendu final de notre calendrier :



a- Création du projet

L'installation de Node.js est requise pour développer en React.
La première étape consiste à créer le projet à l'aide de la commande ci-dessous :

npx create-react-app calendrier-en-react

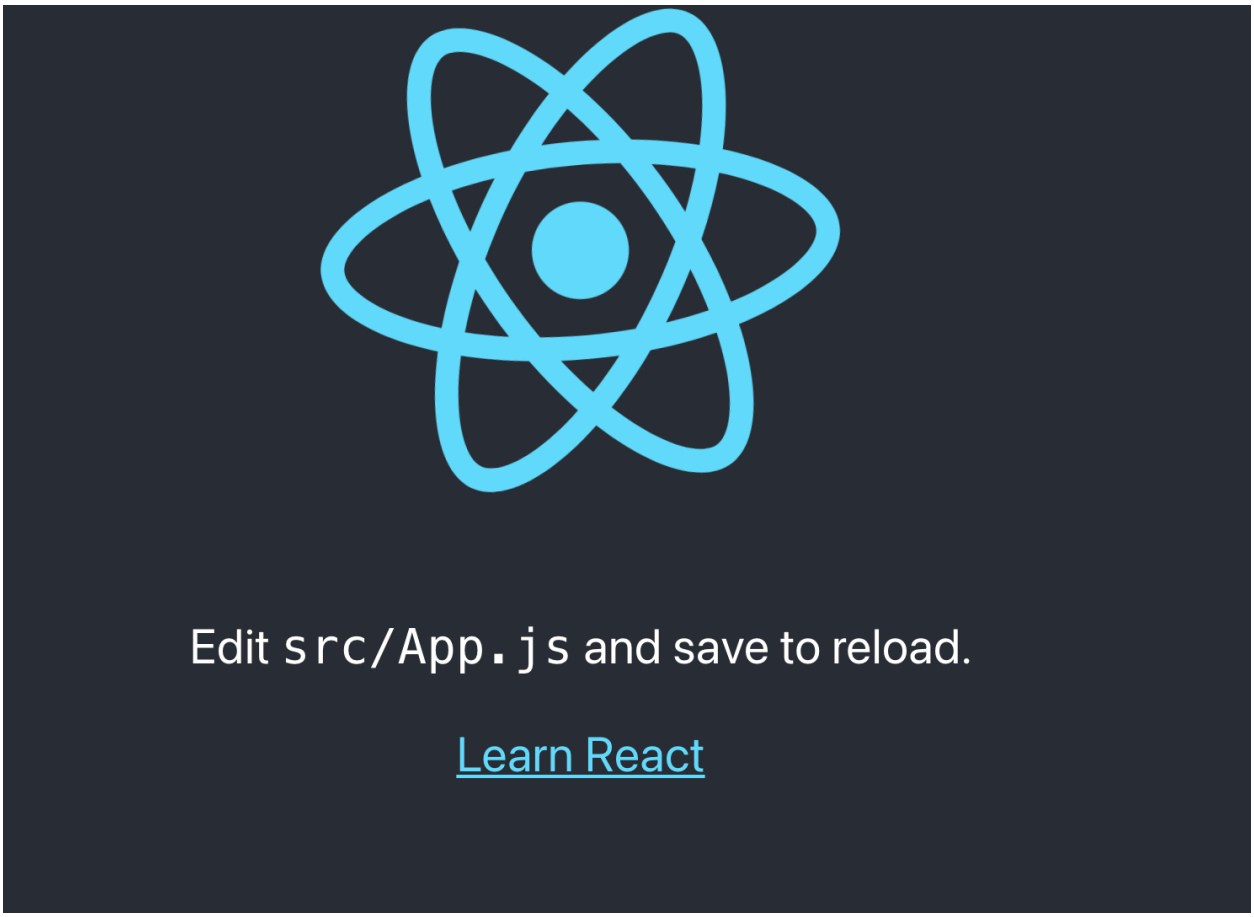
@ Gabriel Y. Gamy

Se rendre ensuite dans le dossier et rouler le projet :

cd calendrier-en-react

npm start

La page de base s'affiche comme à l'écran ci-dessous :



b- Les composants du calendrier

Développer en React consiste à mettre en place une architecture de composants. Chaque composant étant dédié à une responsabilité unique.

En observant le rendu final du calendrier, nous pouvons déjà avoir une idée d'architecture des composants :

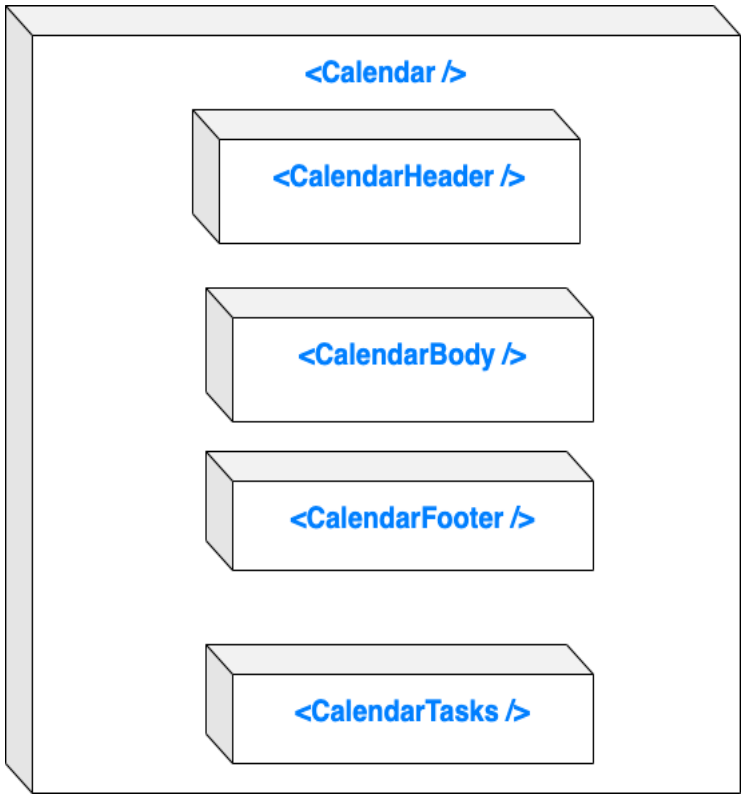
- <CalendarHeader /> : Contient l'entête du calendrier, le mois et jour, les icônes « Previous » et « Next », l'entête des jours de la semaine (en commençant par Dimanche).

- <CalendarBody /> : Contient les jours du calendrier pour le mois sélectionné

- <CalendarFooter /> : Contient les boutons d'actions « Previous » et « Next »

- <CalendarTasks /> : Contient les tâches associées au jour sélectionné ainsi qu'un formulaire pour saisir une nouvelle tâche.

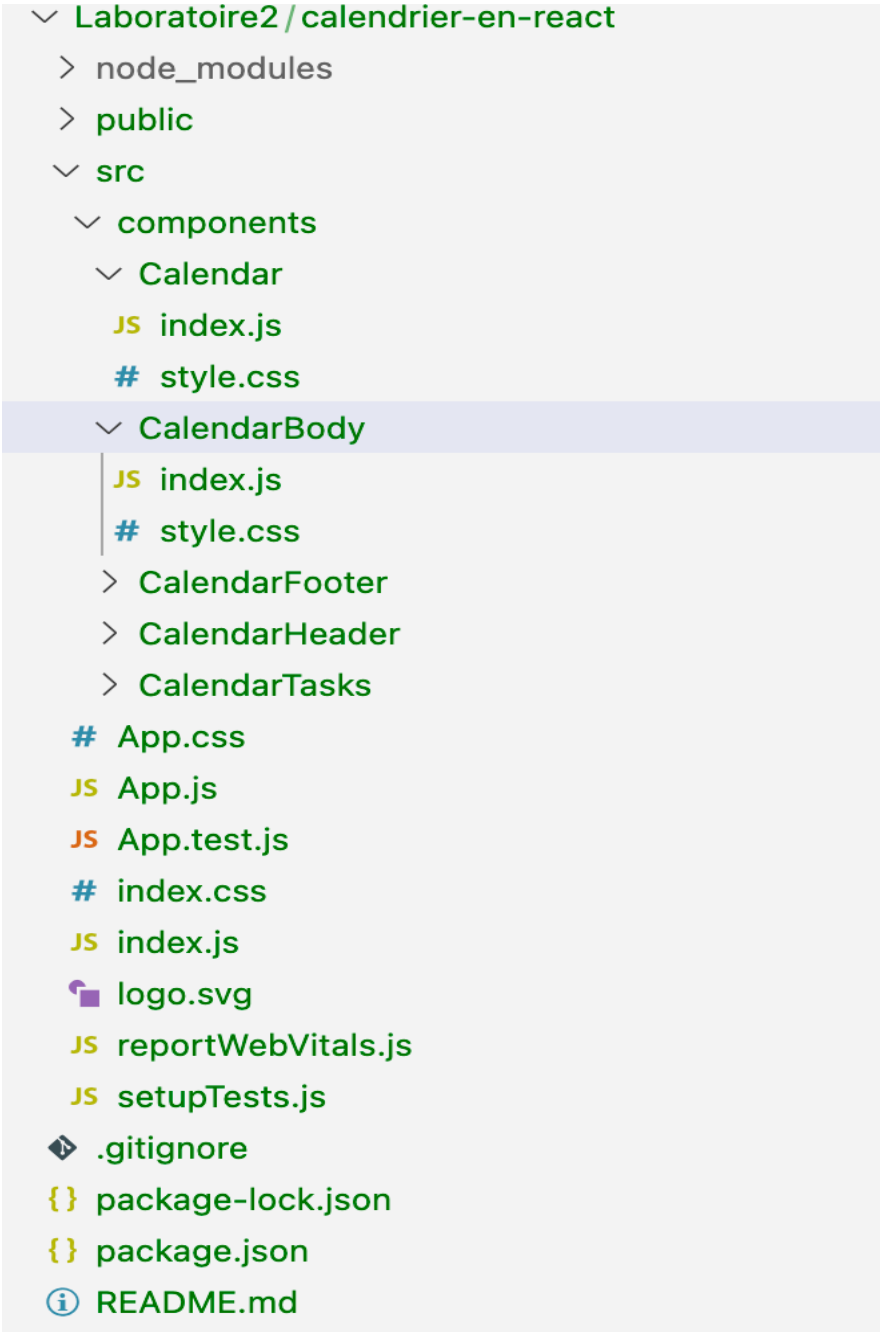
- <Calendar /> : Qui contient tous les autres composants



Structure des composants

- Créer un dossier components/
- Créer un sous-dossier Calendar/ dans le dossier components/
- Ajouter un fichier index.js et style.css dans le dossier Calendar/

- Faire la même chose pour les autres composants



Code pour le composant Calendar

```
import React from "react";
import CalendarBody from "../CalendarBody";
```

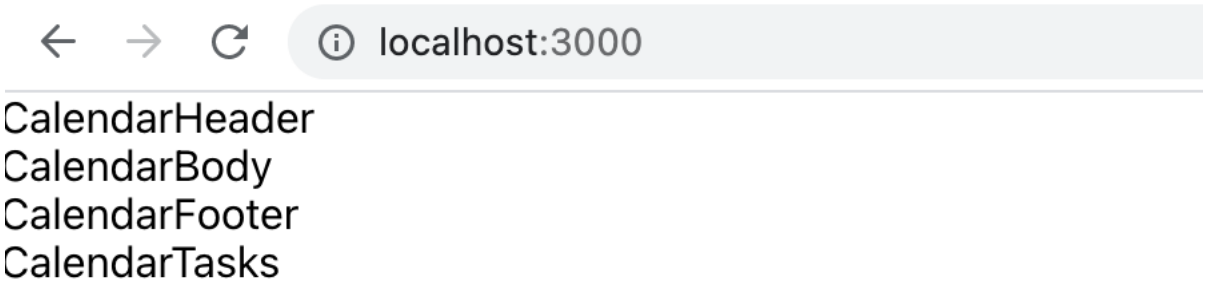
@ Gabriel Y. Gamy

```
import CalendarFooter from "../CalendarFooter";
import CalendarHeader from "../CalendarHeader";
import CalendarTasks from "../CalendarTasks";

import "./style.css";

export default class Calendar extends React.Component {
  state = {};

  render() {
    return (
      <>
        <CalendarHeader />
        <CalendarBody />
        <CalendarFooter />
        <CalendarTasks />
      </>
    );
  }
}
```

Code pour le composant CalendarHeader

```
import React, { Component } from "react";
import "./style.css";

export default class CalendarHeader extends Component {
  render() {
    return <div>CalendarHeader</div>;
  }
}
```

Rendu initial des composants



c- CalendarHeader

Nous pouvons maintenant commencer à construire nos composants en petits morceaux, un composant à la fois. Commençons par le composant CalendarHeader.

- <CalendarHeader /> : Contient l'entête du calendrier, le mois et jour, les icônes « Previous » et « Next », l'entête des jours de la semaine (en commençant par Dimanche).

Rendu initial sans CSS

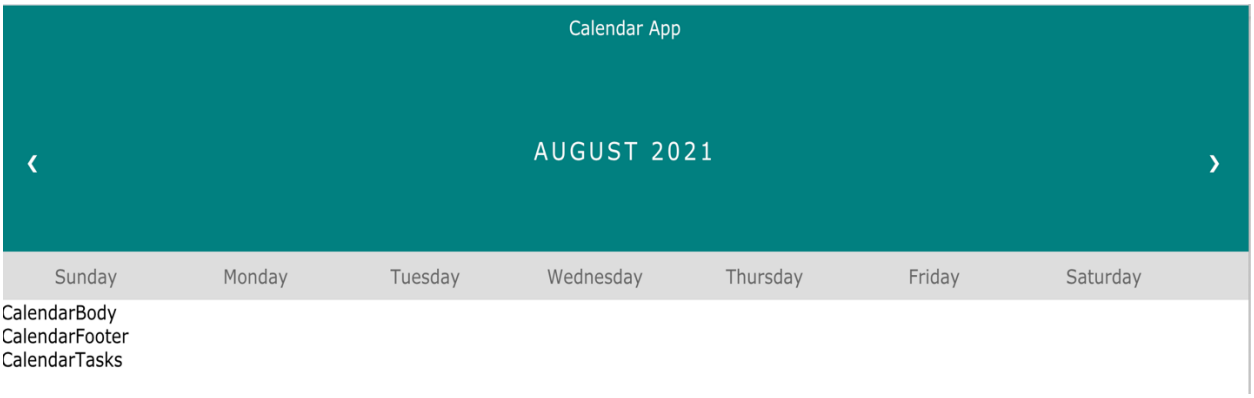
Calendar App

- ❮
- ❯
- August 2021

- Sunday
- Monday
- Tuesday
- Wednesday
- Thursday
- Friday
- Saturday

CalendarBody
CalendarFooter
CalendarTasks

Rendu Final avec CSS



@ Gabriel Y. Gamy

d- CalendarBody

Nous avons notre CalendarHeader. Nous pouvons à présent construire le squelette du CalendarBody.

- <CalendarBody /> : Contient les jours du calendrier pour le mois sélectionné

Pour l'instant, le composant contient uniquement une balise vide.

Code pour le composant CalendarBody

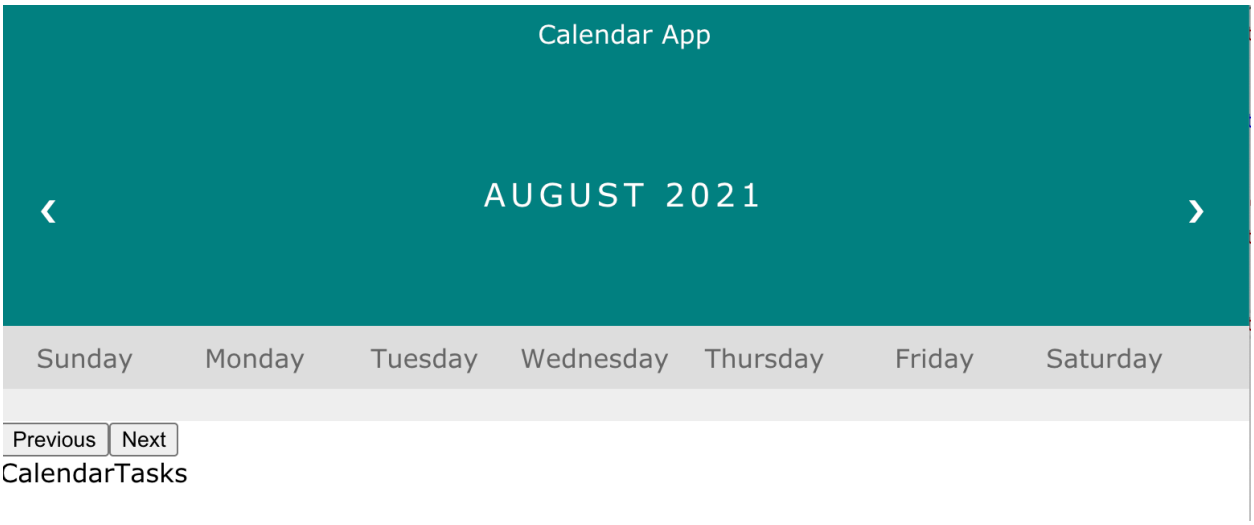
```
import React, { Component } from "react";
import "./style.css";

export default class CalendarBody extends Component {
  render() {
    return <ul id="calendar-days" className="days"></ul>;
  }
}
```

e- CalendarFooter

Les actions « Next » et « Previous » se trouve dans le pied du calendrier.

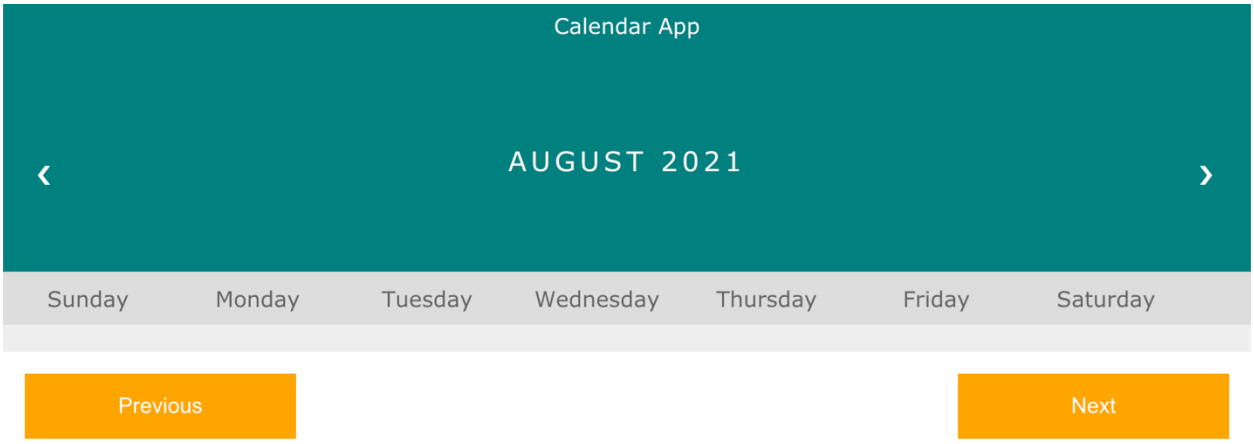
- <CalendarFooter /> : Contient les boutons d'actions « Previous » et « Next »

Rendu initial sans CSS



@ Gabriel Y. Gamy

Rendu Final avec CSS



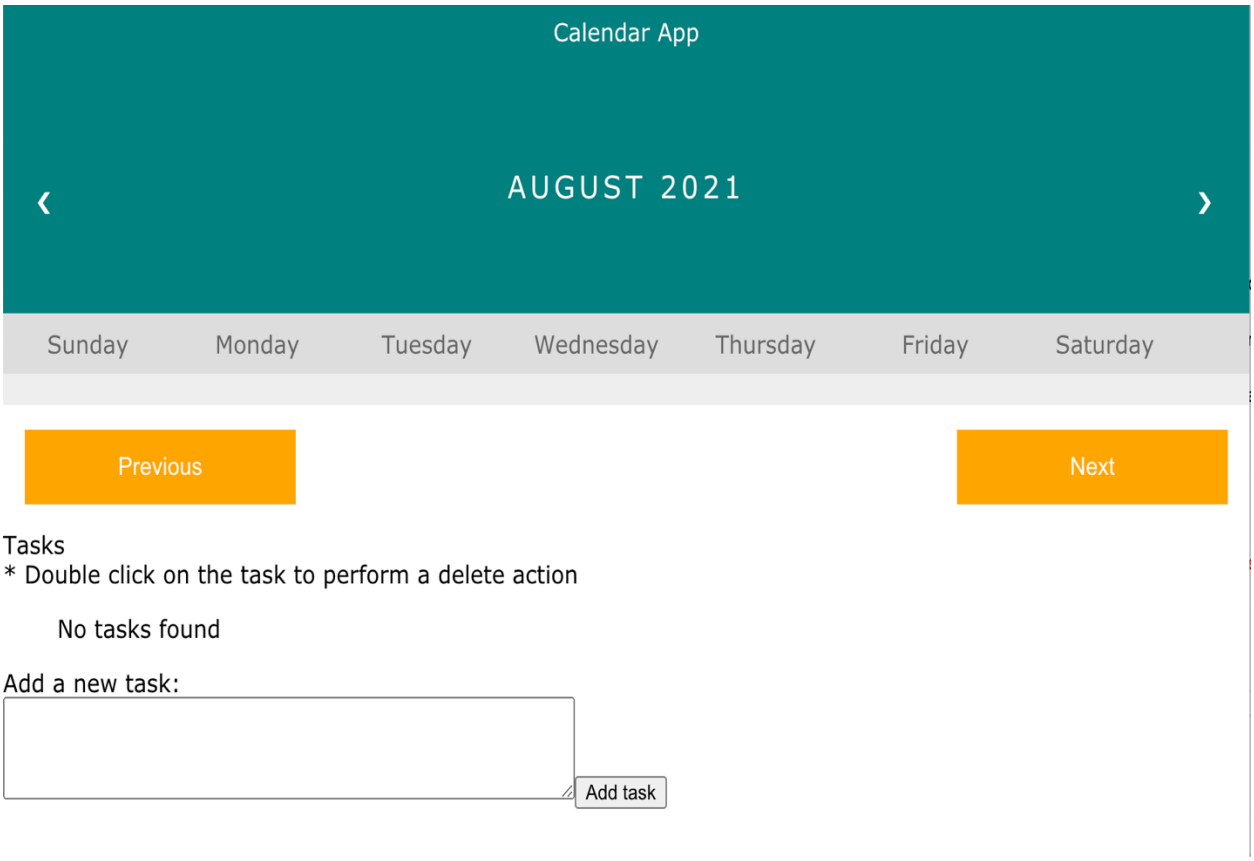
CalendarTasks

f- CalendarTasks

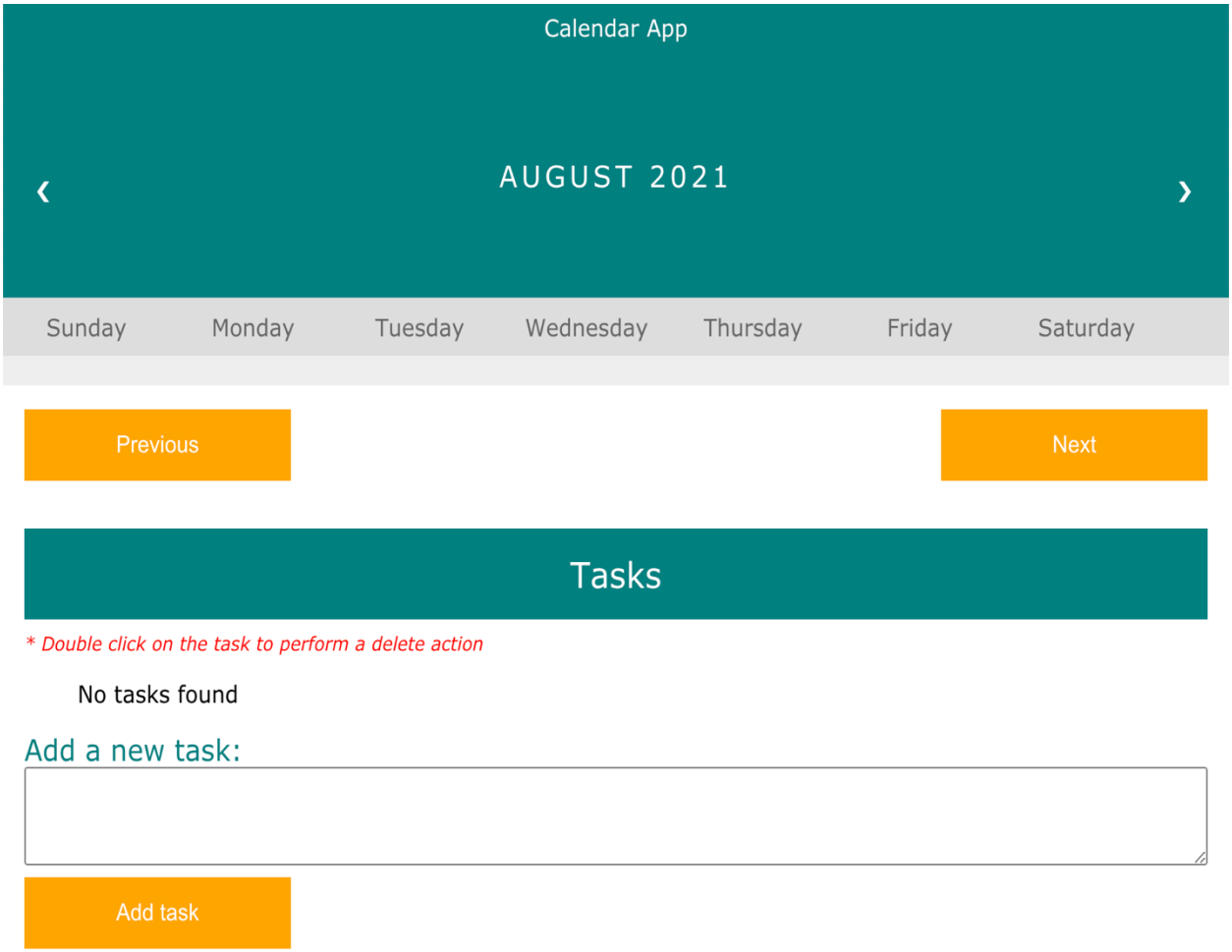
Ajouter la section qui permet de saisir une nouvelle tâche et d'afficher la liste des tâches pour une journée sélectionnée. Uniquement le code JSX et le CSS sont requis pour l'instant. Le comportement JavaScript viendra par la suite.

- <CalendarTasks /> : Contient les tâches associées au jour sélectionné ainsi qu'un formulaire pour saisir une nouvelle tâche.

Rendu initial sans CSS



Rendu Final avec CSS



g- CalendarBody – Afficher les jours du calendrier

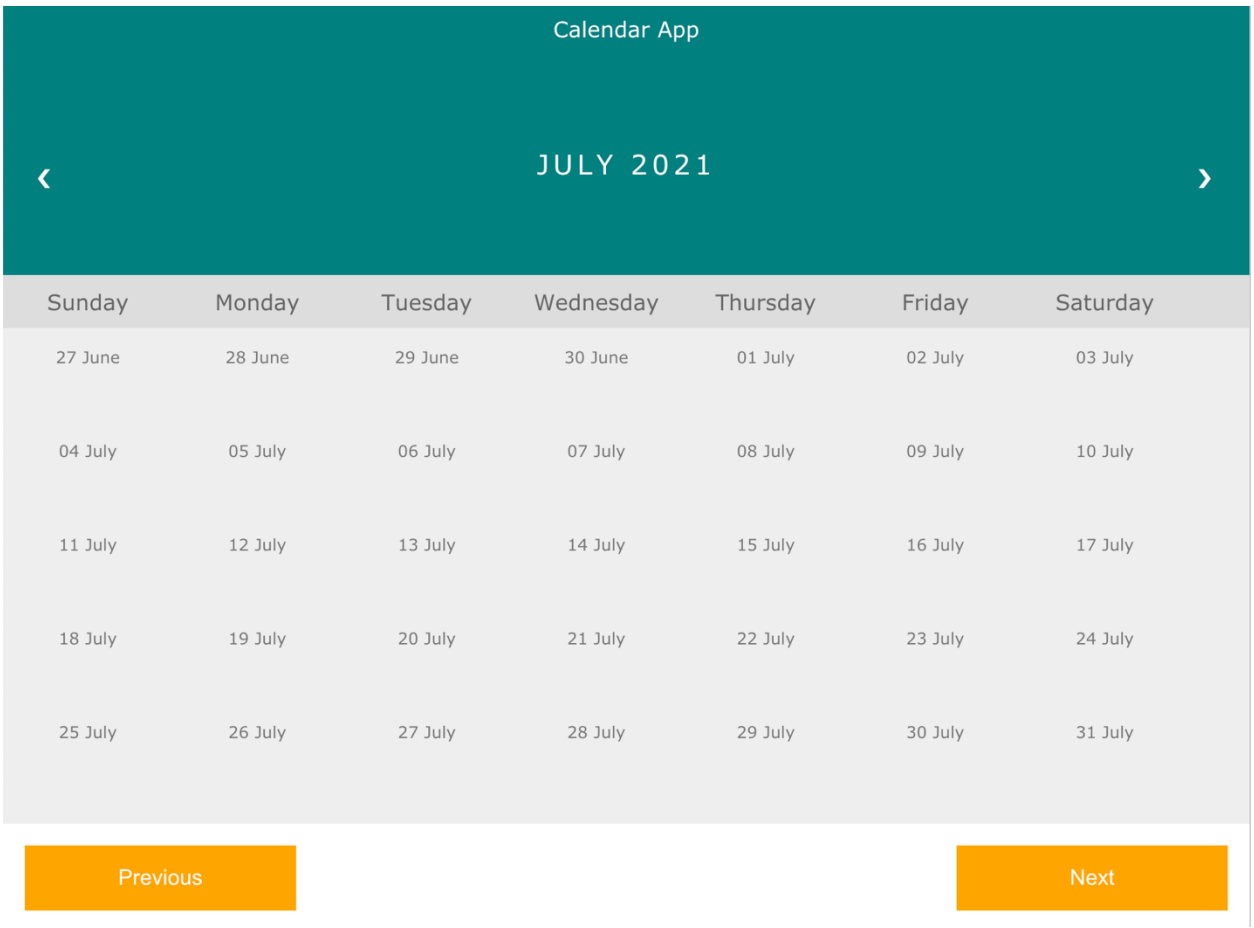
Nous allons maintenant construire le corps du calendrier en y ajouter les jours du mois en cours.

Installer le module moment.js

Utiliser npm pour installer le module moment.js. Ce module nous permet de travailler facilement avec les dates et heures.

npm install moment --save # npm

Afficher la liste des jours du mois actuel



Code Javascript pour la liste des jours

Il faudrait essayer de traduire le code Javascript Calendar.CalendarBuilder en composant React.

Voici le bout de code que nous voulons traduire en React dans le composant CalendarBody.

```
 /**
* calendar-builder.js - Construire le calendrier
*/
var Calendar = Calendar || {};

Calendar.CalendarBuilder = function () {
 /** Index du mois courant de l'année. 0 pour Janvier, 11 pour Decembre */
 const currentMonth = moment().month();
```

@ Gabriel Y. Gamy

```
/** Element HTML <ul id="calendar-days"></ul> */
var calendarDaysElment = document.getElementById("calendar-days");

/** Element HTML <li id="month-text"> */
var monthTextElement = document.getElementById("month-text");

/**
 * Construire le calendrier en fonction du mois.
 * @param { monthToGenerate } Index du mois à générer. Janvier = 0, Decembre = 11
 */
var build = function (monthToGenerate) {
  let monthOffset = monthToGenerate - currentMonth;
  let calendarWeeks = [];

  /** Clone de l'objet moment() afin d'eviter des manupulations sur l'objet de base */
  const _momentCloned = moment().clone().add(monthOffset, "month");
  monthTextElement.innerHTML = _momentCloned.format("MMMM YYYY");

  calendarDaysElment.innerHTML = "";
  calendarDaysElment.setAttribute("data-month", monthToGenerate);

  let startWeek = _momentCloned.startOf("month").week();
  let endWeek = _momentCloned.endOf("month").week();

  // Cas special pour le mois de Decembre. Par exempe startWeek = 49 et endWeek  1
  if (startWeek > endWeek) {
    endWeek = startWeek + 4; // On conserve les 4 prochaines semaines du mois.
  }

  for (var week = startWeek; week <= endWeek; week++) {
    // Obtenir les 7 jours de la semaine courante.
    let days = Array(7)
      .fill(0)
      .map((n, i) =>
        moment().week(week).startOf("week").clone().add(i, "day")
      );

    calendarWeeks.push({ week, days });
  }

  // Construire la balise <li> pour chacun des jours du mois par semaine.
  calendarWeeks.forEach(_renderCalendar);

  // Mettre la date du jour comme active.
  setTodayAsActive();
};
```

@ Gabriel Y. Gamy

```
/**
 * Utilisée dans la boucle d'itération forEach() pour générer
 * les balises <li> pour chacun des jours du mois.
 *
 * @param {week} La semaine en cours d'iteration
 * @param {index} La position de la semaine en cours d'iteration
 * @param {arr} Le tableau des semaines
 */
var _renderCalendar = function (week, index, arr) {
  for (var i = 0; i < week.days.length; i++) {
    var li = document.createElement("li");
    li.setAttribute("class", "day-item");
    calendarDaysElment.appendChild(li);
    li.innerHTML += week.days[i].format("DD MMMM");
    li.addEventListener("click", setActiveDayHandlerCallback);
  }
};

/**
 * Au clic sur un jour du calendrier, mettre ce jour comme le jour actif.
 */
var setActiveDayHandlerCallback = function () {
  const TaskManager = Calendar.CalendarTasks();
  var elems = document.querySelector("li.active");
  if (elems !== null) {
    elems.classList.remove("active");
  }
  this.classList.add("active");
  TaskManager.showTasks(this.textContent);
};

/**
 * Mettre par defaut le jour d'aujourd'hui comme actif
 */
var setTodayAsActive = function () {
  const TaskManager = Calendar.CalendarTasks();
  const today = moment().format("DD MMMM");
  // Transformer HTMLCollection en Array, puis obtenir l'element <li> d'aujourd'hui
  const todayElementList = Array.from(
    document.getElementsByClassName("day-item")
  ).filter((element) => element.textContent === today);

  const todayElement = todayElementList.length ? todayElementList[0] : null;

  if (todayElement != null) {
```

```
    todayElement.classList.add("active");
    TaskManager.showTasks(todayElement.textContent);
   }
  };

  return {
   build,
  };
};
```

h- CalendarHeader & CalendarFooter

Implémenter les actions « Next » et « Previous » dans l'entête et le pied du calendrier. L'information devra être remontée au composant parent afin de pouvoir recalculer les jours du mois sélectionné.

Code JavaScript pour la navigation « Next » et « Previous »

Voici le code équivalent Javascript que l'on souhaite traduire en React.

```
/**
 * calendar-navigation.js - Écouter les évènements sur les boutons Next et Previous
 */
var Calendar = Calendar || {};

Calendar.CalendarNavigation = function () {
  const Builder = Calendar.CalendarBuilder();
  const calendarDaysElment = document.getElementById("calendar-days");
  const prevButtons = document.getElementsByClassName("js-prev");
  const nextButtons = document.getElementsByClassName("js-next");

  var init = function () {
   previousHandlder();
   nextHandler();
  };

  var previousHandlder = function () {
   for (var i = 0; i < prevButtons.length; i++) {
    prevButtons[i].addEventListener("click", function () {
     var month = parseInt(calendarDaysElment.getAttribute("data-month"));
     Builder.build(month - 1);
    });
   }
  };
```

```
  var nextHandler = function () {
    for (var i = 0; i < prevButtons.length; i++) {
      nextButtons[i].addEventListener("click", function () {
        var month = parseInt(calendarDaysElment.getAttribute("data-month"));
        Builder.build(month + 1);
      });
    }
  };

  return {
    initialize: init,
  };
};
```

i- CalendarTasks

Enfin, nous pouvons implémenter la gestion des tâches des jours du calendrier. Le composant doit recevoir en paramètre le jour actif (sélectionné) dans le calendrier.

- Un changement de la journée active entraine un affichage de la liste des tâches de cette journée
- Un double clic sur une tâche entraine la suppression de cette tâche.
- Un maximum de cinq tâches supportées. Après un dépassement, afficher un message d'erreur sous forme d'alerte.
- Pour une journée sans tâches, afficher un message « No tasks found »

```
<CalendarTasks activeDay={this.state.activeDay} />
```

Code JavaScript pour la gestion des tâches associées aux jours du calendrier

```
/**
 * calendar-tasks.js - Gestions des taches d'un jour du calendrier.
 */
var Calendar = Calendar || {};

// Dictionnaire des taches. key => day, value => Array(task)
// { "01 July": ['Learning Javascript', 'HTML test']}
var tasks = {};

Calendar.CalendarTasks = function () {
  const addTaskInput = document.getElementsByClassName(
    "calendar-tasks__save-input"
  )[0];
  const addTaskBtn = document.getElementsByClassName(
```

```
  "calendar-tasks__save-btn"
)[0];

var init = function () {
  addTaskHandler();
};

var addTaskHandler = function () {
  addTaskBtn.addEventListener("click", function () {
    const taskValue = addTaskInput.value.trim();
    const dayActiveElement = document.querySelector("li.active");
    const dayText = dayActiveElement ? dayActiveElement.textContent : "";

    if (!dayText.length) {
      alert("No selected day found!");
      return;
    }

    if (!taskValue.length) {
      alert("Please enter a valid task!");
      return;
    }

    tasks[dayText] = tasks[dayText] || [];

    if (tasks[dayText].length >= 5) {
      alert("You have exceeded the maximum. Please delete some tasks.");
      return;
    }

    tasks[dayText].push(taskValue);

    showTasks(dayText);
  });
};

var deleteTaskHandler = function () {
  let elements = document.getElementsByClassName("calendar-tasks__item");
  for (var i = 0; i < elements.length; i++) {
    elements[i].removeEventListener("dblclick", deleteTaskHandlerCallback);
    elements[i].addEventListener("dblclick", deleteTaskHandlerCallback);
  }
};

var deleteTaskHandlerCallback = function () {
  let day = this.getAttribute("data-day");
```

```
    let index = parseInt(this.getAttribute("data-index"));
    tasks[day].splice(index, 1);
    showTasks(day);
  };

  var showTasks = function (day) {
    tasks[day] = tasks[day] || [];
    const tasksOfDay = tasks[day];
    const tasksElement = document.getElementsByClassName(
      "calendar-tasks__items"
    )[0];

    tasksElement.innerHTML = tasksOfDay.length ? "" : "No tasks found";

    for (var i = 0; i < tasksOfDay.length; i++) {
      var li = document.createElement("li");
      li.setAttribute("class", "calendar-tasks__item");
      li.setAttribute("data-day", day);
      li.setAttribute("data-index", i);
      li.innerHTML = tasksOfDay[i];
      tasksElement.appendChild(li);

      deleteTaskHandler();
    }
  };

  return {
    initialize: init,
    showTasks,
  };
};
```