

Hello React

The lesson gives you an introduction to React. You may ask yourself: Why should I learn React in the first place? The lesson might give you the answer to that question.

We'll cover the following



- Hello React
- Requirements for a React Project
 - Editor and Terminal
 - Node and NPM
- Exercises

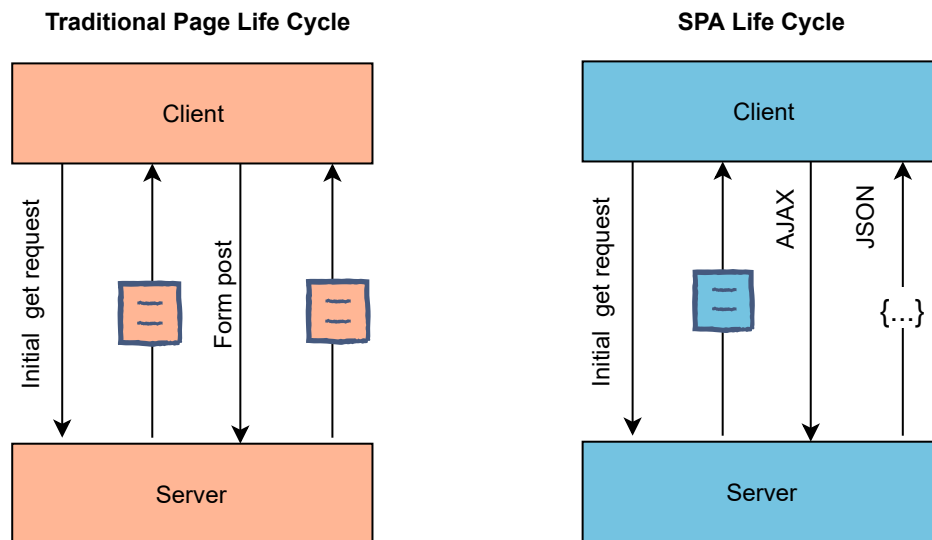
In the first part of this learning experience, we'll cover the fundamentals of React, after which we'll create our first React project. Then we'll explore new aspects of React by implementing real features, the same as developing an actual web application. By the end, we'll have a working React application with features like client and server-side searching, remote data fetching, and advanced state management.

Hello React

Single-page applications ([SPA](#)) have become increasingly popular with first-generation SPA frameworks like Angular (by Google), Ember, Knockout, and Backbone. Using these frameworks makes it easier to build web applications which advanced beyond vanilla JavaScript and jQuery. React, yet another solution for SPAs, was released by Facebook later in 2013. All of them are used to create entire web applications in JavaScript.

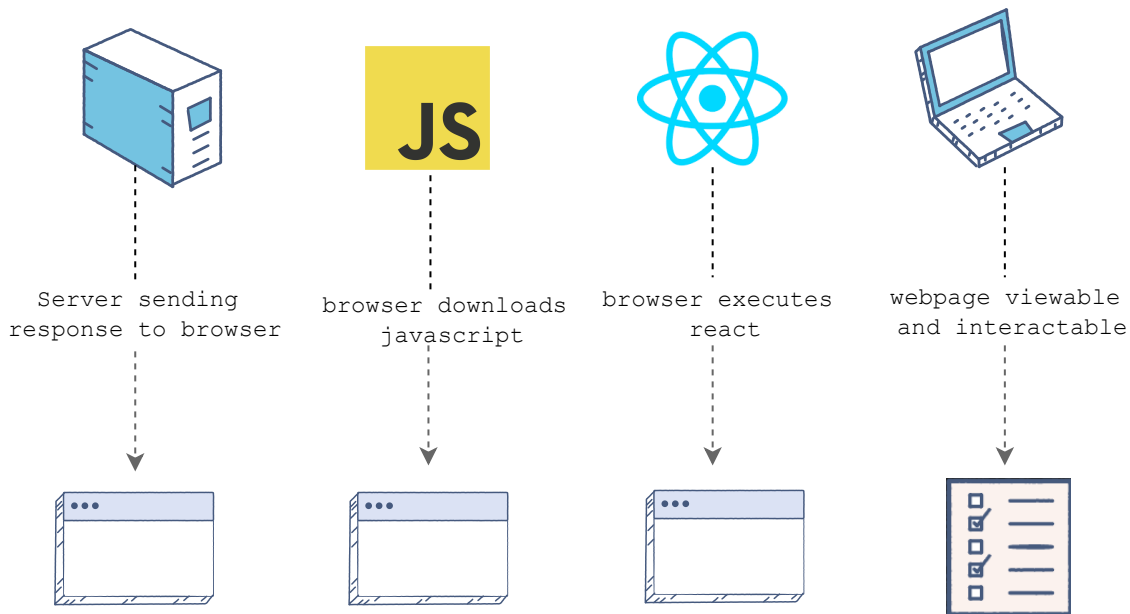
For a moment, let's go back in time before SPAs existed: In the past, websites and web applications were rendered from the server. A user visits a URL in a browser and requests one HTML file and all its associated HTML, CSS, and JavaScript files from a web server. After some network delay, the user sees the rendered HTML in the browser (client) and starts to interact with it. Every additional page transition (meaning: visiting another URL) would initiate this chain of events again. In this

version from the past, essentially everything crucial is done by the server, whereas the client plays a minimal role by just rendering page by page. While barebones HTML and CSS was used to structure the application, just a little bit of JavaScript was thrown into the mix to make interactions (e.g. toggling a dropdown) or advanced styling (e.g. positioning a tooltip) possible. A popular JavaScript library for this kind of work was jQuery.



In contrast, modern JavaScript shifted the focus from the server to the client. The most extreme version of it: A user visits a URL and requests one *minimal HTML file* and one *associated JavaScript file*. After some network delay, the user sees the *by JavaScript rendered HTML* in the browser and starts to interact with it. Every additional page transition *wouldn't* request more files from the web server, but would use the initially requested JavaScript to render the new page. Also every additional interaction by the user is handled on the client too. In this modern version, the server delivers mainly JavaScript across the wire with one minimal HTML file. The HTML file then executes all the linked JavaScript on the client-side to render the entire application with HTML and JavaScript for its interactions.

React, among the other SPA solutions, makes this possible. Essentially a SPA is one bulk of JavaScript, which is neatly organized in folders and files, to create a whole application whereas the SPA framework gives you all the tools to architect it. This JavaScript focused application is delivered once over the network to your browser when a user visits the URL for your web application. From there, React, or any other SPA framework takes over for rendering everything in the browser and for dealing with user interactions.



With the rise of React, the concept of components became popular. Every component defines its look and feel with HTML, CSS and JavaScript. Once a component is defined, it can be used in a hierarchy of components for creating an entire application. Even though React has a strong focus on components as a library, the surrounding ecosystem makes it a flexible framework. React has a slim API, a stable yet thriving ecosystem, and a welcoming community. We are happy to welcome you :-)

Requirements for a React Project

To follow this course you'll need to be familiar with the basics of web development, i.e., how to use HTML, CSS, and JavaScript. It also helps to understand [APIs](#), as they will be covered thoroughly.


Editor and Terminal

For the entire course, you do not need an IDE on your local machine, you can use the **Educative SPA widget** to see the live execution of codes.

📖 While Educative SPA widget is a great tool for sharing code online, local machine setup is a better tool for learning different ways to create a web application. Also, if you want to develop applications professionally, a local setup will be required. [Steps for setting up a local environment](#) are mentioned in the Appendix chapter.

Node and NPM

Before we can begin, we'll need to have [node and npm](#) installed. Both are used to manage libraries (node packages) you will need along the way. These node packages can be libraries or whole frameworks.

 We have installed external node packages via npm (node package manager).

You can check the version of node and npm by typing the following commands in the terminal below:

```
node --version
```

```
npm --version
```

● Terminal



Exercises

- Read more about [why I moved from Angular to React](#).
- Read more about [how to learn a framework](#).
- Read more about [how to learn React](#).
- Read more about [why frameworks matter](#).
- Scan through [JavaScript fundamentals needed for React](#) – without worrying too much about the React – to test yourself about several JavaScript features used in React.