Hello! Let's get started with React. And the first question is What is React ?

React is an open-source , component-based library.  It was maintained by Facebook in 2013. React is currently one of the most popular JavaScript front-end libraries in the Software Industry. And we can see the increase of React JS popularity in the last 5 years by Google Trends.

### React is not a full-fledged framework that has an opinion on how everything in your app should behave. Instead, React works primarily in the View layer where all of its worries and concerns revolve around your visual elements and keeping them up-to-date. This means you are free to use whatever you want for the M and C part of your MVC architecture.

### React can also render on the server using Node, and it can power native apps using React Native.

### Create React App

[Create React App](https://github.com/facebookincubator/create-react-app) is a comfortable environment for learningReact, and is the best way to start building **a** new[single-page](https://reactjs.org/docs/glossary.html" \l "single-page-application)application in React.

It sets up your development environment so that you can use the latest JavaScript features, provides a nice developer experience, and optimizes your app for production. Under the hood, it uses [Babel](https://babeljs.io/" \t "_blank) and [webpack](https://webpack.js.org/" \t "_blank), but you don’t need to know anything about them.

**React Core Concepts**

React has some unique core concepts. It has a virtual DOM, JSX, components, props. Also, each React component has a state and a lifecycle, which we will go into.

**Virtual DOM**

DOM manipulation is the heart of the modern, interactive web. Unfortunately, it is also a lot slower than most JavaScript operations. Each time you make a change in the code, DOM will be completely updated and rewritten. This is an expensive operation and consumes lots of time. In this point, React provides a solution: The Virtual DOM.

The virtual DOM is only a virtual representation of the real DOM, lightweight copy. Everytime the state of our application changes, the virtual DOM gets updated instead of the real DOM.

So when something changes:

React first creates an exact copy of the DOM.

Then React figures out which part is new and only updates that specific part in the Virtual DOM.

Finally, React copies only the new parts of the Virtual DOM to the actual DOM, rather than completely rewriting it.

This approach makes a webpage much faster than a standard webpage. That’s also one of the reasons why React is so popular.

**JSX**

JSX is a syntax extension to JavaScript used by React.

JSX allows us to write HTML elements in JavaScript and place them in the DOM without any createElement()  and/or appendChild() methods. Later, the JSX code will be translated into normal JavaScript, by Babel.

In summary, React doesn’t have HTML files. This approach makes React faster.

Some important rules about JSX:

* The HTML code must be wrapped in ONE top level element. So if you need to write two paragraphes, you must put them inside a parent element, like a div element.
* With JSX you can write expressions inside curly braces { }. The expression can be a React variable, or property, or any other valid JavaScript expression. JSX will execute the expression and return the result.
* JSX follows XML rules, and therefore HTML elements must be properly closed.

**Components**

A component is an independent, reusable code block, which divides the UI into smaller pieces.

Dividing the source code into components helps us a lot. Each component has its own JS and CSS code, they are reusable, easier to read, write and test.

In web development, as the reasons I explained right above, it’s beneficial to use component-based technologies, and React JS is one of them.

React has 2 types of components: Functional (Stateless) and Class (Stateful).

Let’s see them in details…

**Functional (Stateless) Components**

A functional component is basically a JavaScript function or ES6 arrow function.

Both of the functions are valid React components. They may take props as an argument (when necessary), but they mustreturn a React element.

Functional components are also known as stateless components because, in the past, we couldn’t do more complex things like React State (data) management or life-cycle methods in functional components. However, React introduced React Hooks in version 16.8, which now allows us to use state & other features in functional components.

**Class (Stateful) Components**

Class components are ES6 classes. They are more complex than functional components including constructors, life-cycle methods, render( ) function and state (data) management. In the example, we can see how a simple class component looks like: (see slide)

Here, the Example class extends Component, so React understands that this class is a component, and it renders (returns) a React Element.

**PROPS**

“Props” is a special keyword in React, which stands for properties and is being used for passing data from one component to another.

But the important part here is that data with props are being passed in a uni-directional flow (one way from parent to child).

Furthermore, props data is read-only, which means that data coming from the parent should not be changed by child components.

**STATE**

State is a special object that holds dynamic data, which means that state can change over time and anytime based on user actions or certain events.

State can only be used within a class component. State is private and belongs only to its component where defined, cannot be accessed from outside, but can be passed to child components via props.

State of a component should prevail throughout the lifetime, thus we must first have some initial state, to do so we should define the State in the constructor of the component’s class.

The state object can contain as many properties as you like.

State should never be updated explicitly.

React uses an observable object as the state that observes what changes are made to the state and helps the component behave accordingly. So we need use the this.setState() method to change a value in the state object. Once the update is done the method implicitly calls the render() method to repaint the page.

**LIFECYCLE**

Each component in React has a lifecycle which you can monitor and manipulate during its three main phases: Mounting, Updating, and Unmounting.

Mounting means putting elements into the DOM.

The next phase in the lifecycle is when a component is updated.

A component is updated whenever there is a change in the component's state or props.

In Mounting and Updating phases the render() method is required and will always be called, the others are optionaland will be called if you define them.

The next phase in the lifecycle is when a component is removed from the DOM, or unmounting as React likes to call it.

Let's sum it all up and highlight the pros and cons /strengths and weaknesses/ of ReactJS

**Pros of ReactJS**

1. Virtual DOM enabled developers to work with UI-objects faster and use hot reloading (applying changes in a real-time mode).
2. Permission to reuse React components significantly saves time.
3. One-direction data flow in ReactJS provides a stable code. In such a structure, child elements cannot affect parent data. To change an object, all a developer needs to do is modify its state and apply updates. Correspondingly, only allowed components will be upgraded.
4. An open-source Facebook library. ReactJS uses all advantages of free access – a lot of useful applications and additional tools from off-company developers.
5. Wide React toolset. For example, [React Developer Tools](https://chrome.google.com/webstore/detail/react-developer-tools/fmkadmapgofadopljbjfkapdkoienihi?hl=en" \t "_blank) extension for Chrome and a similar one for [Firefox](https://addons.mozilla.org/en-US/firefox/addon/react-devtools/" \t "_blank) wich allow for examining component hierarchies(хаеракис) in the virtual DOM and editing states and properties.
6. ReactJS is well-suited for Single-Page-Applications

**Cons of ReactJS**

1. High pace of development. The environment constantly changes, and developers must regularly relearn the new ways of doing things.
2. The problem with documentation traces back to constant releases of new tools. React technologies are updating and accelerating so fast that there is no time to write proper instruction.
3. Developers and designers complain about JSX’s complexity and consequent steep learning curve.