

★ Get unlimited access to the best of Medium for less than \$1/week. [Become a member](#)



12 React Best Practices and Security

React security best practices to build high-performing apps



XongoLab Technologies · [Follow](#)

Published in FAUN—Developer Community · 5 min read · Aug 10, 2023



16



In the ever-expanding landscape of digital innovation, React has emerged as a formidable force in crafting dynamic and engaging web applications.

With its component-based architecture and virtual DOM, React empowers developers to create seamless user experiences while maintaining optimal performance.

However, it is important to follow security best practices when **developing React applications** to protect them from attacks.

Here are 12 React security best practices:

1. Use default XSS protection with data binding

React automatically escapes values in data bindings, which helps to protect against cross-site scripting (XSS) attacks. For example, the following code will safely render the value of the `username` prop, even if it contains malicious code:

```
const MyComponent = ({ username }) => {  
  return <h1>Hello, {username}</h1>;  
};
```

2. Watch out for dangerous URLs and URL-based script injection

Never render untrusted URLs directly into the DOM. Instead, use a library like DOMPurify to sanitize the URL before rendering it. For example, the following code will safely render the URL of the `image` prop, even if it contains malicious code:

```
const MyComponent = ({ image }) => {  
  const sanitizedImageUrl = DOMPurify.sanitize(image);  
  return <img src={sanitizedImageUrl} />;  
};
```

3. Sanitize and render HTML

If you need to render HTML in your React application, be sure to sanitize it first to remove any malicious code. You can use a library like DOMPurify to do this. For example, the following code will safely render the HTML of the `content` prop, even if it contains malicious code:

```
const MyComponent = ({ content }) => {  
  const sanitizedContent = DOMPurify.sanitize(content);  
  return <div dangerouslySetInnerHTML={sanitizedContent} />;  
};
```

4. Avoid direct DOM access

Whenever possible, avoid accessing the DOM directly from your React code. This can help to prevent XSS attacks and other security vulnerabilities. For example, instead of using the following code to add a new element to the DOM:

```
const MyComponent = ({ name }) => {  
  const element = document.createElement('div');  
  element.textContent = name;  
  document.body.appendChild(element);  
};
```

You can use the following code, which uses React's `createElement()` method to create a new element and then safely inserts it into the DOM:

```
const MyComponent = ({ name }) => {  
  const element = React.createElement('div', { textContent: name });  
  return element;  
};
```

5. Secure React server-side rendering

If you are using server-side rendering with React, be sure to secure it properly. This includes using HTTPS and properly sanitizing all input data. For example, if you are using a library like Express to render your React application server-side, you can use the `helmet` middleware to secure your application. The `helmet` middleware will automatically configure a number of security features, such as HTTPS, Content Security Policy, and X-Frame-Options.

6. Check for known vulnerabilities in dependencies

React and its dependencies are constantly being updated to fix security vulnerabilities. Be sure to regularly check for known vulnerabilities in your dependencies and update them as needed. You can use a tool like `npm audit` to check for known vulnerabilities in your dependencies.

7. Avoid JSON injection attacks

When sending JSON data to the client, be sure to escape all special characters. This will help to prevent JSON injection attacks. For example, the following code will safely send the JSON data of the `user` prop to the client, even if it contains malicious code:

```
const user = {
  name: 'John Doe',
  email: 'johndoe@example.com',
};

const json = JSON.stringify(user, null, 2);
const escapedJson = json.replace(/</g, '\\u003C');

const response = {
  data: escapedJson,
};

return response;
```

8. Use non-vulnerable versions of React

Only use the latest, non-vulnerable versions of React. You can check the React security advisories page to see if there are any known security vulnerabilities in your version of React.

9. Use linter configurations

A linter can help you to find potential security vulnerabilities in your React code. There are a number of linters that are specifically designed for React, such as ESLint and TSLint. For example, **ESLint** has a number of security rules that can help you to find potential security vulnerabilities in your React code.

10. Avoid dangerous library code

There are a number of libraries that contain dangerous code that can be exploited by attackers. Be sure to research any libraries that you use to make sure that they are safe.

11. Use secure cookies

When using cookies in your React application, be sure to use secure cookies. Secure cookies are only sent over HTTPS, which helps to protect them from

being intercepted by attackers. You can set the `secure` property of a cookie to `true` to make it secure.

12. Encrypt sensitive data

If you need to store sensitive data in your React application, be sure to encrypt it. This will help to protect it from being accessed by unauthorized users. You can use a library like `bcryptjs` to encrypt sensitive data.

In addition to these best practices, there are a number of other things that you can do to improve the security of your React applications. For example, you can:

- *Use a Content Security Policy (CSP) to restrict the resources that your application can load from the web.*
- *Use a Web Application Firewall (WAF) to block malicious traffic from reaching your application.*
- *Implement a user authentication and authorization system to control who has access to your application.*
- *Monitor your application for suspicious activity.*

Wrapping up

React is a powerful JavaScript library for building user interfaces. However, it is important to follow security best practices when developing React applications to protect them from attacks. By following the 12 React security best practices outlined in this article, you can help to keep your React applications safe from attack and protect your users' data.



👏 If you find this helpful, please click the clap 👏 button below a few times to show your support for the author 🙌

🚀 **Join FAUN Developer Community & Get Similar Stories in your Inbox Each Week**

React

Reactjs

Reactjs Development

React Best Practices

Web App Development



Written by XongoLab Technologies

Follow

226 Followers · Writer for FAUN—Developer Community 🐾

Trusted web & mobile app development company serving clients across the globe since 2011. Working with AI, DevOps, ML & IoT. <https://www.xongolab.com/>

More from XongoLab Technologies and FAUN—Developer Community 🐾



 XongoLab Technologies in Frontend Weekly

Top Frontend Frameworks for Web Development

The ultimate guide to choosing the right Frontend framework for your next project!

Jun 28



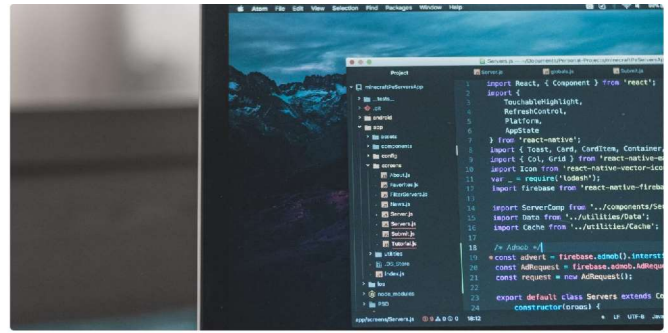
Jul 11


 1.1K

 9



Open in app ↗



 Dylan Cooper in FAUN—Developer Community

Essential for Lazy Developers: Five Efficient Python Decorators

Decorators to make you a better Pythonista.

Medium



Search



Write



 Rak Siv in FAUN—Developer Community

Nitric is Terraform for Developers

As of the 29th of July this year, Terraform has been around for an entire decade! That's right...

Aug 7

 61

 1



 XongoLab Technologies in Nerd For Tech

How To Build An API: A Complete Guide

A comprehensive guide about API development that includes its types, tools,...

Aug 26, 2022

 27



See all from XongoLab Technologies

See all from FAUN—Developer Community

Recommended from Medium

AHMOUL.COM
Software Development Engineer
Seattle, WA
Mar. 2020 – May 2021

- Developed Amazon checkout and payment services to handle traffic of 10 Million daily global transactions
- Integrated Iframes for credit cards and bank accounts to secure 80% of all consumer traffic and prevent CSRF, cross-site scripting, and cookie-jacking
- Led Your Transactions implementation for JavaScript front-end framework to showcase consumer transactions and reduce call center costs by \$25 Million
- Recovered Saudi Arabia checkout failure impacting 4000+ customers due to incorrect GET form redirection

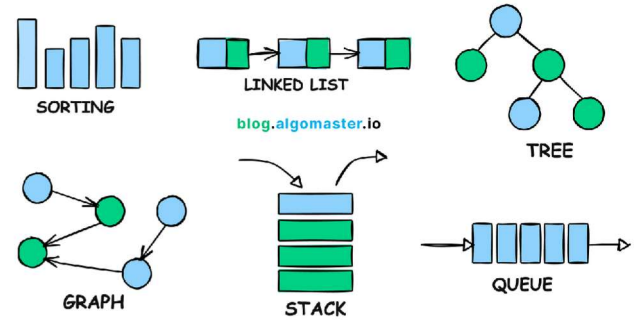
Projects

NinjaPrep.io (React)

- Platform to offer coding problem practice with built in code editor and written + video solutions in React
- Utilized Nginx to reverse proxy IP address on Digital Ocean hosts
- Developed using Styled-Components for 95% CSS styling to ensure proper CSS scoping
- Implemented Docker with Seccomp to safely run user submitted code with < 2.2s runtime

HeatMap (JavaScript)

- Visualized Google Takeout location data of location history using Google Maps API and Google Maps heatmap code with React
- Included local file system storage to reliably handle 5mb of location history data
- Implemented Express to include routing between pages and jQuery to parse Google Map and implement heatmap overlay



Alexander Nguyen in Level Up Coding

The resume that got a software engineer a \$300,000 job at Google.

1-page. Well-formatted.



Jun 1



19.93K



359



Ashish Pratap Singh in AlgoMaster.io

How I Mastered Data Structures and Algorithms

Getting good at Data Structures and Algorithms (DSA) helped me clear interview...



Jul 23



756



6



Lists



Stories to Help You Grow as a Software Developer

19 stories · 1319 saves



General Coding Knowledge

20 stories · 1536 saves



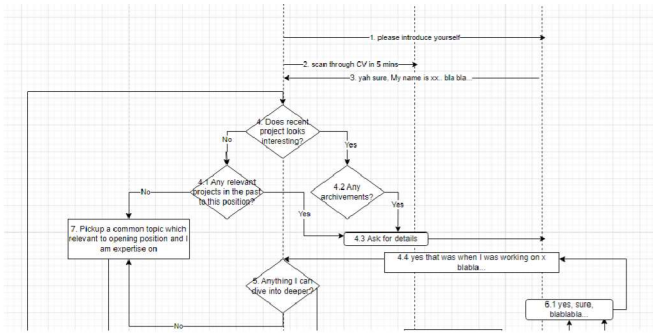
Medium's Huge List of Publications Accepting...

334 stories · 3436 saves



Natural Language Processing

1670 stories · 1250 saves



LORY

Today I Interviewed for a Lead Front-End Role

And They Asked Me a Couple of Tough Questions

Aug 3 731 12

```
console.log('start');

const promise1 = new Promise((resolve, reject) => {
  console.log(1)
  resolve(2)
})

promise1.then(res => {
  console.log(res)
})

console.log('end');
```

Shuai Li in Programming Domain

Can You Answer This Senior Level JavaScript Promise Interview...

Most interviewees failed on it.

Aug 12 1.5K 16

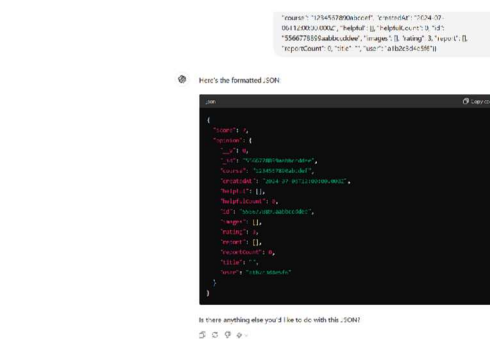


Maryum Khan

Implementing Efficient Role-Based Access Control in React with Redux

Role-Based Access Control (RBAC) is a security paradigm essential for modern web...

Apr 8 170 3



Suman Sourabh

How I Use ChatGPT as a Frontend Developer (5 Ways)

With ChatGPT chat links and screenshots and final result

Jul 6 1K 30

See more recommendations