Lesson 29

План заняття

- Вступ до React;
- Налаштування webpack;
- JSX;
- Вступ до rendering.

React

React це JavaScript-бібліотека для створення інтерфейсів користувача. Вона дозволяє вам збирати складний UI з маленьких ізольованих шматочків коду, які називаються «компонентами».

Переваги React:

- Використовує деклоративний підхід. Декллоративний підхід полягає у зазначенні якого результату ви хочете досягти замість кроків, необхідних для його отримання.
- Заснований на компонентах. Ви створюєте інкапсульовані компоненти з власним станом, а потім поєднуєте їх у складні інтерфейси користувача.
- Дуже гнучкий та дозволяє вам писати як на стороні бєка за допомогою NextJS, робити мобільні застосунки за допомогою React Native.
- Простий у розумінні. Не потребує зайвого вивчення технології.

Чому з’явились фреймворки?

Тому що з кожним роком складність застосунків зростала. Підтримувати код ставало складніше. Потрібна була уніфікація і автоматизація процесів розробки.

Декларативний підхід

У класичному імперативному програмуванні ми взаємодіяли з DOM. Як це виглядало? Щоб створити якусь сутність ми:

1. Йшли до document;
2. Викликали createElement, та вказували що саме ми хочемо створити;
3. Сетили атрибути та контент;
4. Використовували обробники подій;
5. Знову йшли до document та вставляли кудись свій елемент.

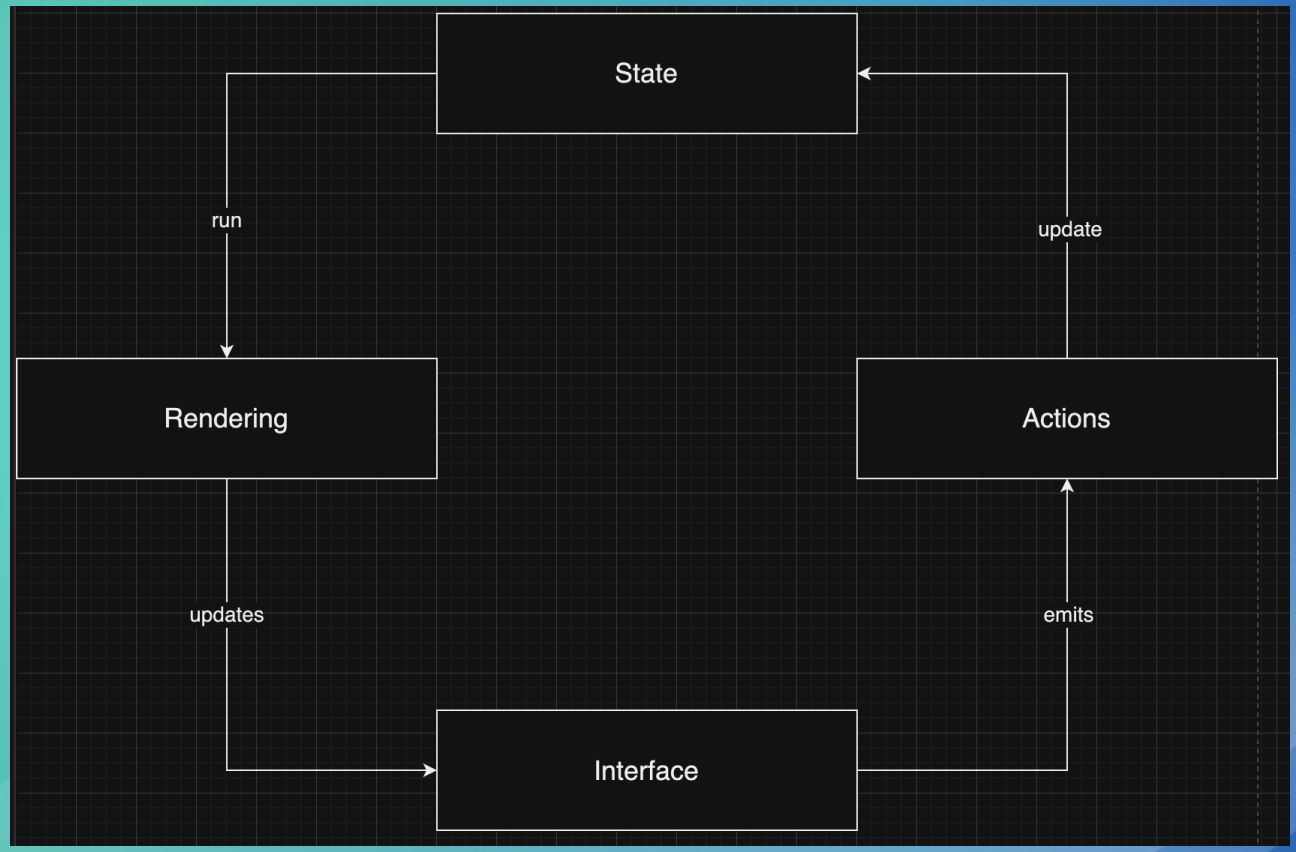
У react концепції нам не потрібно цього робити! Нам слід просто сказати реакту яку сутність ми хочемо зараз отримати, та що їй потрібно мати. Далі react все зробить за нас.

Тобто спілкуватись з DOM потрібно тільки у разі крайній потреби. Наприклад тоді, коли ми викликаємо анімацію, яка не інтегрована до react (наприклад GSAP). Це скоріше виключення.

Концепція інтерактивності

1. Користувач бачить ваш інттерфейс;
2. Робить якусь дію;
3. Дія відновлює стан вашого застосунку;
4. Зміна стану запускає рендорінг;
5. Рендорінг відновляє інтерфейс;

Це іде по колу, доки користувач не завершить роботу з вашим застосунком. Так працює будь яка система чи браузер чи мобільний застосунок.



React

Що він нам надає:

- Рендерінг HTML;
- Обробка та додавання подій до DOM;
- Зберігає стан нашого застосунку і кожного окремого елемента (якщо нам це потрібно);
- Автоматично запускає рендорінг, коли змінюється стан;
- Перевикористання компонентів де завгодно.

Усе інше ми мусимо додати до React за нашою потребою.

React stack

Розширює можливості реакту:

- React-router;
- Redux & react-redux || Recoil...;
- Axios;
- Formic & react-hook-form;
- React-transition-group;
- Що завгодно з https://www.npmjs.com/search?q=react

Конфігуруємо оточення

Частіше за все для створення react проекту використовують webpack. Ви можете розгортати його і за допомогою gulp але це буде складніше.

Існує більш простий метод, за допомогою використання CLI та комадни create-react-app. Він автоматично створить базовий React проект.

Для створення базового проекту запускаємо команду:

npx create-react-app my-app

Докладніше тут https://create-react-app.dev/docs/getting-started/

Щоб зрозуміти що create-react-app використовує як конфігурацію потрібно запустити команду

npm eject - вона розпакує конфіги, та ви зможете їх змінити, але зворотного шляху немає!

Тільки хардКод

Щоб власноруч зконфігорувати webpack ттреба як мінімум додати нові пресети.

npm i –D @babel/preset-react

Він буде розуміти як потрібно парсити синтаксис jsx та потім вже вантажити react та react-dom у проєкт.

npm i react react-dom

Тоді ви зможете використовувати react як нативну бібліолтеку.

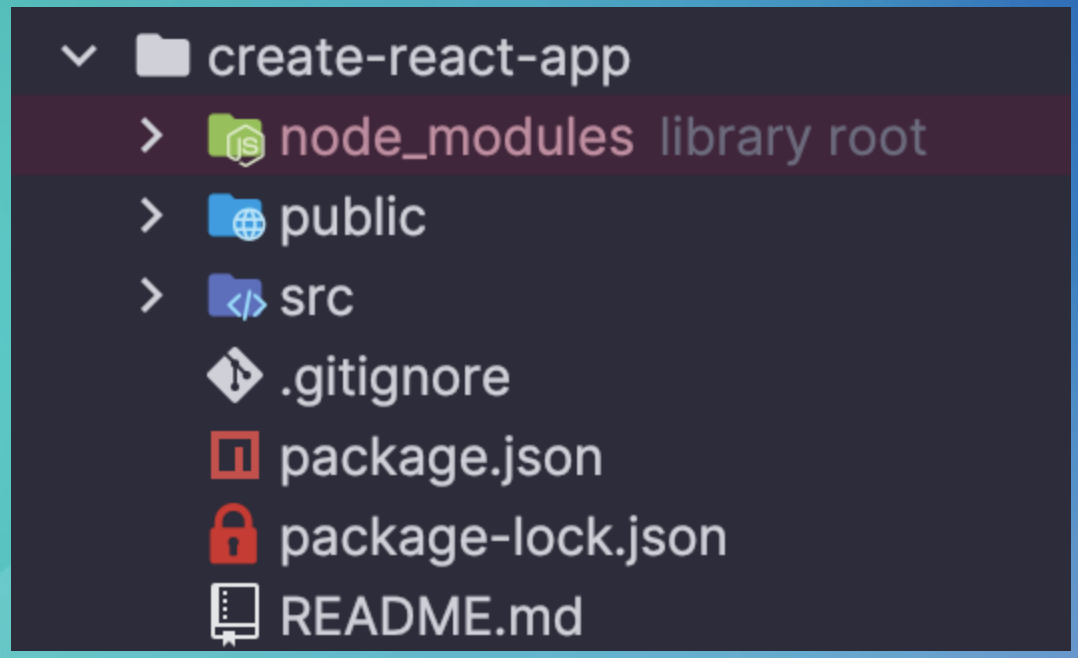
Далі вам порібно все конфігорувати власноруч!

Структура проекту

public - дерікторія де зберігаються статичні файли, які не потребують ніяких змін зі сторони webpack.

src - дерікторія з усіма нашими компонентами, зображеннями та стилями. Там є все що стосується розробки.

У корені проекту будуть зберігатись усі конфігурації проекту.



React.render()

На відміну від DOM-елементів, елементи React – це прості об'єкти, які не забирають багато ресурсів. React DOM оновлює DOM, щоб він відповідав переданим React-елементам.

 - це «кореневий» вузлол DOM, оскільки React DOM керуватиме його вмістом.

Для рендерингу React-елемента спочатку передайте DOM-елемент в ReactDOM.createRoot(), далі передайте React-елемент в root.render()

Елементи React іммутабельні. Після створення елемента не можна змінити його нащадків чи атрибути.

React DOM порівнює елемент та його дочірнє дерево з попередньою версією та вносить до DOM лише мінімально необхідні зміни.

Virtual DOM

Віртуальний DOM (VDOM) — це концепція програмування, де ідеальне або «віртуальне» представлення інтерфейсу користувача зберігається в пам'яті та синхронізується з «реальним» DOM бібліотекою, такою як ReactDOM. Цей процес називається звіркою (reconciliation).

Докладніше про звірку тут https://legacy.reactjs.org/docs/reconciliation.html

Цей підхід увімкне декларативний API React: ви вказуєте React, у якому стані має бути інтерфейс користувача, і він гарантує, що DOM відповідає цьому стану. Це абстрагує маніпуляції атрибутами, обробку подій і ручне оновлення DOM, які вам довелося б використовувати для створення програми.

У світі React термін «віртуальний DOM» зазвичай асоціюється з елементами React, оскільки вони є об'єктами, що представляють інтерфейс користувача. Однак React також використовує внутрішні об'єкти, які називаються «волокнами» (fibers), щоб зберігати додаткову інформацію про дерево компонентів. Їх також можна вважати частиною реалізації «віртуального DOM» у React.

Докладніше про fibers тут https://github.com/acdlite/react-fiber-architecture

Simple example of VDOM

hillel IT school

```
{
  "tag": "h1",
  "attributes": {
    "class": "title",
    "id": 1
  },
  "children": [
    "Hello World",
    {
      "tag": "p",
      "attributes": {
        "class": "description"
      },
      "children": "description"
    }
  ]
}
```

React.createElement()

Ця функція створює елемент React із заданим типом, атрибутами та дочірніми елементами.

Синтаксис:

```
const el = React.createElement(type, props, ...children);
```

Результат виклику буде сходим на:

```
const el = {
  type: 'h1',
  props: {
    className: 'App',
    children: 'Hello world'
  }
};
```

Докладніше тут https://react.dev/reference/react/createElement

React.createElement()

hillel IT school

React.createElement може приймати дочірні елементи для рендеру їх усередині створюваного елемента. Так ми можемо вкладати одні елементи в інші та будувати наш додаток по цеглинках. Це дуже схоже на те, як працюють HTML-теги.

Але описувати розмітку за допомогою React.createElement не дуже зручно – код неочевидний та однотипний.

Щоб вирішити цю проблему було створено JSX.

JSX

JSX – розширення мови JavaScript. JSX виготовляє «елементи» React.

Замість того, щоб штучно розділити технології, поміщаючи розмітку та логіку в різні файли, React поділяє відповідальність за допомогою слабко пов'язаних одиниць, які називаються «компонентами», які містять і розмітку, і логіку.

JSX допускає використання будь-яких коректних JavaScript-виразів усередині фігурних дужок.

Після компіляції кожен JSX-вираз стає звичайним викликом JavaScript-функції, результат якого об'єкт JavaScript.

JSX підтримує усі атрибути DOM улументів. Оскільки JSX ближче до JavaScript, ніж до HTML, React DOM використовує стиль іменування camelCase для властивостей замість звичайних імен HTML-атрибутів.

Якщо тег порожній, то його можна відразу ж закрити за допомогою точно так само, як і в XML.

За замовчуванням React DOM екранує всі значення, включені в JSX, перш ніж відрендерувати їх. Це гарантує, що ви ніколи не впровадите щось, що не було явно написано у вашому додатку. Все перетворюється на рядки, перед тим як бути відрендереним. Це допомагає запобігати атакам міжсайтовим скриптингом (XSS).

JSX

JSX - це синтаксичний цукор для функції React.createElement. Його можна розцінювати як шаблонизатор, у якому доступний звичайний JS.

Він був створений для зручності роботи розробника - набагато простіше працювати з чимось схожим на HTML-теги, ніж зі звичайною JS-функцією.

У результаті кожен JSX-тег за допомогою Babel перетворюється на виклик функції React.createElement.JSX - це синтаксичний цукор для функції React.createElement. Його можна розцінювати як шаблонизатор, у якому доступний звичайний JS.

Він був створений для зручності роботи розробника - набагато простіше працювати з чимось схожим на HTML-теги, ніж зі звичайною JS-функцією.

У результаті кожен JSX-тег за допомогою Babel перетворюється на виклик функції React.createElement.

JSX

Подивитись як Babel перепорює JSX можна тут

Наприклад:

Hello! ;

Буде перетворений до

React.createElement("div", null, "Hello!");

JSX or JS?

hillel IT school

Декілька гарних питань.

Коли мені потрібно використовувати js, а коли jsx для створення реакт компонентів?

Це залежить від того як сконфікурован Webpack. Якщо Webpack шукає jsx в усіх файлах, то можна писати як завгодно.

Чому при компіляції JSX мені видється помилка на React?

Скоріше за все у webpack.config не був ініціалізован глобально React. Тому вам потрібно зробити

import React from 'react'.

React render

hillel IT school

React може ренорити:

1. Рядки та числа; числа будуть наведені до рядкового типу. При цьому JSX видаляє порожні рядки та пробіли на початку та в кінці рядка. Нові рядки, що примикають до тега, будуть видалені. Нові рядки між рядковими літералами стискаються в одну пробіл;
2. JSX копоненти;
3. JS-вирази, загорнуті в {}. Результат виразу буде приведений до рядка;
4. Boolean, null, undefined - React просто проигнорирует их;
5. Також масиви з валідними реакт компонентами.

Не може рендорити пусті об’єкти!

Якщо ви рендорите масиви з даними ви мусите передавати унікальні key щоб реакт правильно будував VDOM та змінював тільки ті данні які змінились. Без зайвого перебудування.

Про key можна почитати тут https://uk.legacy.reactjs.org/docs/lists-and-keys.html

Render DOM element props

У пропси можна передавати:

- рядки та числа; числа будуть наведені до рядкового типу;
- JS-вирази, загорнуті в {}. Результат виразу буде наведено до рядка;
- Boolean, null, undefined - React просто проігнорує їх;
- нічого - тоді значення автоматично виставиться у true;

Пропси можна передавати за допомогою spread operator:

```
const props = { className: "test", id: "heading"};

<h1 {...props}>Hello!</h1>
```

Custom component

Щоб створити свій власний компонет, вам порібно створити функію або клас і назвати його з ВЕЛИКОЇ літери. Так ви підкажете Babel що це не звичайний тєг і він це буде обробляти за іншими алгоритмами.

Bable не буде додавати ваш компонет як строку до tag, а передасть туди посилання на ваш компонент.

Наприклад:

```
function Foo() {
  return <div>Hello</div>
}

<Foo />
```

Дочірні елементи передаються як і для DOM-елементів.

У компоненти можна передавати будь-які пропси.

```
<App some="hello" data={[1, 2, 3]} options={{ test: 1 }}>
   <h1>Hello!</h1>
</App>
```

Дочірні елементи запишуться в пропс “children”, решта пропсів передадуть як є.

Дякую за увагу