REACT

LE BASI

```html
<html lang="en">
  <head>
    <meta charset="UTF-8">
    <meta
      name="viewport"
      content="width=device-width, initial-scale=1.0">
    <title>Document</title>
  </head>
  <body>
    <div class="content">Content</div>
  </body>
</html>
```

```css
body {
  padding: 0;
  margin: 0;
  background: #fff;
}

.content {
  max-width: 900px;
  margin: 0 auto;
}
```

```
const el = document.querySelector('.content');

el.innerHTML = "Contenuto aggiornato";
```

`innerHTML` è altamente sconsigliato per ragioni di **sicurezza**

OBIETTIVI

- modificare il contenuto della pagina
- reagire agli eventi che si verificano (es: click)

REACT

- sintassi facile e **sicura** per scrivere HTML usando javascript
- **reactive**: permette di modificare l'HTML in seguito ad eventi

JSX

- sintassi simile ad HTML
- in realtà è **javascript** “mascherato”

```
<MyComponent textVariable="Contenuto da inserire" />
```

- Componente con l'**iniziale maiuscola**
- attributi `camelCase`
- gli attributi possono contenere anche numeri, booleani, funzioni, array… **{tra parentesi graffe}**
  - *(in HTML gli attributi sono sempre stringhe)*

Un componente è una funzione javascript
che ritorna un altro componente/funzione

```
function MyComponent(props) {
  return (
    <div className="content">
      {props.textVariable}
    </div>
  )
}
```

Gli attributi in react si chiamano **props**

```
const Page = () => (
  <Layout>
    <Header>
      <Logo img="./company.png" />
      <Navigation activePage="home" />
      <Search />
    </Header>
    <Container>
      <MyComponent textVariable="Contenuto principale" />
    </Container>
  </Layout>
);
```

Composition: per ottenere applicazioni complesse, combina diversi componenti in una struttura gerarchica, come in HTML

```
function MyComponent({children}) {
  return (
    <div className="content">
      {children}
    </div>
  )
}
```

Per innestare i componenti,
utilizza la prop **children**

Così:

```
<MyComponent>
  <p>
    Contenuto della variabile <em>children</em>
  </p>
</MyComponent>
```

REACTIVE

Facciamo in modo che l’applicazione si comporti in modo dinamico

Passiamo una funzione come parametro a un componente:

```
<Search
  onSearch={
    async (searchQuery) => {
      const searchResults = await
        fetch('http://api.com/search?' + searchQuery);
      setContent(searchResults);
    }
  }
/>
```

Si può fare perché le funzioni sono **first class objects**

Ora vediamo come quella funzione viene utilizzata all'interno del componente Search, tramite la prop *onSearch*:

```
1 const Search = ({ onSearch }) => {
2   return (
3     <form onSubmit={ () => onSearch(what) }>
4       Cerca:
5       <input type="text" name="what" />
6     </form>
7   )
8 }
```

Attenzione: pseudo-code

Ora vediamo come quella funzione viene utilizzata all'interno del componente Search, tramite la prop *onSearch*:

```
const Search = ({ onSearch }) => {
  return (
    <form onSubmit={ () => onSearch(what) }>
      Cerca:
      <input type="text" name="what" />
    </form>
  )
}
```

Attenzione: pseudo-code

Ora vediamo come quella funzione viene utilizzata all'interno del componente Search, tramite la prop *onSearch*:

```
const Search = ({ onSearch }) => {
  return (
    <form onSubmit={ () => onSearch(what) }>
      Cerca:
      <input type="text" name="what" />
    </form>
  )
}
```

Attenzione: pseudo-code

Un passo indietro: cos'è quel `setContent`?

```
<Search
  onSearch={async (searchQuery) => {
    const searchResults = await
      fetch('http://api.com/search?' + searchQuery);
    setContent(searchResults);
  }}
/>
```

E’ un hook di React, che permette di gestire una variabile di stato.

```
const Page = () => {
  const [content, setContent] = useState("Contenuto inizial
  return (
    <Layout>
      <Search onSearch={async (searchQuery) => {
        const searchResults = await
          fetch('http://api.com/search?' + searchQuery);
        setContent(searchResults);
      }}/>
      <MyComponent textVariable={content} />
    </Layout>
  );
}
```

E’ un hook di React, che permette di gestire una variabile di stato.

```
const Page = () => {
  const [content, setContent] = useState("Contenuto inizial
  return (
    <Layout>
      <Search onSearch={async (searchQuery) => {
        const searchResults = await
          fetch('http://api.com/search?' + searchQuery);
        setContent(searchResults);
      }}/>
      <MyComponent textVariable={content} />
    </Layout>
  );
}
```

E’ un hook di React, che permette di gestire una variabile di stato.

```
const Page = () => {
  const [content, setContent] = useState("Contenuto inizial
  return (
    <Layout>
      <Search onSearch={async (searchQuery) => {
        const searchResults = await
          fetch('http://api.com/search?' + searchQuery);
        setContent(searchResults);
      }}/>
      <MyComponent textVariable={content} />
    </Layout>
  );
}
```

E’ un hook di React, che permette di gestire una variabile di stato.

```
const Page = () => {
  const [content, setContent] = useState("Contenuto inizial
  return (
    <Layout>
      <Search onSearch={async (searchQuery) => {
        const searchResults = await
          fetch('http://api.com/search?' + searchQuery);
        setContent(searchResults);
      }}/>
      <MyComponent textVariable={content} />
    </Layout>
  );
}
```

E’ un hook di React, che permette di gestire una variabile di stato.

```
const Page = () => {
  const [content, setContent] = useState("Contenuto inizial
  return (
    <Layout>
      <Search onSearch={async (searchQuery) => {
        const searchResults = await
          fetch('http://api.com/search?' + searchQuery);
        setContent(searchResults);
      }}/>
      <MyComponent textVariable={content} />
    </Layout>
  );
}
```

Quando cambia lo stato di un compomente, la funzione che lo rappresenta viene eseguita nuovamente (rendering) con il valore di stato aggiornato.

In React, l’interfaccia è una funzione dello stato:

```
UI = f(state)
```

Per modificare l’interfaccia bisogna cambiare il suo stato

APPROFONDISCI

- React.dev

GESTISCI LO STATO

Lo stato è la “memoria interna” di un componente

```
function MyButton() {
  const [count, setCount] = useState(0);
  // ...
  return <div></div>
}
```

```
function MyButton() {
  const [count, setCount] = useState(0);
  // ...
  return <div></div>
}
```

- `useState(initialValue)`: imposta un valore iniziale (zero, nell'esempio)
- `count`: il valore corrente dello stato
- `setCount`: una funzione che serve per modificare il valore di `count`

```
function MyButton() {
  const [count, setCount] = useState(0);
  const handleClick = () => setCount(count + 1);
  return <button onClick={handleClick}>
    Clicked {count} times
  </button>
}
```

Ad ogni click di `button`, il counter viene incrementato e il suo valore viene visualizzato all'interno del pulsante (variabile di stato `{count}`)

Prova ad usare le funzionalità di **stato** sulla documentazione ufficiale di React

STATO CONDIVISO

A volte vogliamo che lo stato di un componente sia utilizzabile anche in un altro componente.

Individua il componente **comune** più vicino

```
<div class="container>
  <div class="column">
    <HandleState /> {/* Modifica lo stato */}
  </div>
  <div class="column">
    <DisplayState /> {/* Visualizza o stato */}
  </div>
</div>
```

Individua il componente **comune** più vicino

```
<div class="container>
  <div class="column">
    <HandleState /> {/* Modifica lo stato */}
  </div>
  <div class="column">
    <DisplayState /> {/* Visualizza o stato */}
  </div>
</div>
```

Trasformiamo il in un Componente

```
<Container>
  <div class="column">
    <HandleState /> {/* Modifica lo stato */}
  </div>
  <div class="column">
    <DisplayState /> {/* Visualizza o stato */}
  </div>
</Container>
```

In quel componente puoi gestire lo stato a cui accedono i componenti figli

```
const Container = () => {
  const [state, setState] = useState('initial');

  return (
    <div class="container>
      <div class="column">
        <HandleState onClick={setState} />
      </div>
      <div class="column">
        <DisplayState value={state} />
      </div>
    </div>
  )
}
```

In quel componente puoi gestire lo stato a cui accedono i componenti figli

```
const Container = () => {
  const [state, setState] = useState('initial');

  return (
    <div class="container>
      <div class="column">
        <HandleState onClick={setState} />
      </div>
      <div class="column">
        <DisplayState value={state} />
      </div>
    </div>
  )
}
```

In quel componente puoi gestire lo stato a cui accedono i componenti figli

```
const Container = () => {
  const [state, setState] = useState('initial');

  return (
    <div class="container>
      <div class="column">
        <HandleState onClick={setState} />
      </div>
      <div class="column">
        <DisplayState value={state} />
      </div>
    </div>
  )
}
```

In quel componente puoi gestire lo stato a cui accedono i componenti figli

```
const Container = () => {
  const [state, setState] = useState('initial');

  return (
    <div class="container>
      <div class="column">
        <HandleState onClick={setState} />
      </div>
      <div class="column">
        <DisplayState value={state} />
      </div>
    </div>
  )
}
```

Prova ad usare lo **stato condiviso** nell’esempio sulla documentazione ufficiale di React

REACT CONTEXT

Si usa quando lo stato di un componente deve essere utilizzato da un altro componente, che si trova molto lontano nell’alberatura

Esempio

```
<ComponentWithState> // questo componente contiene uno stato
  <Nested>
    <Nested>
      <Nested>
        <Nested>
          <TargetComponent /> // ...che deve essere usato qui
        </Nested>
      </Nested>
    </Nested>
  </Nested>
</ComponentWithState>
```

```
function ComponentWithState() {
  const [state, setState] = useState();
  return <Nested state={state}/>
}
```

Usando la tecnica delle props, il componente Nested fa solo da passacarte

```
function Nested({state}) {
  return <TargetComponent state={state}/>
}
```

Context permette di scavalcare i passacarte e rendere disponbile lo stato a qualsiasi livello innestato

```
export const Context = createContext();
function ComponentWithState() {
  const [state, setState] = useState();
  return (
    <Context.Provider value={state}>
      <Nested /> // non fa più da passacarte
    </Context.Provider>
  )
}
```

```
function TargetComponent() {
  const state = useContext(Context);
  // ...utilizza lo stato del componente superiore
  return "..."
}
```

SIDE EFFECTS

Hai notato che per modificare lo stato di un componente abbiamo sempre utilizzato on`Click`?

Questa funzione è un **event handler**, che ci permette di modificare lo stato quando l'utente esegue un'azione.

Ma talvolta lo stato deve essere modificato in base ad altri *eventi*, non generati dall'utente.

Chiameremo questi eventi **side effects**.

Oppure, quando avviene un cambiamento di stato e deve essere invocata una funzione esterna: anche questo è un **side effect**.

Per gestire i **side effects**
utilizzeremo un altro *hook* di React:

```
const Component = () => {

  useEffect(() => {
    // side effect code
  }, [dependencies]);

  return <>...qualcosa</>
}
```

Un utilizzo tipico: visualizzare il contenuto di una pagina

```
const Component = ({ pageName }) => {
  const [pageContent, setPageContent] = useState("");

  useEffect(() => {
    fetch('/api/' + pageName)
      .then(res => res.json())
      .then(setPageContent);
  }, [pageName]);

  return <>{pageContent}</>
}
```

Quando cambia `pageName` viene invocata l'api che va a modificare la variabile `pageContent`.

TIPS

```
<div>
  {isLoggedIn ? (
    <AdminPanel />
  ) : (
    <LoginForm />
  )}
</div>
```

- operatori ternari per il conditional rendering

```
<div>
  {isLoggedIn && <AdminPanel />}
</div>
```

- usa l’operatore logico booleano && al posto di *if()*

```
const listItems = products.map(product =>
  <li key={product.id}>
    {product.title}
  </li>
);

return (
  <ul>{listItems}</ul>
);
```

- usa i metodi degli array

```
function MyButton() {
  const [count, setCount] = useState(0);
  return (
    <button onClick={() => setCount(count + 1)}>
      Clicked {count} times
    </button>
  );
}

export default function MyApp() {
  return (
    <MyButton />
    <MyButton />
  );
}
```

- ogni componente ha il suo stato

```jsx
function MyButton({count, setCount}) {
  return (
    <button onClick={() => setCount(count + 1)}>
      Clicked {count} times
    </button>
  );
}

export default function MyApp() {
  const [count, setCount] = useState(0);
  return (
    <MyButton count={count}, setCount={setCount}/>
    <MyButton count={count}, setCount={setCount}/>
  );
}
```

- stato condiviso tra più componenti

```
function MyButton({count, setCount}) {
  return (
    <button onClick={() => setCount(count + 1)}>
      Clicked {count} times
    </button>
  );
}

export default function MyApp() {
  const [count, setCount] = useState(0);
  return (
    <MyButton count={count}, setCount={setCount}/>
    <MyButton count={count}, setCount={setCount}/>
  );
}
```

- stato condiviso tra più componenti

ERRORI DA EVITARE

```
1 function MyButton() {
2   const [count, setCount] = useState(0);
3   return (
4     <button onClick={() => { count = count + 1 }}> // NO!
5       Clicked {count} times
6     </button>
7   );
8 }
```

- non modificare lo stato direttamente, utilizza sempre la funzione `setState` (in questo esempio: `setCount`)

```
function MyButton({count}) {
  return (
    <button onClick={() => { count = count + 1 }}> // NO!
      Clicked {count} times
    </button>
  );
}
```

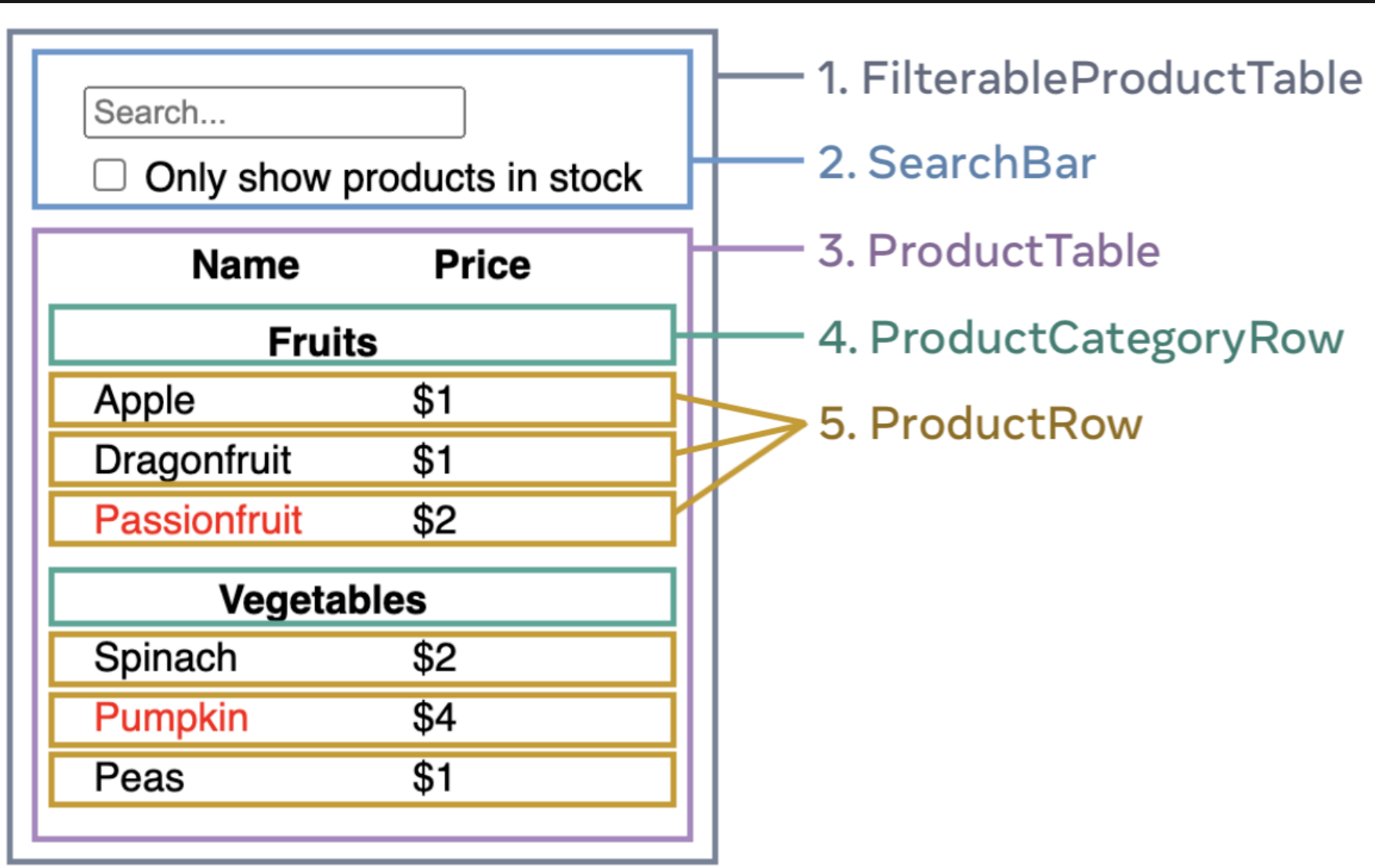
- **rispetta** il flusso di dati, sempre dall'alto verso il basso. Non modificare una proprietà di un componente di livello superiore (in questo esempio: una prop)

```
function MyButton({count, setCount}) {
  return (
    <button onClick={() => {setCount(count + 1)}}>
      Clicked {count} times
    </button>
  );
}
```

Un componente figlio può modificare lo stato di un genitore, se gli viene fornita una funzione per modificarlo (tramite props o context)

THINKING IN REACT

Ovvero: come procedere quando realizzi un’applicazione React


Search...
Only show products in stock
Name
Price
Fruits
Apple
$1
Dragonfruit
$1
Passionfruit
$2
Vegetables
Spinach
$2
Pumpkin
$4
Peas
$1
1. FilterableProductTable
2. SearchBar
3. ProductTable
4. ProductCategoryRow
5. ProductRow

- disegna una **gerarchia di componenti**

- costruisci una versione **statica** in React

- progetta una rappresentazione minima ma completa dello stato
- identifica dove implementare lo stato
  - quali componenti devono gestire lo stato?
  - qual è il primo genitore comune di due componenti che devono condividere lo stesso stato?
  - **lift state up**

- implementa le prop che contengono le funzioni di callback (**inverse data flow**)

APPROFONDISCI

React.dev

INIZIALIZZARE L'APPLICAZIONE

```html
<html>
  <head></head>
  <body>
    <div id="app"></div>
  </body>
</html>
```

```jsx
import { createRoot } from 'react-dom/client';
import { Page } from './Page.jsx'
const root = createRoot(document.getElementById('app'));
root.render(<Page />);
```

LIBRERIE

Utilizzare una libreria di componenti (UI Kit): MUI

```
export const MyCard: React.FC = () => {
  return (
    <Card>
      <CardMedia
        component="img"
        alt="Yosemite National Park"
        image="/static/images/cards/yosemite.jpeg"
      />
      <Stack direction="row" alignItems="center" spacing={3} p
        <Stack direction="column" spacing={0.5} useFlexGap>
          <Typography>Yosemite National Park, California, USA<
          <Stack direction="row" spacing={1} useFlexGap>
            <Chip
              size="small"
              label={active ? 'Active' : 'Inactive'}
```

Utilizzare una libreria per il routing: React Router

```
function App() {
  return (
    <BrowserRouter basename="/">
      <Routes>
        <Route index element={<Home />} />
        <Route path="about" element={<About />} />

        <Route element={<AuthLayout />}>
          <Route path="login" element={<Login />} />
          <Route path="register" element={<Register />} />
        </Route>

        <Route path="concerts">
          <Route index element={<ConcertsHome />} />
          <Route path=":city" element={<City />} />
```