# React, an introduction

Justina Balsė
Julius Zabulėnas

TECHIN

# Content

- ReactJS
- NPM
- Node.js
- Babel
- ES5 vs. ES6
- JSX
- create-react-app
- Kosmosas Nr. 1 …
- Kosmosas Nr. 2 …
- Tikslas - https://superm.react-tutorial.app/


Justina Blasė, Julius Zabulėnas

# Prettier setup

- Let’s allow Prettier to format JSX attributes on each new line

TECHIN

# ReactJS

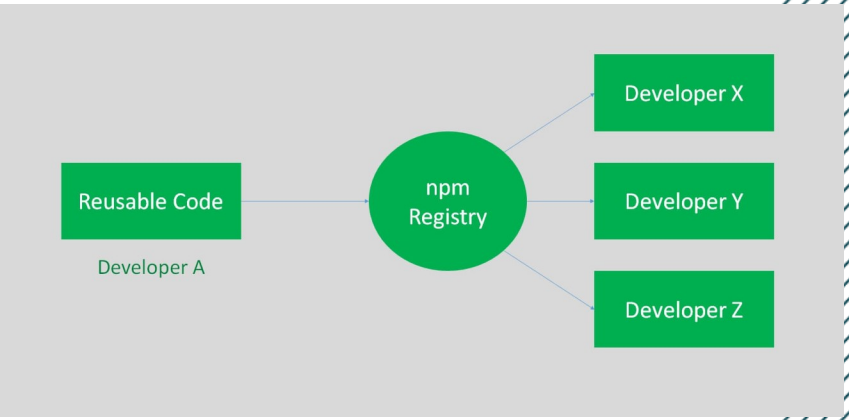
**React.js is a JavaScript library.** It was developed by engineers at Facebook.

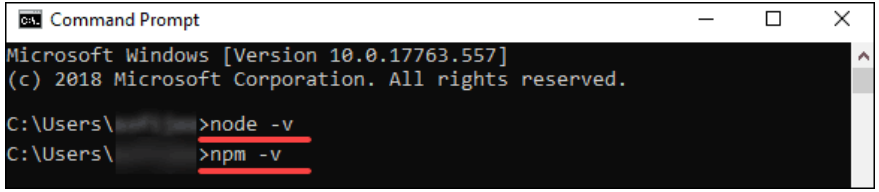
Here are just a few of the reasons why people choose to program with React:

- React is fast. Apps made in React can handle complex updates and still feel quick and responsive
- React is modular. Instead of writing large, dense files of code, you can write many smaller, reusable files
- React is scalable. Large programs that display a lot of changing data are where React performs best
- React is popular. While this reason has admittedly little to do with React's quality, the truth is that understanding React will make you more employable

# What is npm?

- npm - **Node Package Manager**
- It is an **online** repository for the publishing of open-source Node.js projects
- It is a command-line utility for interacting with said repository that aids in package installation, version management, and dependency management



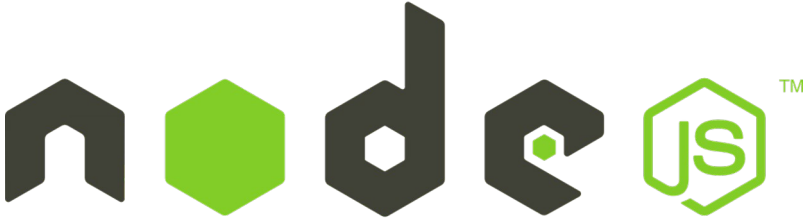






Justina Blasė, Julius Zabulėnas

TECHIN

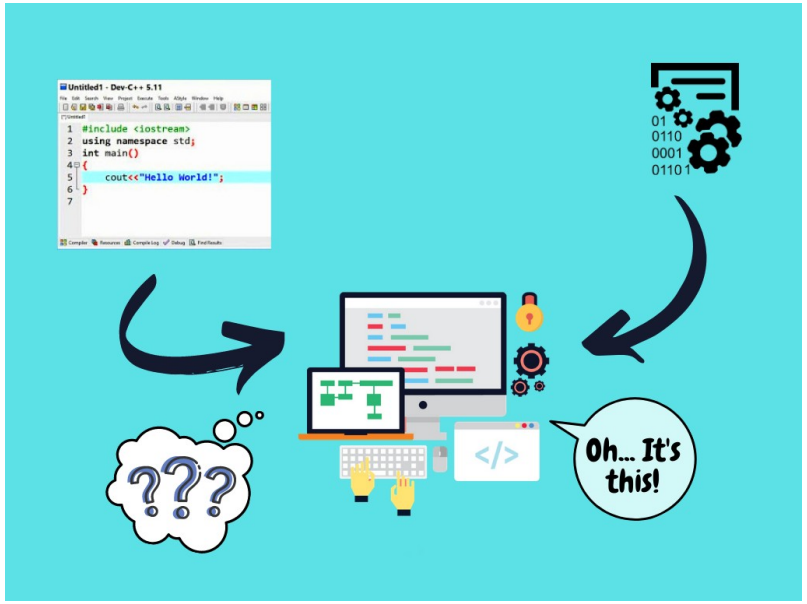
# What is node.js




- Node.js is an open source server environment
- Node.js is environment for execution of the JavaScript code without web browser
- https://nodejs.org/en/

# Kas yra *compiler*?

- Compiler is a program that converts instructions into a machine-code or lower-level form so that they can be read and executed by a computer
- Examples:
  - C++ → Assembly
  - Java → Java Bytecode
  - ES6 → ES5

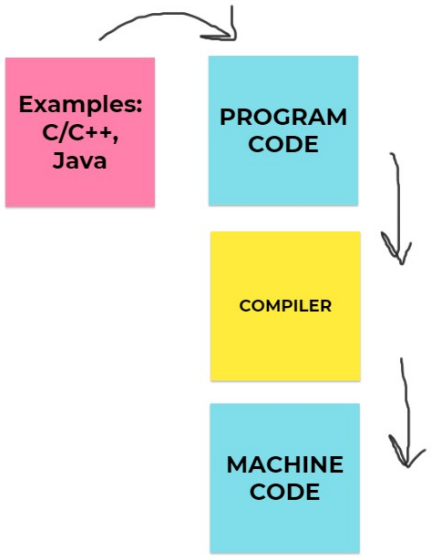







TECHIN

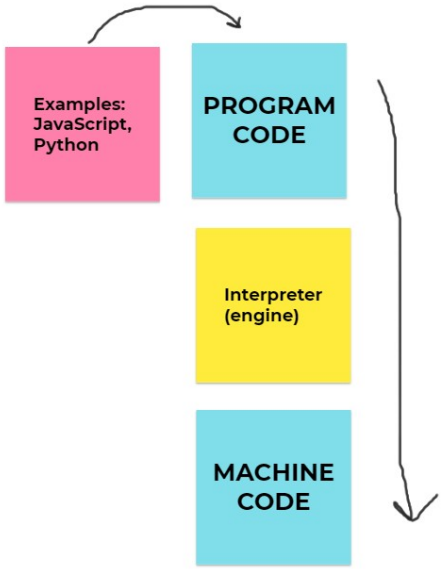
# Compiled Language




- Most programs are written in high-level languages like C, Perl, or Java
- Just as human languages facilitate communication among people, computer languages simplify the task of instructing a computer.
- Since computers primarily comprehend numbers, communicating with them is akin to conversing with someone who speaks a different language
- Interpreters and compilers serve as the necessary translators for effective communication

# Interpreted Language

- An interpreter translates code into machine code, instruction by instruction - the CPU executes each instruction before the interpreter moves on to translate the next instruction
- Interpreted code will show an error as soon as it hits a problem, so it is easier to debug than compiled code



# What is Babel?

- **Babel** is a JavaScript compiler
- Babel is a toolchain that is mainly used to convert ECMAScript 2015+ code into a backwards compatible version of JavaScript in current and older browsers or environments
- Try it!

```
// Babel Input: ES2015 arrow function
[1, 2, 3].map((n) => n + 1);

// Babel Output: ES5 equivalent
[1, 2, 3].map(function (n) {
  return n + 1;
});
```

When writing React applications, you will use a lot of JavaScript concepts, such as:

- const/let
- Template strings
- Arrays/Objects
- Array methods (filter, find, etc.)
- Spread operator
- Array/object destructuring
- Arrow functions

```
() => {}
```

# `var`, `let`, `const`

ES5

- The scope of a variable defined with **var** is **function scope** or declared outside any function, global.

ES6

- The scope of a variable defined with **let ir const** is block scope.

# Objects

```
// ES5

var person = {
  name: "John",
  sayHi: function () {
    console.log("Hi!");
  },
};

person.name;
person.sayHi();
```

```
// ES6

const person = {
  name: "John",
  sayHi() {
    console.log("Hi!");
  },
};

person.name;
person.sayHi();
```

# Arrow function expression

```
// ES5

// var sum = function (num1, num2) {
//  return num1 + num2;
// };

function sum (num1, num2) {
  return num1 + num2;
};
```

```
// ES6
// Geriau naudoti taip

const sum = (num1, num2) => num1
+ num2;
```

# Arrow function expressions

- Specifying parameters:
  - `() => { ... }  // no parameter`
  - `x => { ... }  // one parameter`
  - `(x, y) => { ... }  // several parameters`
- Specifying a body:
  - `x => { return x * x }  // block`
  - `x => x * x  // expression, equivalent to previous line`

TECHIN

# Map | Filter | Reduce

```
map([🐮, 🌮, 🐓, 🌽], cook)
=> [🍔, 🍟, 🍗, 🍿]

filter([🍔, 🍟, 🍗, 🍿], isVegetarian)
=> [🍟, 🍿]

reduce([🍔, 🍟, 🍗, 🍿], eat)
=> 💩
```

# Arrow function expression | map( )

```
const numbers = [1, 2, 3];

const newNumArr =
numbers.map(function (item) {
  return item * 2;
});

const newNumArr2 =
numbers.map((item) => item *
2);
```

# Arrow function expression | `filter()`

```
const tasks = [
  { id: 1, task: "make coffee", status: false
},
  { id: 2, task: "make tea", status: false },
  { id: 3, task: "eat a cow", status: true },
];


// ES5
var activeTasks1 = tasks.filter(function
(task) {
  return !task.status;
});


// ES6
const activeTasks2 = tasks.filter((task) => !
task.status);
```

# Arrow function expression | `reduce()`

```
const cart = [
  { id: 1, item: "coffee", price: 1.99 },
  { id: 2, item: "tea", price: 0.78 },
  { id: 3, item: "cow", price: 1000 },
];


// ES5
var sum1 = cart.reduce(function (sum, item) {
  return sum + item.price;
}, 0);


// ES6
const sum2 = cart.reduce((sum, item) => {
  return sum + item.price;
}, 0);
```

# Destructuring assignment

```
// Oh No!

const person = {
  firstName: "John",
  lastName: "Snow",
  age: 59,
};

const firstName =
person.firstName;
const lastName = person.lastName;
const age = person.age;
```

```
// Wow

const person = {
  firstName: "John",
  lastName: "Snow",
  age: 59,
};
const { firstName, lastName, age }
= person;

const { firstName: fn } = person;
console.log(fn); // John
```

# Spread syntax

```
// ES5

const arr1 = [23, 59, 61];
const arr2 = [71, 54, 96, 77];
const mergeArrays =
arr1.concat(arr2);
```

```
// ES6

const arr1 = [23, 59, 61];
const arr2 = [71, 54, 96, 77];
const mergeArrays1 =
[...arr1, ...arr2];
const mergeArrays2 =
["Hi", ...arr1,
"Wow", ...arr2];
```

# Spread syntax

```
const name = { firstName:
"John" };
const surname = { lastName:
"Snow" };

const newObject =
{ ...name, ...surname, age:
59 };

console.log(newObject);
```

```
▼ {…}
    age: 59
    firstname: "John"
    lastname: "Snow"
  ▶ <prototype>: Object { … }
```

# Naujas projektas - naujas `package.json`

- Kaip sukurti?
  - `npm init`
  - `npm init -y`
- Kas ten?
  - Informacija apie jūsų projektą: pavadinimas, versija, kūrėjas-autorius, ...
  - Ir vėliau – priklausomybės, t.y., *dependencies*

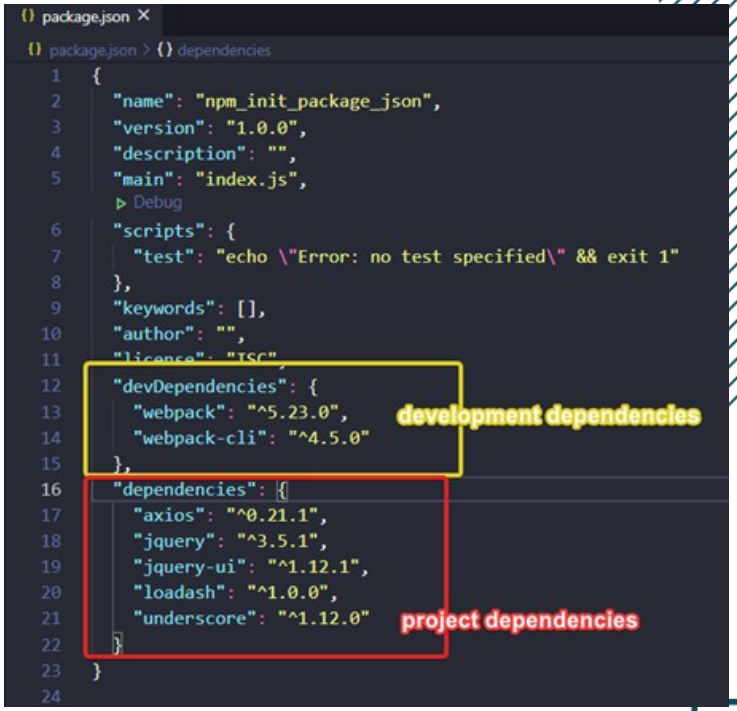
package.json > ...

```
1  {
2    "name": "breaking-bad-api-vanilla-js",
3    "version": "1.0.0",
4    "description": "API example",
5    "main": "main.js",
       ▷ Debug
6    "scripts": {
7      "test": "echo \"Error: no test specified\" && exit 1"
8    },
9    "keywords": [
10     "JavaScript",
11     "HTML",
12     "CSS",
13     "BOOTSTRAP",
14     "Breaking",
15     "Bad",
16     "API"
17   ],
18   "author": "Priebalse",
19   "license": "ISC"
20 }
21
```

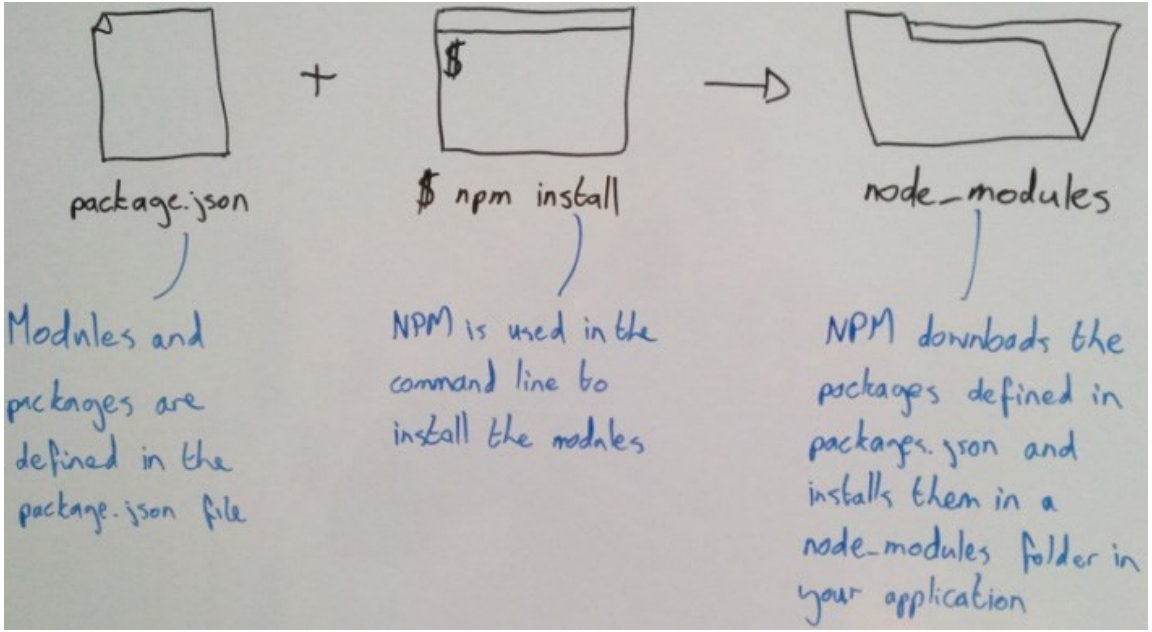
# package.json

- "dependencies": Packages required by your application in production.
  - `npm install <package-name>`

- "devDependencies": Packages that are only needed for local development and testing.
  - `npm install <package-name> --save-dev`



# package.json → npm install → node_modules



# What is JSX?

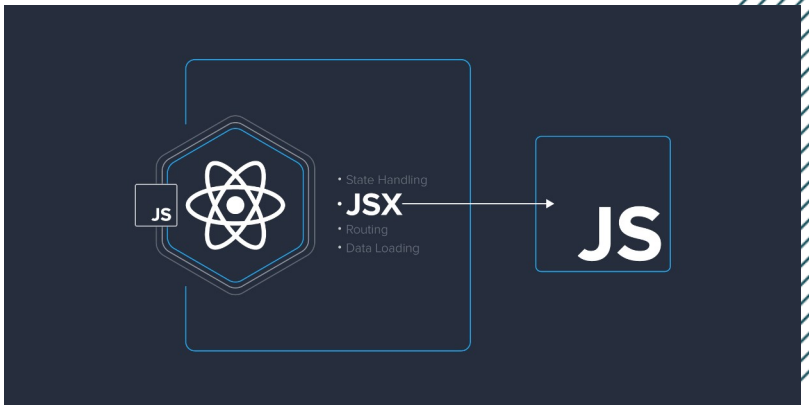
- JSX is an XML/HTML-like syntax used by React
- By using JSX you can write concise HTML/XML-like structures
- Babel will transform these expressions into actual JavaScript code

```
const nav = (
    <ul id="nav">
      <li><a href="#">Home</a></li>
      <li><a href="#">About</a></li>
      <li><a href="#">Clients</a></li>
      <li><a href="#">Contact Us</a></li>
    </ul>
);
```

# JSX attributes

- Since JSX is closer to JavaScript than to HTML, React DOM uses *camelCase* property naming convention instead of HTML attribute names
- For example, **class** becomes **className** in JSX, and **tabindex** becomes **tabIndex**:

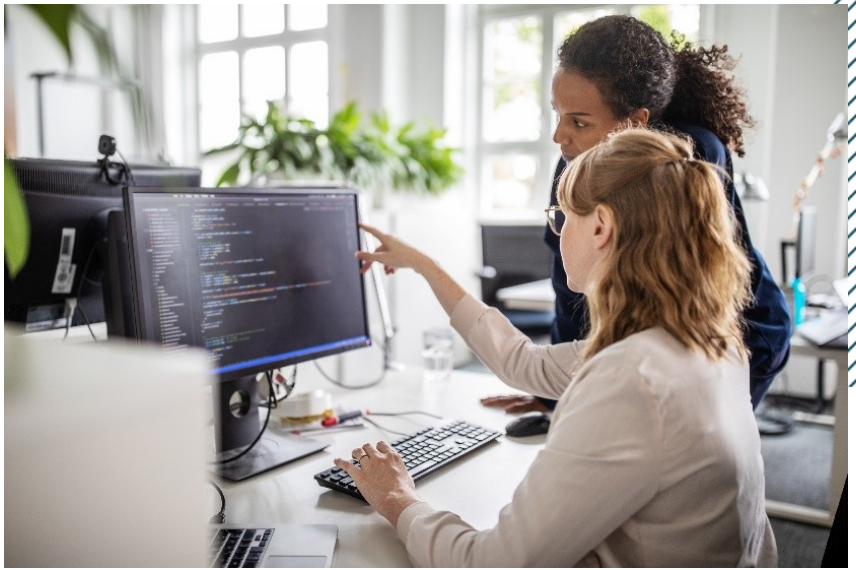
```
<div
  className="collapse"
>
```



# How to create a React app

https://vitejs.dev/guide/

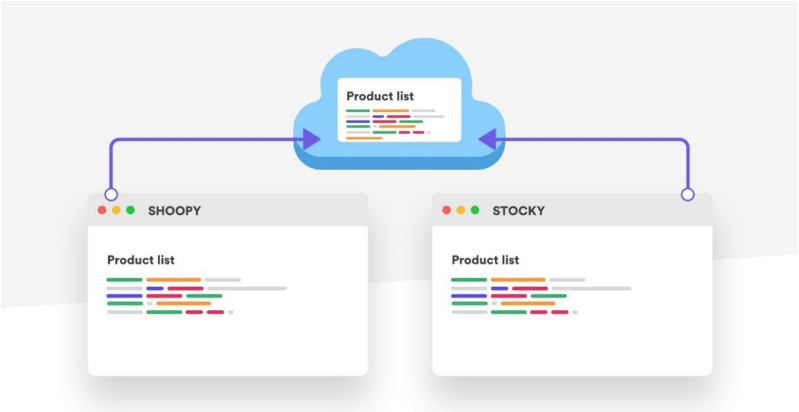
```
npm create vite@latest
cd projekto-pavadinimas
npm install
npm run dev
```



TECHIN

# What is React component?

- Components let you split the UI into independent, reusable pieces, and think about each piece in isolation








TECHIN

# Functional Component

- The simplest way to define a component is to write a JavaScript function
- This function is a valid React component because it accepts a single *props* (which stands for properties) object argument with data and returns a React element

```
export default function Test(props) {
  return <h1>Hello, {props.name}</h1>;
}
```

# Functional component

# Functional Component

```
import Test from
"./components/Test";

function App() {
  return (
    <div>
      <Test name="React" />
    </div>
  );
}

export default App;
```

```
export default function Test(props) {
  return <h1>Hello, {props.name}</h1>;
}
```

# Functional component | `props`

```
import Test from
"./components/Test";

function App() {
  return (
    <div>
      <Test name="React" />
      <Test name="JavaScript" />
      <Test name="JSX" />
    </div>
  );
}

export default App;
```

```
export default function Test(props) {
  return <h1>Hello, {props.name}</h1>;
}

/* Browser output, based on argument
passed:
    Hello, React
    Hello, JavaScript
    Hello, JSX
*/
```

# Functional component | `props`

```
import Test from "./components/Test";

function App() {
  return (
    <div>
      <Test name="React" />
      <Test name="JavaScript" />
      <Test name="JSX" />
    </div>
  );
}

export default App;
```

```
export default function Test(props) {
  console.log(props);
  return <h1>Hello, {props.name}</h1>;
}
```

```
▼Object ℹ
    name: "React"
  ▶ __proto__: Object
▼Object ℹ
    name: "JavaScript"
  ▶ __proto__: Object
▼Object ℹ
    name: "JSX"
  ▶ __proto__: Object
```

# Functional component | props

```
import Test from "./components/Test";


function App() {
  return (
    <div>
      <Test
        name="React"
        content="JS library for building UI."
      />
      <Test
        name="JavaScript"
        content="is a programming language."
      />
      <Test
        name="JSX"
        content="allows us to write HTML in React."
      />
    </div>
  );
}


export default App;
```

```
export default function Test(props) {
  console.log(props);
  return (
    <h1>
      {props.name} - {props.content}
    </h1>
  );
}



// Browser output:
// React - JS library for building UI.
// JavaScript - is a programming language.
// JSX - allows us to write HTML in React.
```

```
▶ {name: "React", content: "JS library for building UI."}
▶ {name: "JavaScript", content: "is a programming language."}
▶ {name: "JSX", content: "allows us to write HTML in React."}
```

# Functional component | destructuring assignment

```
import Test from "./components/Test";


function App() {
  return (
    <div>
      <Test
        name="React"
        content="JS library for building UI."
      />
      <Test
        name="JavaScript"
        content="is a programming language."
      />
      <Test
        name="JSX"
        content="allows us to write HTML in React."
      />
    </div>
  );
}


export default App;
```

```
function Test({ name, content }) {
  return (
    <h1>
      {name} - {content}
    </h1>
  );
}

export default Test;
```

# Kaip pavadinti komponentus? PascalCase

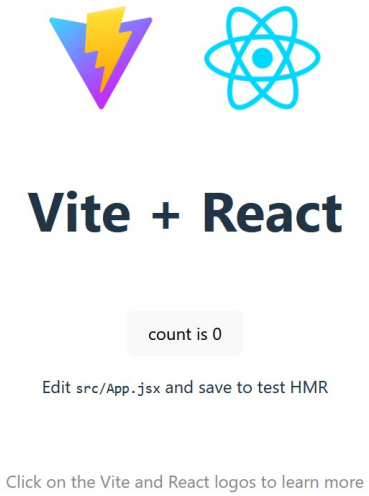
## Naming

- **Extensions**: Use `.jsx` extension for React components.
- **Filename**: Use PascalCase for filenames. E.g., `ReservationCard.jsx`.
- **Reference Naming**: Use PascalCase for React components and camelCase for their instances. eslint: `react/jsx-pascal-case`

```
// bad
import reservationCard from './ReservationCard';

// good
import ReservationCard from './ReservationCard';

// bad
const ReservationItem = <ReservationCard />;

// good
const reservationItem = <ReservationCard />;
```

# Praktika (1)

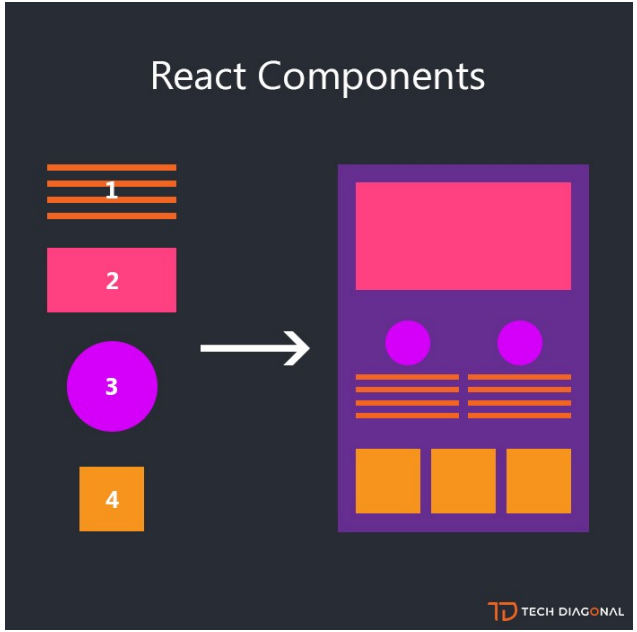
- Įdiegti Node.js
- Paleisti pirmą React aplikaciją



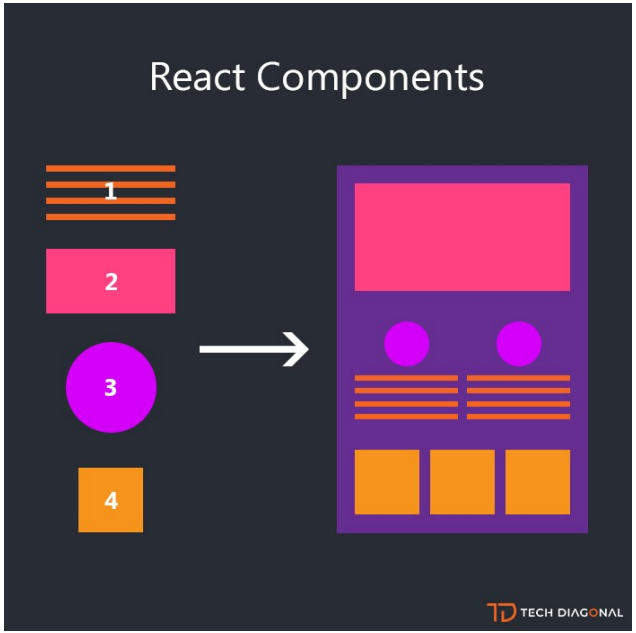
# Praktika (2)

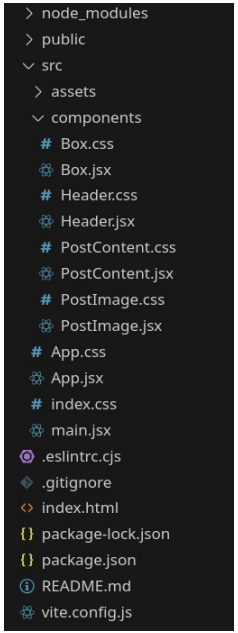
Sukurti 4 funkcinius komponentus

1. PostContent: antraštė `h3` ir pastraipa
2. Header: antraštė `h1`, nuotrauka
3. PostImage: nuotrauka
4. Box: blokas `div`, aukštis, plotis, spalva



# Praktika (2) – tęsinys toliau




Failų struktūra

# Praktika (2) | Header.jsx – tęsinys toliau

```jsx
import "./Header.css";

export default function Header() {
  return (
    <div className="header vh-100">
      <h1 className="header__title">Page title</h1>
      <img
        src="image-url"
        alt="Fill here"
        className="header__img w-100 h-25 object-fit-cover"
      />
    </div>
  );
}

```

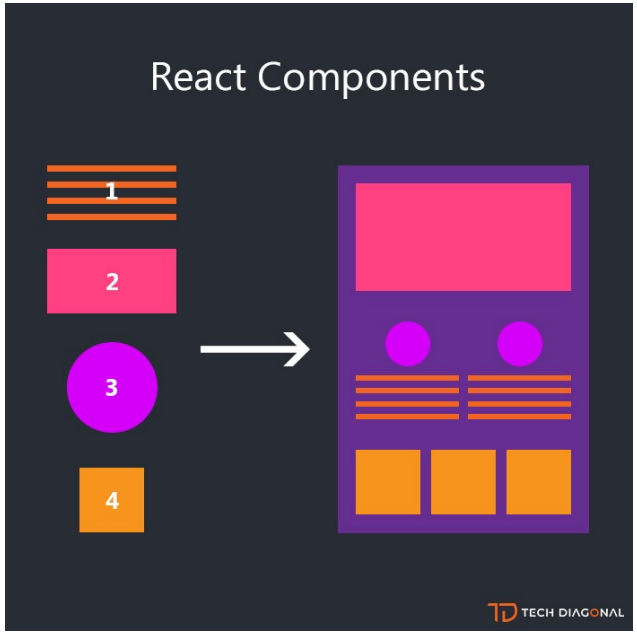
# Praktika (2) | Reikia Bootstrap? - tęsinys toliau

1. https://getbootstrap.com/docs/5.3/getting-started/download/#npm
   Terminale:
   npm install bootstrap
2. Ar atsirado package.json faile?
   "bootstrap": "^X.X.X"
3. O kaip dabar panaudoti?
   Naudojame `import "bootstrap/dist/css/bootstrap.css";`
4. O kur įdėti?
   main.jsx

# Praktika (2) | Klijuojam - tęsinys toliau

```
import Header from "./components/Header";
import PostImage from "./components/PostImage";
import PostContent from "./components/PostContent";
import Box from "./components/Box";

function App() {
  return (
    <div class="container">
      <div class="row">
        <div class="col">
          <Header />
        </div>
      </div>

      <div class="row">
        <div class="col-6">
          <PostImage />
          <PostContent />
        </div>

        <div class="col-6">
          <PostImage />
          <PostContent />
        </div>
      </div>

      <div class="row">
        <div class="col-3">
          <Box />
        </div>
        <div class="col-3">
          <Box />
        </div>
        <div class="col-3">
          <Box />
        </div>
        <div class="col-3">
          <Box />
        </div>
      </div>
    </div>
  );
}

export default App;
```

# Praktika (2) | Rezultatas

# Klaidos

KLAIDELĖ: Adjacent JSX elements must be wrapped in an enclosing tag. Did you want a JSX fragment <>...</>?

Kaip išspręsti?

Funkcijos return dalyje viską reikia apgaubti į vieną *bloką*:

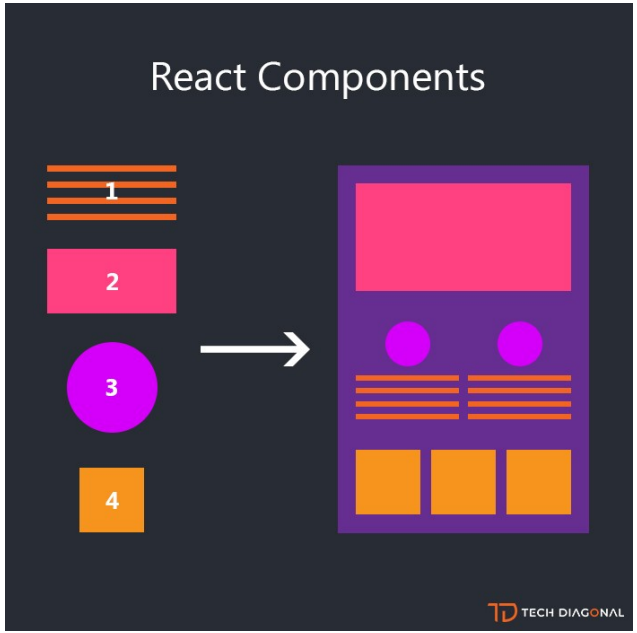
- <>...</>
- ...
- <React.Fragment>...</React.Fragment>
- Daugiau

```
6  function App() {
7    return (
8      <>
9        <Header />
10       <PostImage />
11       <PostContent />
12       <Box />
13     </>
```

# Praktika (3) – tęsinys toliau

Turimiems komponentams, perduoti informaciją per *props*.

1. PostContent: antraštės h3 tekstas ir pastraipos tekstas
2. Header: antraštės h1 tekstas, nuotrauka šaltinis (src)
3. PostImage: nuotraukos šaltinis (src)
4. Box: spalva



# Praktika (3) | App.jsx ir Header.jsx – tęsinys toliau

```jsx
import Header from "./components/Header";
import PostImage from "./components/PostImage";
import PostContent from "./components/PostContent";
import Box from "./components/Box";

function App() {
  return (
    <div className="container">
      <div className="row">
        <div className="col">
          <Header
            title="Labas, aš mokausi"
            content="Man patinka React"
          />
        </div>
      </div>
      {/* ... */}
```

```jsx
import "./Header.css";

export default function Header(props) {
  console.log(props); // Kas ten?

  return (
    <div className="header vh-100">
      <h1 className="header__title">{props.title}</h1>
      <img
        src="img-url"
        alt="Fill here"
        className="header__img w-100 h-25 object-fit-cover"
      />
      <p>{props.content}</p>
    </div>
  );
}
```

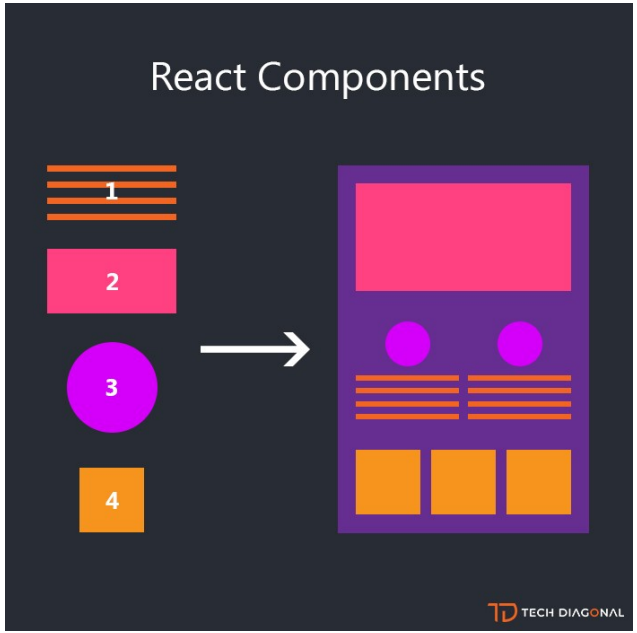
# src={require(.../images/...)}

- Nuotraukos įterpimui lokaliai, galime naudoti `<img src={require('./images/react-logo.png')} alt="React Logo" />`

# Praktika (4) – tęsinys toliau

O jeigu PostContent komponentų yra daugiau, pavyzdžiui trys?

1. Sukurti masyvą, kuris saugo kelis objektus su informacija: title ir content, o gal dar image?



# Praktika (4) PostsList.jsx ir PostContent.jsx – tęsinys toliau

src > components > PostsList.js > PostsList > posts

```javascript
import React from "react";
import PostContent from "./PostContent";

function PostsList() {
  let posts = [
    {
      title: "HTML",
      content: "Lorem ipsum HTML",
      img: "https://picsum.photos/id/123/200/200",
    },
    {
      title: "CSS",
      content: "Lorem ipsum CSS",
      img: "https://picsum.photos/id/237/200/200",
    },
    {
      title: "JavaScript",
      content: "Lorem ipsum JavaScript",
      img: "https://picsum.photos/id/222/200/200",
    },
  ];

  let list = posts.map((post) => {
    return (
      <PostContent title={post.title} content={post.content} img={post.img} />
    );
  });

  return <div>{list}</div>;
}

export default PostsList;
```

src > components > PostContent.js > default

```javascript
import React from 'react'

function PostContent(props) {
    let {title, content, img} = props;
  return (
    <div>
        <h3>{title}</h3>
        <img src={img} alt={title} />
        <p>{content}</p>
    </div>
  )
}

export default PostContent
```



Justina Blasė, Julius Zabulėnas

TECHIN

# Praktika (4) App.js – tęsinys toliau

```
1  import PostsList from "./components/PostsList";
2
3  function App() {
4    return (
5      <div>
6        <PostsList />
7      </div>
8    );
9  }
10
11 export default App;
```

# Praktika (4) Ar konsolę stebim? - tęsinys toliau

```
Warning: Each child in a list should have a unique "key" prop.        react-jsx-dev-runtime.development.js:117

Check the render method of `PostsList`. See https://reactjs.org/link/warning-keys for more information.
    at PostContent (http://localhost:3000/static/js/bundle.js:212:5)
    at PostsList
    at div
    at div
    at div
    at App
```

# Praktika (4) - Darom tinkamiau – tęsinys toliau

## Quickstart

To create a random UUID...

**1. Install**

```
npm install uuid
```

**2. Create a UUID** (ES6 module syntax)

```
import { v4 as uuidv4 } from 'uuid';
uuidv4(); // ⇨ '9b1deb4d-3b7d-4bad-9bdd-2b0d7b3dcb6d'
```

... or using CommonJS syntax:

```
const { v4: uuidv4 } = require('uuid');
uuidv4(); // ⇨ '1b9d6bcd-bbfd-4b2d-9b5d-ab8dfbbd4bed'
```

# Praktika (4) – Kur įdėti?

```
1  import PostContent from "./PostContent";
2  import { v4 as uuidv4 } from "uuid";
3
4  export default function PostsList() {
5    const posts = [
6      {
7        title: "HTML",
8        content: "Lorem ipsum HTML",
9        img: "https://picsum.photos/id/123/200/200",
10     },
11     {
12       title: "CSS",
13       content: "Lorem ipsum CSS",
14       img: "https://picsum.photos/id/237/200/200",
15     },
16     {
17       title: "JavaScript",
18       content: "Lorem ipsum JavaScript",
19       img: "https://picsum.photos/id/222/200/200",
20     },
21   ];
22
23   const list = posts.map((post) => {
24     return (
25       <PostContent
26         key={uuidv4()}
27         title={post.title}
28         content={post.content}
29         img={post.img}
30       />
31     );
32   });
33
34   return <div>{list}</div>;
35 }
```