

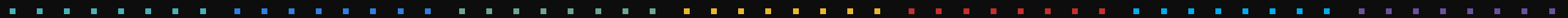
WMNWA 2210

WEB APP DEVELOPMENT

React 2 | Components



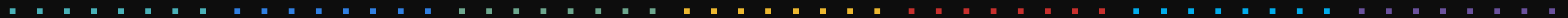
ΠΕΡΙΕΧΟΜΕΝΑ



Περιεχόμενα

- Components
- JSX
 - Fragment
- Props
- Rendering
- State

COMPONENTS



Components

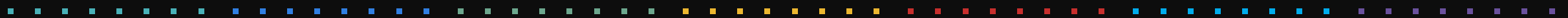
Composition & Re-usability

Η **React** βασίζεται στη φιλοσοφία της σύνθεσης (*composition*) και όχι σε αυτή της κληρονομικότητας (*inheritance*).

Τα **Components** επιτρέπουν το διαχωρισμό του *UI* σε μικρά, ανεξάρτητα, επαναχρησιμοποιούμενα κομμάτια κώδικα.

Παρέχουν ένα βαθμό ενθυλάκωσης (*encapsulation*), όπως και τα *Web Components*, αν και δεν είναι αυτός ο βασικός σκοπός της **React**.

Βασικός σκοπός της **React** είναι να αποτυπώνει τα δεδομένα μιας εφαρμογής στο **DOM**.



Components

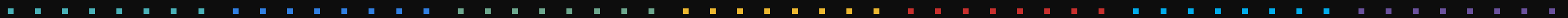
Virtual DOM

Το **Virtual DOM** είναι ένα μια ιδεατή αναπαράσταση του **UI**, στη μνήμη, η οποία συγχρονίζει με το πραγματικό **DOM** μέσω μιας βιβλιοθήκης όπως η **ReactDOM**.

Η παραπάνω διαδικασία ονομάζεται *reconciliation*.

Η προσέγγιση αυτή επιτρέπει την *declarative* φύση της **React**. Δηλώνεις την κατάσταση στην οποία θες να βρίσκεται το **UI** και η βιβλιοθήκη αναλαμβάνει να το κάνει.

Αντί να αναφέρεται απευθείας στο **DOM**, δημιουργεί μια αφηρημένη έκδοχό του.



Components

Function vs Class Components

Ο ενδεδειγμένος τρόπος για να δηλωθεί ένα **Component** στη **React**, από την έκδοση **16.8** και έπειτα, είναι μέσω μιας **JS** συνάρτησης (*function component*).

Πριν την έκδοση αυτή, για τον ίδιο σκοπό, χρησιμοποιούνταν κλάσεις (*class component*).

Τα *function component* υπήρχαν και νωρίτερα από την έκδοση **16.8**, αλλά περιορίζονταν σε απλές χρήσεις, καθώς δεν μπορούσαν να διαχειριστούν *internal state* κ.λπ.

Αυτό άλλαξε με την εισαγωγή των *React hooks*.

Components

Κανόνες

Ένα *function component* πρέπει:

- Να επιστρέφει ένα *React element*, είτε μέσω της *createElement*, είτε με χρήση της **JSX**.
 - Αν δεν υπάρχει ένα μοναδικό *root element* αλλά περισσότερα, μπορεί να γίνει χρήση του **<React.Fragment>**.
- Να έχει όνομα που ξεκινά με κεφαλαίο γράμμα.

Components

Παράδειγμα

```
const styles = { color: '#61DAFB', backgroundColor: 'black' };

function Header() {
  return React.createElement(
    'h1',
    {
      id: 'main-header',
      style: styles
    },
    'Hello, from JSX! Time is ',
    new Date().toLocaleTimeString()
  );
}

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(React.createElement(Header));
```

Components

JSX

Είναι μια **XML** επέκταση της **ECMAScript** που επιτρέπει να συνδυάζουμε **markup** με κώδικα **JavaScript** με έναν ευπαρουσίαστο τρόπο.

```
const element = <h1>Hello, world!</h1>;
```

Η **JSX** θυμίζει αλλά δεν είναι *template engine*.

Ως επέκταση:

- δεν μπορεί να εκτελεστεί απευθείας από το *browser*, χρειάζεται είτε τη βοήθεια μιας βιβλιοθήκης (**Babel**), είτε *transpilation*.
- μπορεί να εμπεριέχει οποιονδήποτε κώδικα **JavaScript**.

Components

Παράδειγμα

```
const styles = { color: '#61DAFB', backgroundColor: 'black' };

const Header = () => (
  <h1 id="main-header" style={styles}>
    Hello, from JSX! Time is {new Date().toLocaleTimeString()}
  </h1>
);

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(<Header />);
```

Components

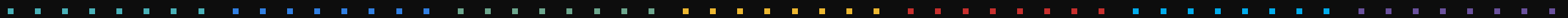
Props

Εννοιολογικά, τα **Components** είναι σαν οποιαδήποτε άλλη συνάρτηση σε **JavaScript**.

Δέχονται είσοδο (**props**) και επιστρέφουν **React element**, τα οποία περιγράφουν τι πρέπει να αποτυπωθεί στο **UI**.

Όλα τα *attributes* που υπάρχουν στη **JSX** περνούν, μέσω του αντικειμένου **props**, στο αντίστοιχο **Component**.

Επιπλέον, αν ένα **Component** περικλείει άλλα **Component** ή **HTML tag**, αυτά είναι διαθέσιμα μέσω του **props.children**.



Components

Props & Curly Brackets

Αν ένα **attribute** παίρνει τιμή "δυναμικά", τότε πρέπει να χρησιμοποιηθούν τα **{ }** αντί των **" "** ή **' '**.

Το ίδιο ισχύει και όταν θέλουμε να περάσουμε μια τιμή που δεν είναι αλφαριθμητική (**string**).

```
const title = 'This is a title';  
  
const component = <ProductList title={title} noOfItems={10} />
```

Components

Παράδειγμα #1

```
const cssAttrs = { color: '#61DAFB', backgroundColor: 'black' };

const Header = (props) => (
  <h1 id="main-header" style={props.style}>
    Hello, from {props.greeter}!
    Time is {new Date().toLocaleTimeString()}
  </h1>
);

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(<Header greeter="JSX" style={cssAttrs} />);
```

Παράδειγμα #2

Components

```
const movies = { title: 'Dune', rating: 8.4, sum: 117 };

const Header = () => (
  <h1>Box Office</h1>
);

const Movie= ({ movie }) => (
  <div>`${movie.title} | ${movie.sum}m | ${movie.rating}★`</div>
);

const App = () => (
  <>
    <Header />
    <Movie movie={movie} />
  </>
);

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(<App />);
```

Παράδειγμα #3

Components

```
const CodeFormatter = ({ children }) => {
  const style = {
    backgroundColor: "lightgrey",
    padding: "1em",
  };
  return (<pre style={style}>{children}</pre>);
};

const App = () => (
  <CodeFormatter>
    let a = 1;
  </CodeFormatter>
);

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(<App />);
```


Components

Loops & Conditionals

Η **React** ακολουθεί τη φιλοσοφία του *Functional Programming*.

Σε αυτή επικρατεί η *declarative* προσέγγιση, έναντι της *imperative*. Μέσα στα *curly brackets* της **JSX** ενσωματώνουμε *expression* και όχι *statement*.

Για το λόγο αυτό, σε κώδικα **React** θα δούμε να χρησιμοποιείται συχνά η μέθοδος **map**, στη θέση εντολών όπως οι **for**, **while**, κ.λπ.

Επίσης, η *conditional* λογική αναπαριστάται, όχι με τις εντολές **if** ή **switch**, αλλά με τους *ternary*, *logical* και *nullish coalescing* τελεστές.

Αυτά αφορούν κυρίως στη **JSX**.

Components

Παράδειγμα

```
const movies = [{ title: 'Dune', rating: 8.4, sum: 117 }, ...];

const Header = () => (<h1>Box Office</h1>);

// Loop items with `.map()` method
// Conditional logic with ternary operator (? :)
const MovieList = ({ movies }) => (
  <ul>
    {movies.map((m) => (
      <li>`${m.title} ${m.sum ? `| ${m.sum}m` : ''} | ${m.rating}★`</li>
    ))}
  </ul>
);

const App = () => (
  <>
    <Header />
    <MovieList movies={movies} />
  </>
);

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(<App />);
```

Components

Prop destructuring

Τα **function components**, όπως προαναφέρθηκε, δέχονται το πολύ μία παράμετρο.

Η παράμετρος αυτή, συνήθως, έχει το όνομα **props**, χωρίς αυτό να είναι δεσμευτικό.

Για να αποφευχθεί η συνεχής αναφορά στην παράμετρο αυτή, πολύ συχνά χρησιμοποιείται η τεχνική του *destructuring*.

```
const Header = (props) => (<h1>{props.title}</h1>);  
const Header = ({ title }) => (<h1>{title}</h1>);
```



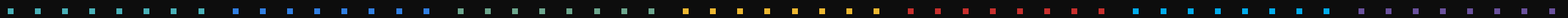
Components

Stateful Components

Στο παρελθόν, τα **Function Components** χρησιμοποιούνταν για απλά στοιχεία, που δεν είχαν αποθήκευαν εσωτερικά την κατάσταση στην οποία βρίσκονταν (*stateless*).

Με την εμφάνιση των **React Hooks**, αυτό άλλαξε και έκανε τη χρήση των **class component** μη αναγκαία.

Για να αναπαρασταθεί η εσωτερική κατάσταση ενός **Component**, γίνεται χρήση της μεθόδου **useState**.



Components

useState hook

Η μέθοδος **useState** δέχεται ως όρισμα μια αρχική τιμή και επιστρέφει έναν πίνακα (θυμίζει **tuple**).

Ο πίνακας περιέχει την τρέχουσα τιμή ως πρώτο στοιχείο και μια συνάρτηση που αλλάζει την τιμή αυτή, ως δεύτερο.

```
const [state, setState] = useState(initialState);
```

Και εδώ, η τεχνική του *destructuring* είναι πολύ συνηθισμένη.

Components

useState hook

- Το πρώτο στοιχείο (ίδιου τύπου με την αρχική τιμή), θα χρησιμοποιηθεί μέσα στην **JSX** για να αναπαραστήσει τη συγκεκριμένη τιμή.
- Το δεύτερο στοιχείο (συνάρτηση), θα χρησιμοποιηθεί για την αλλαγή της τιμής, καθώς "απαγορεύεται" (*immutability*) να επέμβουμε απευθείας σε αυτή.
 - Αν δοθεί, ως όρισμα, μια τιμή, αυτή η τιμή θα αντικαταστήσει την παλιά.
 - Αν η νέα τιμή εξαρτάται από την προηγούμενη, τότε πρέπει να δοθεί, ως όρισμα, μια συνάρτηση που να περιγράφει την αλλαγή, βάσει της προηγούμενης τιμής.

Components

Παράδειγμα #1

```
const Counter = ({initial}) => {  
  const [count, setCount] = useState(initial);  
  return (  
    <>  
      Count: {count}  
      <button onClick={() => setCount(initial)}>Reset</button>  
      <button onClick={() => setCount(prev => prev - 1)}>-</button>  
      <button onClick={() => setCount(prev => prev + 1)}>+</button>  
    </>  
  );  
}
```

Παράδειγμα #2

Components

```
const ClickCounter = () => {
  const [count, setCount] = React.useState(0);

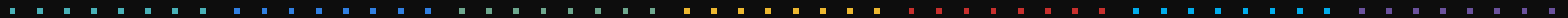
  const incrementCounter = () => { setCount((prev) => prev + 1); };
  const resetCounter = () => { setCount(0); };

  return (
    <div>
      <button onClick={incrementCounter}>Click me!</button>
      <button onClick={resetCounter}>Reset</button>
      <span>Clicks: {count}</span>
    </div>
  );
};

const App = () => (
  <ClickCounter />
);

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(<App />);
```


HOMework



Homework

Accordion menu

Να φτιαχτεί ένα **React Component** που να υλοποιεί ένα *accordion menu*.

Το μενού θα αποτελείται από δυάδες **h2** & **div**, οι οποίες μπορεί να είναι αποθηκευμένες σε έναν πίνακα.

Όταν ο χρήστης κάνει κλικ σε ένα **h2** εμφανίζεται το αντίστοιχο **div**, ενώ κρύβονται όλα τα υπόλοιπα. Αρχικά, στο μενού, όλα τα **div** είναι κρυμμένα.

Στο **state** θα πρέπει να αποθηκεύεται το **index** του στοιχείου που είναι ορατό ή *-1* αν είναι όλα κρυμμένα.

Starting point

Χρήσιμα links

 Components and Props – React
<https://reactjs.org/docs/components-and-props.html>

 State and Lifecycle – React
<https://reactjs.org/docs/state-and-lifecycle.html>


 Handling Events – React
<https://reactjs.org/docs/handling-events.html>

 React Function Components
<https://www.robinwieruch.de/react-function-compon...>

Extra info


 A Beginner's Guide to Loops in React JSX
<https://www.telerik.com/blogs/beginners-guide-loops...>

 Conditional Rendering – React
<https://reactjs.org/docs/conditional-rendering.html>

 Conditional Rendering
<https://beta.reactjs.org/learn/conditional-rendering>

 Conditional (ternary) operator - JavaScript | MDN
<https://developer.mozilla.org/en-US/docs/Web/JavaS...>

 Nullish coalescing operator (??) - JavaScript | MDN
<https://developer.mozilla.org/en-US/docs/Web/JavaS...>

 Destructuring assignment - JavaScript | MDN
<https://developer.mozilla.org/en-US/docs/Web/JavaS...>

 Passing Functions to Components – React
<https://reactjs.org/docs/faq-functions.html>

 A Beginner's Guide to Loops in React JSX
<https://www.telerik.com/blogs/beginners-guide-loops...>

THANK YOU!

