Talk is Cheap, Show me the Code

> **Before starting to build any app, We have to make a plan, make a wireframe So, We can easily design and do code beautiful**

1. **Planning for the UI**

   Before we start coding, plan things out. Planning will make things easier to understand. We should know exactly what to build :

   - Name the App
   - UI Structure

```
+ Header
    - Logo
    - Nav Items
+ Search
+ Restaurant Container
+ Restaurant Card
    - Dish Name
    - Image
    - Restaurant Name
    - Rating
    - Cuisines
    - Time to Deliver

+ Footer
    - Copyright
    - Links
    - Address
    - Contact
```

   - Keep that as a reference and start coding the app

Style :

Inline CSS :

Writing the CSS along with the element in the same file. It is not recommended to use inline styling. So you should avoid writing it.

Inline CSS Example :

We can store the CSS in a variable :

```
const styleCard = {
    backgroundColor : "#f0f0f0"
}
```

and then use it as :

```
<div class="card" style={styleCard}> Card </div>
```

or

Directly write css as :

```
<div class="card" style={{ backgroundColor : "#f0f0f0" }}> Card </div>

**
> First bracket is to tell that whatever is coming next will be JavaScript

> The second bracket is for JavaScript object
**
```

External CSS :

- Inside the `style.css`

CSS Frameworks :

- BootStrap
- React BootStrap
- Tailwind CSS

CSS Modules

Introducing Props

- Short form for properties.
- To dynamically send data to a component we can use `props`.
- Passing a prop to a component is just like passing an argument to a function.
- Basically, React is wrapping these inside a object and it is passing to that function.
- Prop is just a javaScript object.

1. **Passing Props to a Component**

   Example :

   ```
   <RestaurantCard
   resName="Meghana Foods"
   cuisine="Biryani, North Indian"
   />
   ```

> `resName` and `cuisine` are a props and this prop is passing to a component.

2. **Receiving props in the Component**

   Props will be wrapped and send in Javascript object

   Example :

```
const RestaurantCard = (props) => {
    return(
        <div>{props.resName}</div>
    )
}
```

3. **Destructuring Props**

   Example :

   This is known as Destructuring on the fly. This destructuring is Javascript (ES6).

```
const RestaurantCard = ({resName, cuisine}) => {
    return(
        <div>{resName}</div>
    )
}
```

   or

   we can write like this :

```
const RestaurantCard = (props) => {

    const {resName, cuisine} = props

    return(
        <div>{resName}</div>
    )
}
```

4. **How will this restaurant data come from backend into us?**
   - It will come in form of a `json`.

Config Driven UI

1. What is config?
   - Config is the data coming from APi.

    - Basically, how our UI looks like is totally depend on using a config

2. where that config comes from ?

    - that config comes from backend.
    - your UI is basically driven by a Config

- It is a user Interface that is built and configured using a declaration configuration file or data structure, rather than being hardcoded.
- Config is the data coming from the api which keeps on changing according to different factors like user, location, etc.

3. **To add something in the elements of array**

   Example :

   Adding “,”(comma) after every value in an Array, use `join()`

```
resData.data.cuisine.join(", ")
```

Good Practices of Destructuring

Two way for doing destructuring :

1. We can use component key attributes name in the place of props into child components using curly braces.

example :

```
const Destructuring = ({name, avgRating, cuisine}) => {
    return (
        <div>Hello got this {name} through Destructuring.</div>
    )
}
```

2. We can extract from props objects. After that we can just use the var name which we have given for extracting. Get the desired UI.

Example :

```
const {name, avgRating, cuisine} = resData?.data;
```

This `?` is called as Optional Chaining.

- **What is Optional Chaining?**

- The optional chaining `?.` is an error-proof way to access nested ob+ ject properties, even if an intermediate property doesn't exist. It+ works similar to Chaining `.` except that it does not report the error, instead it returns a value which is undefined. It also works with function call when we try to make a call to a method which may not exist.
- When we want to check a value of the property which is deep inside a tree-like structure, we often have to perform check whether intermediate nodes exist.

```
let Value = user.dog && user.dog.name;
```

- The Optional Chaining Operator allows a developer to handle many of those cases without repeating themselves and/or assigning intermediate results in temporary variables:

```
let Value = user.dog?.name;
```

Syntax:

```
obj?.prop
obj?.[expr]
arr?.[index]
func?.(args)
```

example : Optional Chaining with Object

```
const user = {
    dog: {
        name: "Alex"
    }
};

console.log(user.cat?.name); //undefined
console.log(user.dog?.name); //Alex
console.log(user.cat.name);
```

example : Optional Chaining with Function Call

```
let user1 = () => console.log("Alex");

let user2 = {
    dog(){
        console.log("I am Alex");
    }
```

```
  }

  let user3 = {};

  user1?.();   // Alex
  user2.dog?.(); // I am Alex
  user3.dog();     // ERROR - Uncaught TypeError:
                  // user3.dog is not a function.
  user3.dog?.(); // Will not generate any error.
```

Map() method

Repeating ourselves (repeating a piece of code again and again)-

Dynamic Component `listing` using JS `map()` function to loop over an array and pass the data to component once instead of hard coding the same component with different props values

Avoid ❌

```
<RestaurantCard resName="Meghana Foods" />
<RestaurantCard resName="KFC" />
<RestaurantCard resName="McDonald's" />
<RestaurantCard resName="Dominos" />
```

Follow ☑

```
const resList = [
    {
        resName: "Meghana Foods"
    },
    {
        resName: "KFC"
    },
    {
        resName: "McDonald's"
    },
    {
        resName: "Dominos"
    }
]

const Body = () => {
return(
        <div>

        {resList.map((restaurant) => (
```

```
            <RestaurantCard resData={restaurant} />

            )
        )}
        </div>
    )
}
```

Unique Key id while using map-

Each item in the list must be uniquely identified

`Key prop:` it is a special attribute you need to include when creating lists of elements.

- It is not accessible in the child component. So always put in the parent component.
- Key give the elements a stable identity
- Keys help React identities which items have changed, are added or are removed.
- Help in efficient updating of the user interface.

Why?

When we have components at same level and if a new component comes on the first without ID, DOM is going to re-render all the components again. As DOM can't identify where to place it.

But if we give each of them a unique ID then react knows where to put that component according to the ID. It is a good optimization and performance thing.

> Note* Never use index as keys in map. It is not recommended by Official React website. but remember `index` is passed as a second parameter to the arrow function within the `map()` method and that index is used as a value to the key prop.

```
const Body = () => {
return(
    <div>
        {resList.map((restaurant) => (
            <RestaurantCard key={restaurant.id} resData={restaurant} />
        ))}
    </div>
)
}
```

Bad example below :

```
const Body = () => {
return(
    <div>
        {resList.map((restaurant, index) => (
            <RestaurantCard key={index} resData={restaurant} />
        ))}
```

```
    </div>
    )
}
```

React Fragments

- Fragments lets you group a list of children elements without adding extra nodes to the DOM
- We have to enclose multiple elements in a single parent element. So fragments come here instead of “Div” tag. We can replace the enclosing div tag with react fragment and that will prevent the extra node from being added to the DOM.
    - Wrapping elements ex: `<React.Fragment> </React.Fragment>`, also we can pass key attributes.
- We can also use “empty opening tag and empty closing tag”. Ex: `<> </>` but there is only one problem: you can’t pass key attributes in it.

React JS Virtual DOM

1. **What is DOM?**
    - DOM stands for Document Object Model
    - DOM is a structured representation of HTML Elements that are present in a webpage.
    - DOM represents entire UI of application.
    - DOM contains a node for each UI element present in the web document.
2. **Disadvantages of real DOM**
    - Every time DOM gets update then, the updated element & it's children have to be rendered again to update the UI of the page.
    - Even if there is a small update then it starts recalculating the CSS and changing the layouts involves complex algorithm & they do affect the performance.
    - So, react has different approach to deal with it, as it makes use of something known as `Virtual DOM`
3. **What is Virtual DOM?**
    - In React.js, the Virtual DOM is a lightweight representation of the actual Document Object Model (DOM) of a web page.
    - Instead of directly manipulating the real DOM, whenever there are changes in the application state, React first updates the Virtual DOM, which is a copy of the real DOM kept in memory.
4. **How Virtual DOM Works?**

    The virtual DOM works in three simple steps:

    - When a component’s state or props change, React creates a new virtual DOM representation of the component.
    - React then compares this new virtual DOM with the previous version to determine the differences. This process is known as `Reconciliation`.

- Once the differences are identified, React updates the real DOM only with the necessary changes. This minimizes the number of actual DOM manipulations required, making the process more efficient and improving performance.

5. **What is Reconciliation?**

   - The entire process of transforming changes to the real DOM is called Reconciliation.

6. **What is React Fibre?**

   - React Fiber, the reimplementation of the React reconciliation algorithm, is designed to improve this process by introducing a more efficient and flexible approach to handling updates, with the goal of enhancing performance and user experience in React applications.