

PLAIN JAVASCRIPT VS. REACT

Sijad Hashemian

hi@hashemian.me

Sina Maleki

jsina.maleki@gmail.com

Ali Piry

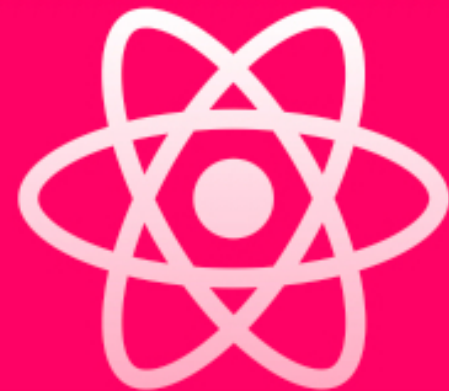
alipiry@disroot.org



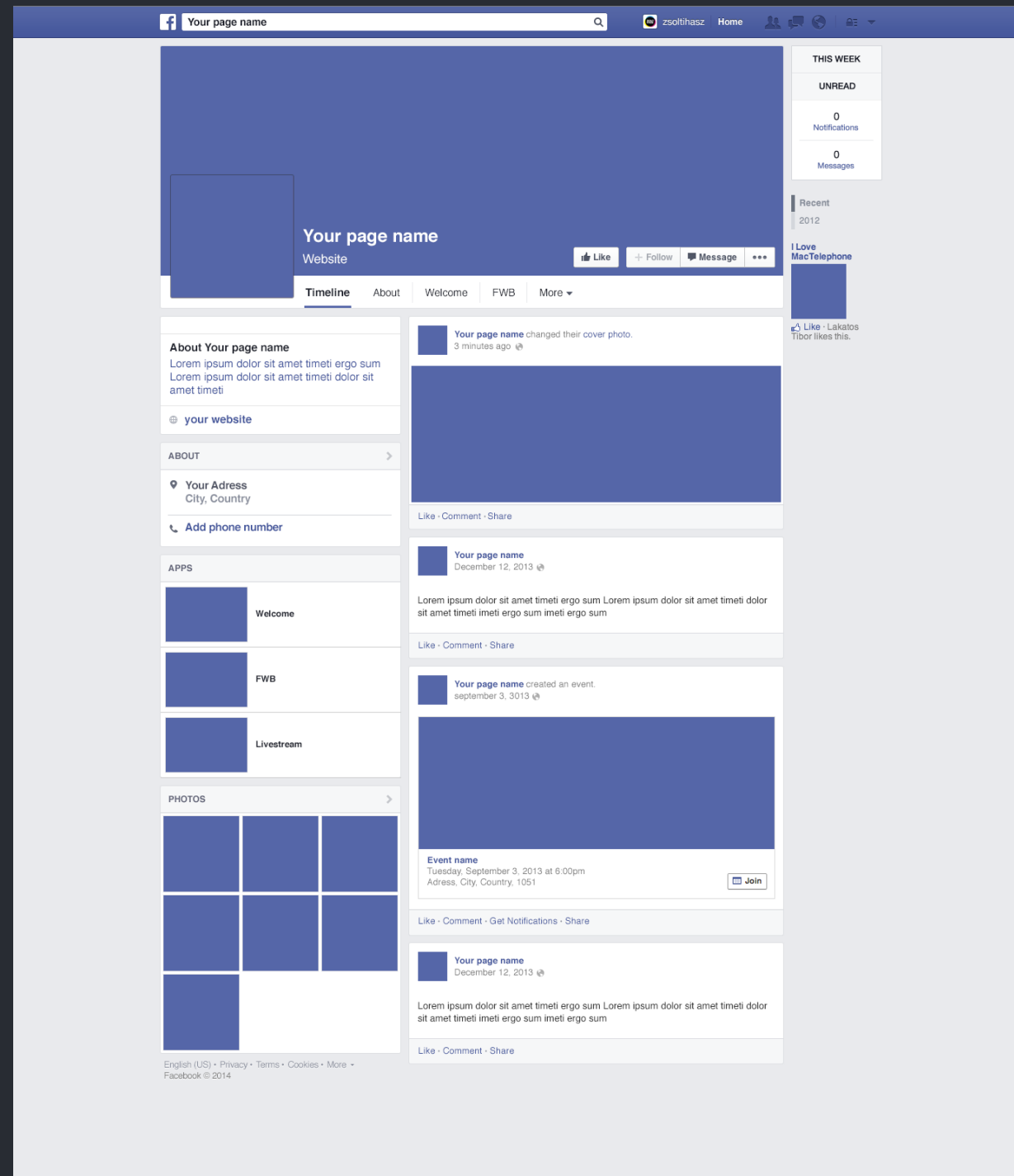
www.jedu.ir



www.karjoo.plus



Learn about the differences between using pure **JavaScript** or a library like **React** to design and develop web apps



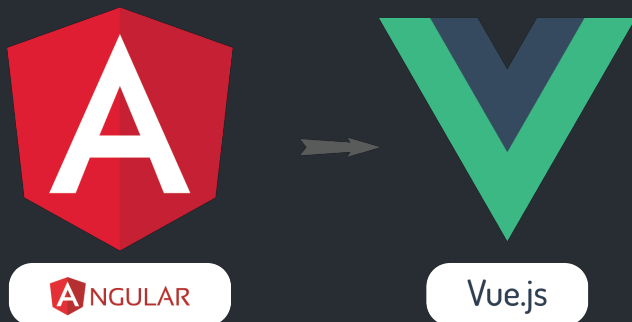
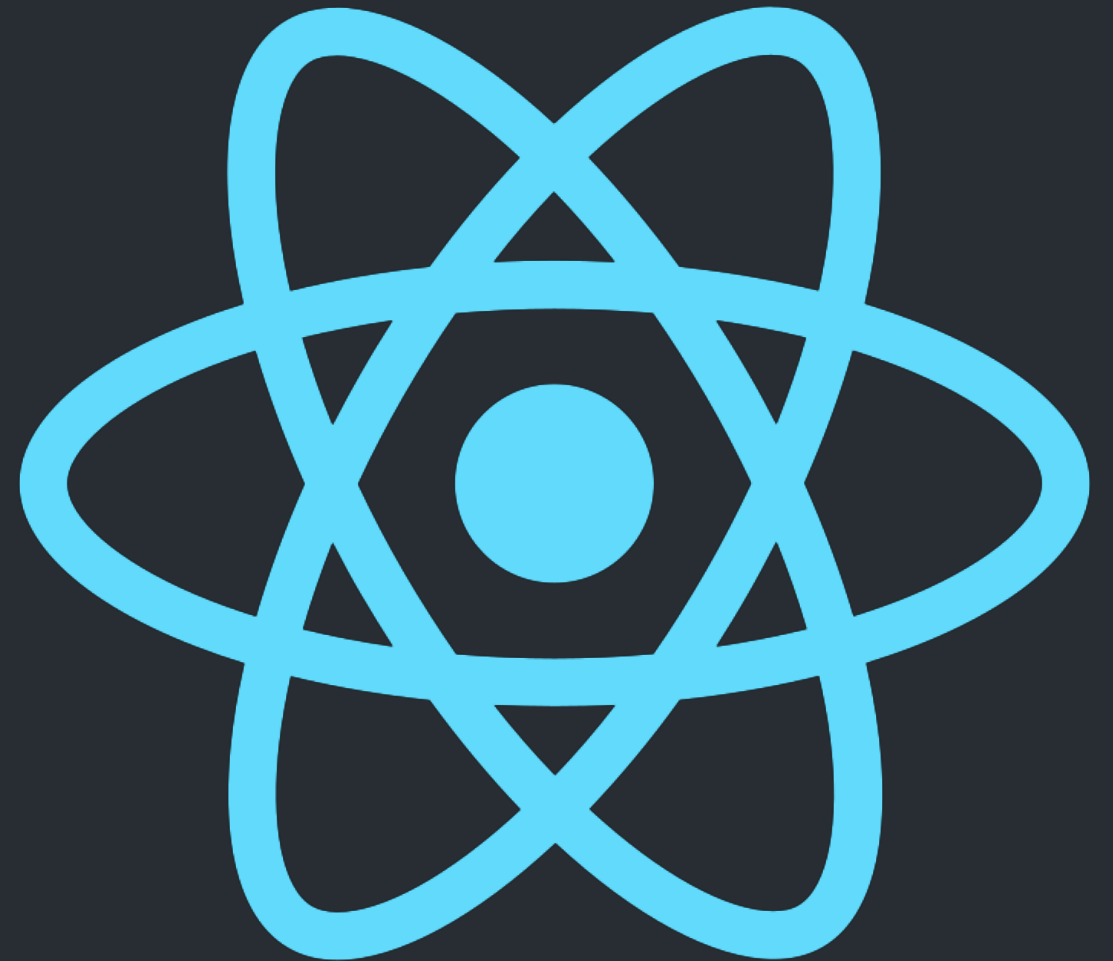
Web apps are becoming increasingly complex and dynamic. In response, new tools and libraries like **React** have been created to speed up the process.

WHAT IS PLAIN JAVASCRIPT?

It's important to point out that **React** itself is written in JavaScript

React

- React is a library that defines the way apps are written
- clear rules about how data can flow through the app
- how the UI will adapt as a result of that changing data



WHAT IS PLAIN JAVASCRIPT?

doesn't set any rules about how data can be defined, or
how the UI can be changed!



THE MAJOR DIFFERENCES

- How the user interface is first created
- How functionality is split up across the app
- How data is stored on the browser
- How the UI is updated

HOW THE USER INTERFACE IS FIRST CREATED

In Plain JavaScript

HTML is dynamically created
on the server, and might look
something like this:

```
<div>  
  <h1>Course List</h1>  
  <ul>  
    <li>Javascript</li>  
    <li>HTML</li>  
    <li>CSS</li>  
  </ul>  
</div>
```


HOW THE USER INTERFACE IS FIRST CREATED

In React

React app will start with a fixed HTML file that looks like this:

```
<div id="root"></div>
```



the app starts with a blank container (a div in this case), and then the UI gets loaded into that container.

Instead of defining the initial UI on the server, the UI gets defined on the browser

HOW THE USER INTERFACE IS FIRST CREATED

In React

The UI is defined by a component that returns JSX

that new CourseList component gets mounted (or "rendered") into the div container using a library called ReactDOM:



```
function CourseList(props) {  
  return (  
    <div>  
      <h1>Course List</h1>  
      <ul>  
        <li>Javascript</li>  
        <li>HTML</li>  
        <li>CSS</li>  
      </ul>  
    </div>  
  );  
}
```

```
ReactDOM.render(<CourseList />, document.getElementById('root'));
```

HOW FUNCTIONALITY IS SPLIT UP ACROSS THE APP

In Plain JavaScript

With a plain JS app, there are no requirements about how you split up functionality or UI components in an application.

And the code that updates the list might be in a separate javascript file:

```
<div>  
  <h1>Course List</h1>  
  <ul>  
    <li>Javascript</li>  
    <li>HTML</li>  
    <li>CSS</li>  
  </ul>  
</div>
```

```
function addCourseToList() {  
  // Add course  
}
```

HOW FUNCTIONALITY IS SPLIT UP ACROSS THE APP

In Plain JavaScript

However, as the **complexity** of JavaScript apps **has grown**, this has caused huge headaches.

Because the code that **updates** a piece of HTML might live in **several different JS files** across the entire application, developers have to keep all of those files open at once

```
<div>
  <h1>Course List</h1>
  <ul>
    <li>Javascript</li>
    <li>HTML</li>
    <li>CSS</li>
  </ul>
</div>
```

```
function addCourseToList() {
  // Add course
}
```

HOW FUNCTIONALITY IS SPLIT UP ACROSS THE APP

In **React**

React enforces that your app is split into **components**

components: maintains all of the code needed to *both* display and update the UI..

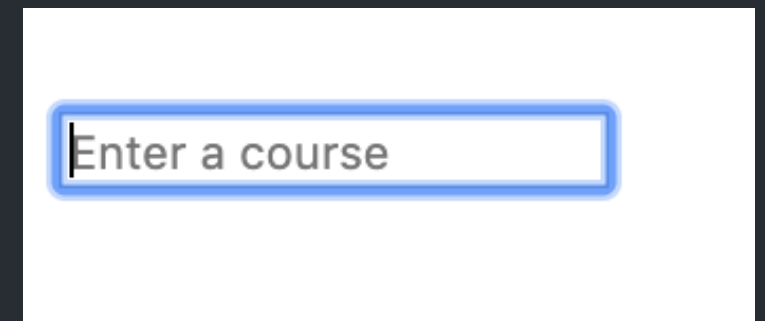
It also allows for greater code reuse since generic components can be made and shared across an app

```
function CourseList(props) {  
  function addCourseToList() {  
    // Add course  
  }  
  return (  
    <div>  
      <h1>Course List</h1>  
      <ul>  
        <li>Javascript</li>  
        <li>HTML</li>  
        <li>CSS</li>  
      </ul>  
    </div>  
  );  
}
```

HOW DATA IS STORED ON THE BROWSER

Once the initial UI is loaded, the user will be able to interact with your app

For interactions like typing into an input box, that text has to be stored somewhere on the browser before it can be used later...

A rectangular text input field with a light blue border and a placeholder text "Enter a course". The input field is centered within a white square background.

```
<input type="text" placeholder="Enter a course" id="item-input" />;
```

*In a plain **JavaScript** app, that user data is generally stored in the **DOM**
(Document Object Model)*

HOW DATA IS STORED ON THE BROWSER

In Plain JavaScript

As the user types into that textbox,
the value of what they are typing is
stored by the browser



It means the actual input UI changes as the user types is abstracted away from the developer...

HOW DATA IS STORED ON THE BROWSER

In Plain JavaScript

also means that when the user submits the form, the developer will have to manually extract the value from that input box by finding it in the DOM first, and then extracting the value:



```
const input = document.getElementById("item-input");  
console.log(input.value);
```

But it can get tedious for an entire form...

HOW DATA IS STORED ON THE BROWSER

In React

React uses a technique called "controlled components"

"A **controlled component** is a **react component** that controls the values of input elements in a form using `useState()`"

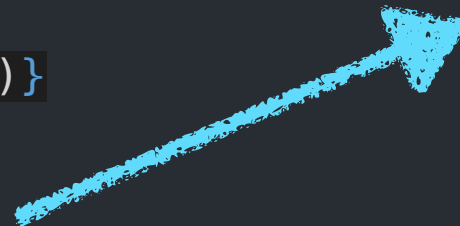


JavaScript

```
const [course, setCourse] = React.useState("");
```

```
<input  
  placeholder="Enter a course"  
  type="text"  
  value={course}  
  onChange={e => setCourse(e.target.value)}  
/>
```

```
console.log(course);
```

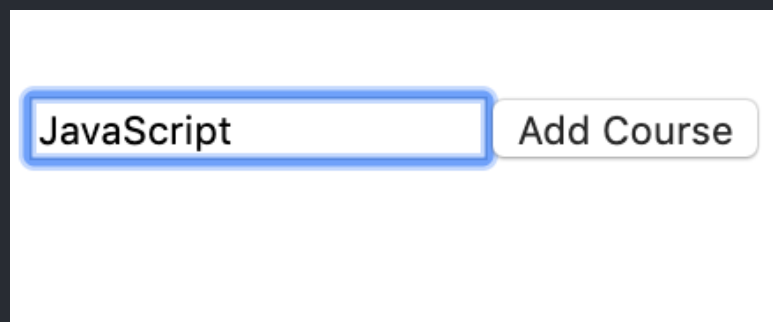


*But it makes it much easier to know the current value of the input box in JavaScript, because it's simply reading the value from **memory***

Storing the entire current state of the app in **JS variables** (instead of the DOM) is one of the major benefits **React apps** have over **plain JavaScript apps**, especially as the complexity of your app grows

HOW THE UI IS UPDATED

*how each app responds to user interaction—like a **button** press to actually add a new item to list—and then updates the UI to reflect that new change.*

A screenshot of a web application interface. It features a white rectangular container. Inside, there is a text input field with a blue border and rounded corners, containing the text 'JavaScript'. To the right of the input field is a button with a light gray background, rounded corners, and the text 'Add Course' in a dark gray font.

HOW THE UI IS UPDATED

In Plain JavaScript

As the user types into that textbox,
the value of what they are typing is
stored by the browser

A UI mockup showing a text input field with the value 'JavaScript' and an 'Add Course' button.

```
<input type="text" placeholder="Enter a course" id="item-input" />  
<button id="add-button">Add Course</button>
```

```
const addButton = document.getElementById("add-button");
```

```
addButton.addEventListener("click", function() {  
    // Append item  
})
```

set a **click listener** on that button

HOW THE UI IS UPDATED

In Plain JavaScript

```
addButton.addEventListener("click", function() {  
  const input = document.getElementById("item-input");  
  console.log(input.value);  
  
  const list = document.getElementById("course-list");  
  const listNode = document.createElement("li");  
  const textNode = document.createTextNode(input.value);  
  
  listNode.appendChild(textNode);  
  list.appendChild(listNode);  
});
```



- JavaScript

HOW THE UI IS UPDATED

In React

set up to keep the entire state of the list in a JS variable

```
const [courseList, setCourseList] = React.useState([]);
```

displayed in JSX by mapping (looping) over each item, and returning a list element for each one

```
<button onClick={addCourse}>Add Course</button>
```

```
const addCourse = () => {  
  setCourseList([...courseList, course]);  
};
```

Spread syntax

```
<ul>  
  {courseList.map(item => (  
    <li>{item}</li>  
  ))}  
</ul>
```

HOW THE UI IS UPDATED

In React

React will automatically register that there has been a change to the list, and update the UI automatically.

```
addButton.addEventListener("click", function() {  
  const input = document.getElementById("item-input");  
  console.log(input.value);  
  
  const list = document.getElementById("grocery-list");  
  const listNode = document.createElement("li");  
  const textNode = document.createTextNode(input.value);  
  
  listNode.appendChild(textNode);  
  list.appendChild(listNode);  
});
```

Updater function in Plain JavaScript

```
const addCourse = () => {  
  setCourseList([...courseList, course]);  
};
```

Updater function In React

HOW THE UI IS UPDATED

In React

What is the benefit of **automatically** updating?

```
addButton.addEventListener("click", function() {  
  const input = document.getElementById("item-input");  
  console.log(input.value);  
  
  const list = document.getElementById("grocery-list");  
  const listNode = document.createElement("li");  
  const textNode = document.createTextNode(input.value);  
  
  listNode.appendChild(textNode);  
  list.appendChild(listNode);  
});
```

```
const addCourse = () => {  
  setCourseList([...courseList, course]);  
};
```


HOW THE UI IS UPDATED

In React

*The **automatically** updating nature of React apps means that you don't have to go into the DOM to **find** where to append your item*

it just happens automatically for you.



Challenge...

Course List

- javascript

1

Course List

2