React Js

Библиотека для JavaScript

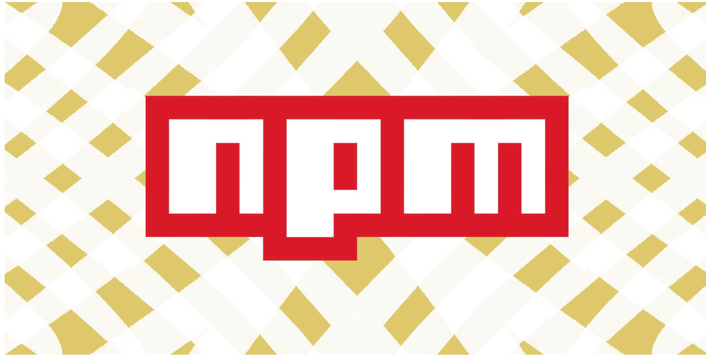
Static HTML File

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8" />

    <title>Hello React!</title>

    <script src="https://unpkg.com/react@^16/umd/react.production.min.js"></script>
    <script src="https://unpkg.com/react-dom@16.13.0/umd/react-dom.production.min.js"></script>
    <script src="https://unpkg.com/babel-standalone@6.26.0/babel.js"></script>
  </head>

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

    <script type="text/babel">
      // React code will go here
    </script>
  </body>
</html>
```

React - the React top level API
React DOM - Специальные методы для DOM
Babel - JS компилятор, который позволяет использовать ES6+ в старых браузерах

Static HTML File

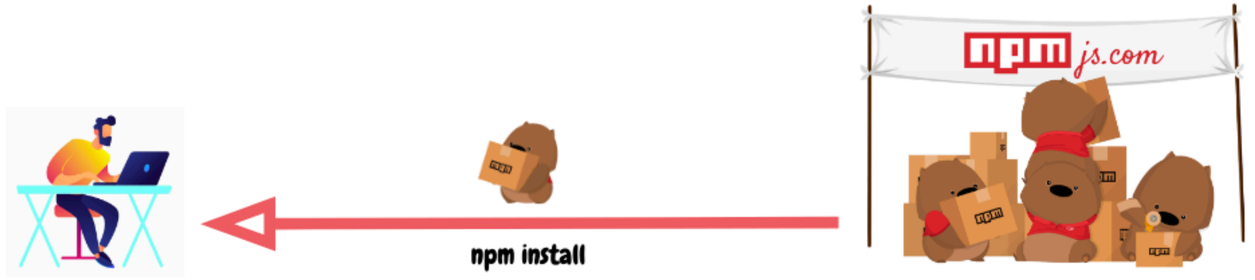
```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8" />

    <title>Hello React!</title>

    <script src="https://unpkg.com/react@16/umd/react.development.js"></script>
    <script src="https://unpkg.com/react-dom@16/umd/react-dom.development.js"></script>
    <script src="https://unpkg.com/babel-standalone@6.26.0/babel.js"></script>
  </head>

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

    <script type="text/babel">
      class App extends React.Component {
        render() {
          return <h1>Hello world!</h1>
        }
      }

      ReactDOM.render(<App />, document.getElementById('root'))
    </script>
  </body>
</html>
```



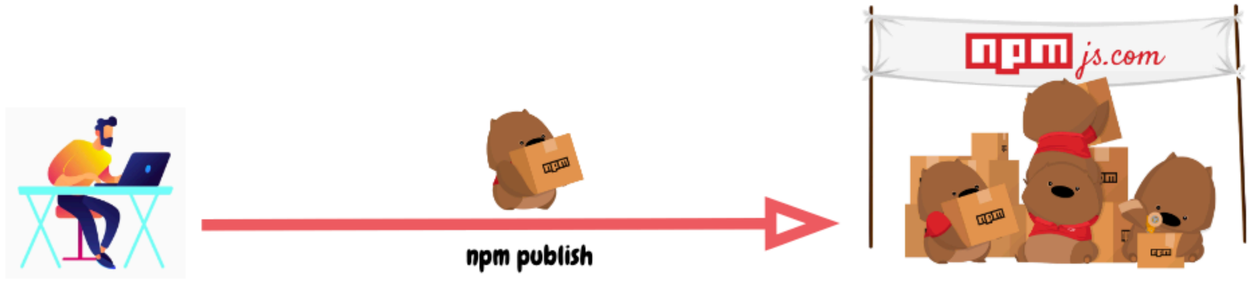
npm (Node Package Manager) – дефолтный пакетный менеджер для JavaScript, работающий на Node.js. Менеджер npm состоит из двух частей:

- CLI (интерфейс командной строки) – средство для размещения и скачивания пакетов,
- онлайн-репозитории, содержащие JS пакеты.

В центре исполнения заказов (npmjs.com) в качестве персональных менеджеров для каждого покупателя работает армия вомбатов (npm CLI).



Процесс установки пакета через npm install



Процесс размещения пакета через npm publish

Файл package.json

Каждый проект в JavaScript – будь то Node.js или веб-приложение – может быть скопирован как npm-пакет с собственным описанием и файлом package.json.

package.json можно представить, как стикеры (список пакетов нужных версий) на npm-коробке (проект). Файл генерируется командой npm init при создании JavaScript/Node.js проекта со следующими метаданными:

- name: название JS библиотеки/проекта.
- version: версия проекта.
- description: описание проекта.
- license: лицензия проекта.

Скрипты npm

В package.json включено поле scripts для автоматизации сборки, например:

```
"scripts": {
    "start": "react-scripts start",
    "build": "react-scripts build",
    "test": "react-scripts test",
    "eject": "react-scripts eject"
  },
```

dependencies и devDependencies

dependencies и devdependencies представляют собой словари с именами npm-библиотек (ключ) и их семантические версии (значение)

Эти зависимости устанавливаются командной npm install с флагами --save и --save-dev. Они предназначены соответственно для использования в продакшене и разработке.

```
"dependencies": {
  "@testing-library/jest-dom": "^5.16.5",
  "@testing-library/react": "^13.4.0",
  "@testing-library/user-event": "^13.5.0",
  "react": "^18.2.0",
  "react-dom": "^18.2.0",
  "react-scripts": "5.0.1",
  "web-vitals": "^2.1.4"
},
"devDependencies": {
  "eslint": "^6.8.0",
  "eslint-plugin-github": "^3.4.1",
  "eslint-plugin-jest": "^23.8.2",
  "prettier": "^1.19.1",
  "ts-jest": "^25.2.1",
  "typescript": "^3.8.3"
},
```

Файл package-lock.json

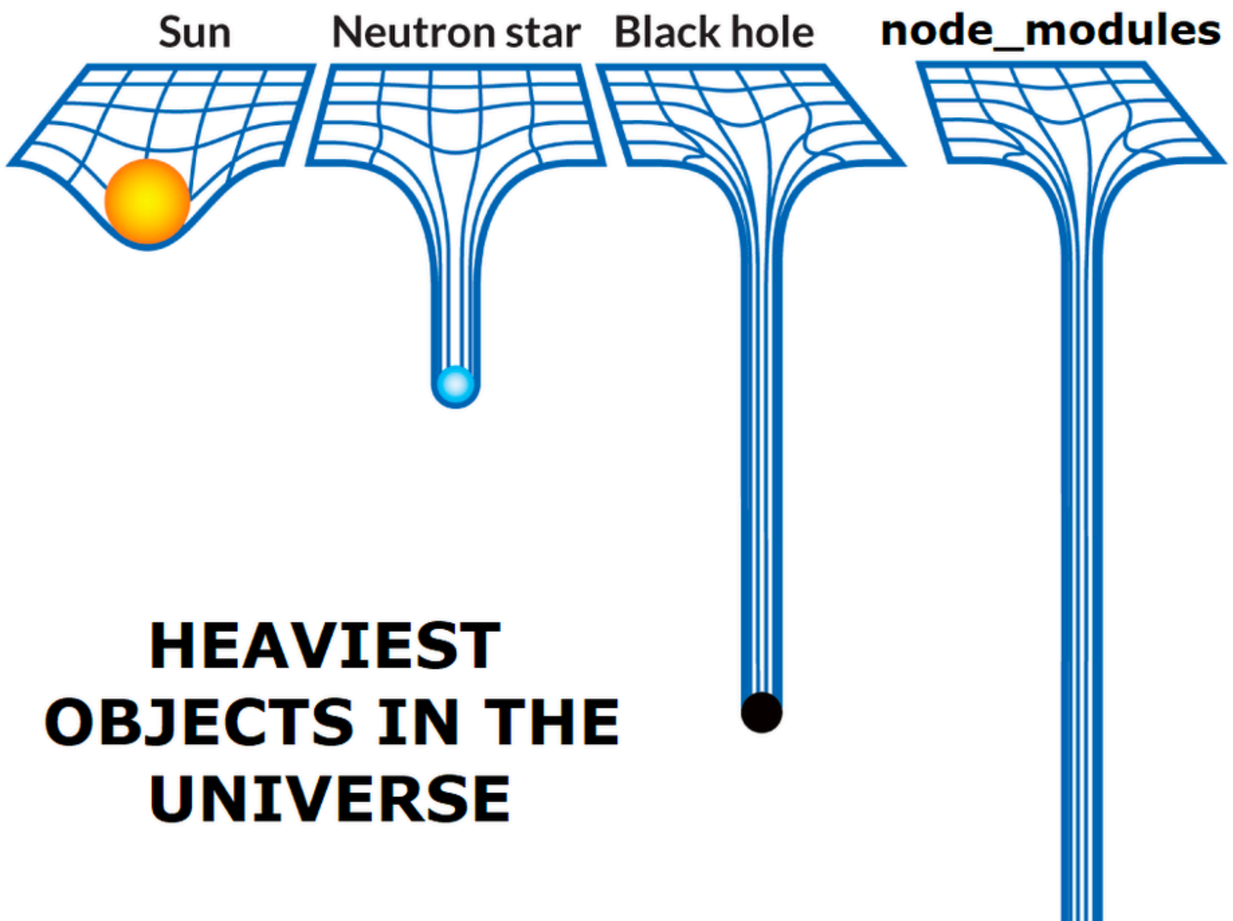
Файл package-lock.json описывает версии пакетов, используемые в JavaScript-проекте. Если package.json включает общее описание зависимостей (название товара), то package-lock.json более детальный – всё дерево зависимостей.

package-lock.json генерируется командой npm install и читается npm CLI, чтобы обеспечить воспроизведение окружения для проекта через npm ci.

```
{
  "name": "beginning",
  "version": "0.1.0",
  "lockfileVersion": 3,
  "requires": true,
  "packages": {
    "": {
      "name": "beginning",
      "version": "0.1.0",
      "dependencies": {
        "@testing-library/jest-dom": "^5.16.5",      You, yesterday • Initialize proje
        "@testing-library/react": "^13.4.0",
        "@testing-library/user-event": "^13.5.0",
        "react": "^18.2.0",
        "react-dom": "^18.2.0",
        "react-scripts": "5.0.1",
        "web-vitals": "^2.1.4"
      }
    },
    "node_modules/@adobe/css-tools": {
      "version": "4.2.0",
      "resolved": "https://registry.npmjs.org/@adobe/css-tools/-/css-tools-4.2.0.tgz",
      "integrity": "sha512-E09FiIft46CmH5Qnjb0wsW54/YQd69LsxeKUOWawmws1XWvyFGURnAChH0m
    },
    "node_modules/@alloc/quick-lru": {
      "version": "5.2.0",
      "resolved": "https://registry.npmjs.org/@alloc/quick-lru/-/quick-lru-5.2.0.tgz",
      "integrity": "sha512-UrcABB+4bUrFABwbluTIBErXwvbsU/V7TZWfmbgJfbkwiBuziS9gxdODUyu
      "engines": {
        "node": "⩾10"
      },
```

npm install

По умолчанию npm install <package-name> со знаком ^ установит последнюю версию пакета. npm install скачает пакет в папку проекта node_modules в соответствии с конфигурацией в файле package.json, обновив версию пакета везде, где это возможно (и, в свою очередь, обновив package-lock.json). При необходимости установки пакета глобально можно указать флаг -g .



Create React App

npm init react-app <project-name>

yarn create react-app <project-name>

npx create-react-app <project-name>

JSX: JavaScript + XML

В нашем коде React мы использовали то, что выглядит как HTML, но это не совсем HTML. Это JSX, что означает JavaScript XML.

С помощью JSX мы можем писать то, что выглядит как HTML, а также мы можем создавать и использовать наши собственные XML-подобные теги. Вот как выглядит JSX, назначенный переменной.

```
const heading = <h1 className="site-heading">Hello, React</h1>
```

Использование JSX не является обязательным для написания React. Под капотом работает createElement, который принимает тег, объект, содержащий свойства, и дочерние элементы компонента и отображает ту же информацию. Приведенный ниже код будет иметь тот же результат, что и приведенный выше JSX.

```
const heading = React.createElement('h1', { className: 'site-heading' }, 'Hello, React!')
```

Жизненный цикл React

Каждый компонент React проходит несколько стадий в процессе своей жизни: он создается, затем добавляется в DOM, получает пропсы, и, наконец, удаляется из дерева. Этот процесс называют жизненным циклом компонента (Component Lifecycle). React предоставляет набор методов, которые позволяют встроиться в этот процесс.

constructor(props): конструктор, в котором происходит начальная инициализация компонента

static **getDerivedStateFromProps**(props, state): вызывается непосредственно перед рендерингом компонента. Этот метод не имеет доступа к текущему объекту компонента (то есть обратиться к объкту компоненту через this) и должен возвращать объект для обновления объекта state или значение null, если нечего обновлять.

render(): рендеринг компонента

componentDidMount(): вызывается после рендеринга компонента. Здесь можно выполнять запросы к удаленным ресурсам

componentWillUnmount(): вызывается перед удалением компонента из DOM

Кроме этих основных этапов или событий жизненного цикла, также имеется еще ряд функций, которые вызываются при обновлении состояния после начального рендеринга компонента, если в компоненте происходят обновления:

static **getDerivedStateFromProps**(props, state)

shouldComponentUpdate(nextProps, nextState): вызывается каждый раз при обновлении объекта props или state. В качестве параметра передаются новый объект props и state. Эта функция должна возвращать true (надо делать обновление) или false (игнорировать обновление). По умолчанию возвращается true. Но если функция будет возвращать false, то тем самым мы отключим обновление компонента, а последующие функции не будут срабатывать.

render(): рендеринг компонента (если shouldComponentUpdate возвращает true)

getSnapshotBeforeUpdate(prevProps, prevState): Он позволяет компоненту получить информацию из DOM перед возможным обновлением. Возвращает в качестве значения какой-то отдельный аспект, который передается в качестве третьего параметра в метод componentDidUpdate() и может учитываться в componentDidUpdate при обновлении. Если нечего возвращать, то возвращается значение null

componentDidUpdate(prevProps, prevState, snapshot): вызывается сразу после обновления компонента (если shouldComponentUpdate возвращает true). В качестве параметров передаются старые значения объектов props и state. Третий параметр - значение, которое возвращает метод getSnapshotBeforeUpdate

"Render Phase"

Pure and has no side effects. May be paused, aborted or restarted by React.

"Pre-Commit Phase"

Can read the DOM.

"Commit Phase"

Can work with DOM, run side effects, schedule updates.