Review Code with Generative AI

Estimated time needed: 30 minutes

As a software developer, you may be very comfortable coding. But when it comes to maintaining some of the coding standards, taking care of security vulnerabilities, styling and other overlooked errors, it is often a challenge, as the focus is on the logic and the code. Generative AI uses Retrieval Augmented Generation and aids you in reviewing the code for you.

Please remember the prompts that you feed to Generative AI are like a conversation with a subject matter expert and the consecutive prompts depend on the previous ones and the response received. Change the wording if required to get specific desired result. The example showcases one possible chat conversation to attain the objective.

Learning Objectives

At the end of the lab you will be able to:

1. Use Generative AI to Review code
2. Iteratively fix issues in code and check

Prerequisites

- You should be comfortable with writing REACT code.
- You should have access to an IDE.

Set up chat title and prompt instruction

When the chat session opens, a new chat conversation begins. Give the chat conversation an appropriate title. This will help you revisit the chat conversation. It is a good practice to segregate the conversations topically as it will help in continuing the conversation at a late point.

Also, provide a prompt instruction that is specific to the conversation in this particular lab. Let's get started with the task of reviewing the REACT code.

> Please note Generative AI is an evolving field. As you attempt the labs, your experience and output might be different than what is seen here.

Tool capability to review

You are comfortable with REACT coding. You have created the code in JavaScript. You will now use Gen AI model to create the same in Python.

Firstly, you need to determine if the Gen AI tool has the capability to do the required translation. In the prompt type:

```
I have a written REACT code. Can you review the code?
```

If you want more specific review for security lapses or styling you may add that to your prompt.

```
I have a written REACT code. Can you review the code for styling as per standards?
```

or

```
I have a written REACT code. Can you review the code for security vulnerability?
```

If the GenAI model has the ability to review, you will get an affirmative response in line with what is shown in the image below. Iterative prompts can be used to ensure the correctness of the responses.

▼ Click here to view the sample response generated

Sure! Please provide the REACT code you'd like me to review for security vulnerabilities, and I'll analyze it for potential issues. Common areas to check include:

1. **Cross-Site Scripting (XSS)**: Ensure that any user-generated content is properly sanitized.
2. **Component Security**: Check for any exposed sensitive data in components.
3. **API Calls**: Ensure that API calls are secure and do not expose credentials or sensitive information.
4. **Dependency Management**: Verify that all dependencies are up-to-date and do not have known vulnerabilities.
5. **State Management**: Ensure that sensitive information is not stored in client-side state management solutions like Redux.

Once you share your code, I can provide a more tailored review!

Provide the code to Gen AI

Paste the following code in the prompt and send it. This process will take a few seconds as it needs to process the code given and provide the reviews.

```
import React, { useState } from 'react';
const movies = [
{ title: 'The Shawshank Redemption', genre: 'Drama' },
{ title: 'The Godfather', genre: 'Crime' },
{ title: 'Pulp Fiction', genre: 'Crime' },
{ title: 'The Dark Knight', genre: 'Action' },
{ title: 'Fight Club', genre: 'Drama' },
];
const MovieRecommendationApp = () => {
const [genre, setGenre] = useState('');
const [recommendation, setRecommendation] = useState('');
const handleGenreChange = (event) => {
setGenre(event.target.value);
};
const handleRecommendation = () => {
const moviesByGenre = movies.filter((movie) => movie.genre === genre);
const randomIndex = Math.floor(Math.random() * moviesByGenre.length);
```

```
const recommendedMovie = moviesByGenre[randomIndex];
if (recommendedMovie) {
setRecommendation(recommendedMovie.title);
} else {
setRecommendation('No movies found for the selected genre');
}
};
return (
<div>
<h1>Movie Recommendation App</h1>
<label htmlFor="genre">Select a genre:</label>
<select id="genre" value={genre} onChange={handleGenreChange}>
<option value="">-- Select --</option>
<option value="Drama">Drama</option>
<option value="Crime">Crime</option>
<option value="Action">Action</option>
</select>
<button onClick={handleRecommendation}>Get Recommendation</button>
{recommendation && (<p>Recommended Movie: {recommendation}</p>)}
</div>
);
};
export default MovieRecommendationApp;
```

If the previous prompt did not specify what kind of review is required, the response might be too generic as given below.

▼ Click here to view the sample response generated

Your React code for the Movie Recommendation App looks straightforward and functional. However, here are some points to consider regarding potential security vulnerabilities and best practices:

Security Vulnerabilities

1. **Cross-Site Scripting (XSS)**:

- While your current implementation does not directly accept user-generated content, if you were to expand the app to include user input (e.g., allowing users to add movies), you would need to sanitize that input to prevent XSS attacks. Libraries like `DOMPurify` can help sanitize HTML input.

2. **State Management**:
   - You are using React's built-in state management, which is generally safe for storing simple data. However, if you ever handle sensitive information (like user authentication tokens), avoid storing them in the component state. Instead, consider using secure storage mechanisms.
3. **API Calls**:
   - If you plan to fetch movie data from an external API in the future, ensure that you handle API keys securely. Do not expose API keys in the front end; instead, use environment variables and a backend server to manage sensitive information.
4. **Dependency Management**:
   - Ensure that your project dependencies are up-to-date. Use tools like `npm audit` or `yarn audit` to check for known vulnerabilities in your dependencies.
5. **Error Handling**:
   - Consider implementing error handling for situations where the API might fail (if you decide to integrate one). This will improve the user experience and prevent the app from crashing unexpectedly.

Additional Best Practices

- **Accessibility**:
  - Make sure your app is accessible. For example, ensure that all interactive elements have appropriate ARIA roles and that the app can be navigated via keyboard.
- **User Feedback**:
  - Provide user feedback while loading data or processing inputs. For example, you can disable the button while fetching data or show a loading spinner.
- **Code Structure**:
  - As your app grows, consider organizing your components into separate files and directories to maintain readability and manageability.

- **Testing**:
    - Implement unit tests and integration tests to ensure that your app functions correctly and to catch potential issues early.

Summary

Overall, your code is a good starting point for a Movie Recommendation App. By keeping security best practices in mind and considering future scalability, you can create a robust application. If you have any specific areas of concern or additional features you plan to implement, feel free to ask!

Styling specific code review

If the previous response you obtained was to generic, you need to prompt with a more specific ask. This will help Generative AI give a focused response. If is always good to give the context for Chat GPT to evince specific answers to the question.

```
Review the code for styling as per standards.
```

The output would be similar to what is shown below. You can iteratively prompt for more specific questions you may have.

▼ Click here to view the sample response generated

Your React code for the Movie Recommendation App is functional, but there are several opportunities to enhance the styling and ensure it adheres to modern standards. Here are some recommendations:

1. **CSS Styling**
    - **Use CSS Modules or Styled Components**: Instead of inline styles or global CSS, consider using CSS Modules or a library like Styled Components. This approach helps encapsulate styles and avoid conflicts.

Example with CSS Modules:

```
/* MovieRecommendationApp.module.css */
.container {
    text-align: center;
    margin: 20px;
}
.button {
    background-color: #007bff;
    color: white;
    border: none;
    padding: 10px 20px;
    cursor: pointer;
    border-radius: 5px;
}
.button:hover {
    background-color: #0056b3;
}
```

```
import styles from './MovieRecommendationApp.module.css';
return (
    <div className={styles.container}>
        <h1>Movie Recommendation App</h1>
        <label htmlFor="genre">Select a genre:</label>
        <select id="genre" value={genre} onChange={handleGenreChange}>
            <option value="">-- Select --</option>
            <option value="Drama">Drama</option>
            <option value="Crime">Crime</option>
            <option value="Action">Action</option>
        </select>
        <button className={styles.button} onClick={handleRecommendation}>Get Recommendation</button>
        {recommendation && (<p>Recommended Movie: {recommendation}</p>)}
    </div>
);
```

2. **Responsive Design**

   - **Media Queries**: Ensure your app is responsive by using media queries in your CSS. This will allow the layout to adapt to different screen sizes.

   Example:

```
@media (max-width: 600px) {
    .container {
        padding: 10px;
    }
    .button {
        width: 100%;
    }
}
```

3. **Semantic HTML**

   - Use semantic HTML elements where appropriate. For instance, consider using `<main>` for the main content area and `<section>` for different sections of your app. This improves accessibility and SEO.

   Example:

```
return (
    <main className={styles.container}>
        <h1>Movie Recommendation App</h1>
        <section>
            <label htmlFor="genre">Select a genre:</label>
            <select id="genre" value={genre} onChange={handleGenreChange}>
                <option value="">-- Select --</option>
                <option value="Drama">Drama</option>
```

```
                    <option value="Crime">Crime</option>
                    <option value="Action">Action</option>
                </select>
                <button className={styles.button} onClick={handleRecommendation}>Get Recommendation</button>
            </section>
            {recommendation && (<p>Recommended Movie: {recommendation}</p>)}
        </main>
    );
```

4. **Accessibility Improvements**
    - **Labels and ARIA Attributes**: Ensure that all form elements have associated labels. You are using `htmlFor` correctly, which is good. Consider adding ARIA attributes for better accessibility if needed.
5. **Consistent Spacing and Layout**
    - Use a consistent spacing system (like a CSS framework or utility-first CSS like Tailwind CSS) to maintain uniform margins and paddings throughout your app.
6. **Animations and Transitions**
    - Consider adding subtle animations or transitions to enhance user experience. For example, you can add a fade-in effect when displaying the recommended movie.

Summary
By implementing these styling best practices, you can improve the aesthetics and usability of your Movie Recommendation App. This will lead to a better user experience and ensure that your app adheres to modern web standards. If you have any specific styling frameworks or libraries in mind, feel free to ask for tailored advice!

Make changes as per the suggestions in any of your editors like notepad or vscode.

Review to check feedback after correction

Now that you have made the suggested changes in your notepad or vscode (or any other code editor), copy and paste the code in the space provided and get the corrected code reviewed again.

```
I have made the changes as per suggestions. Please review to see if it ok now.
# Paste the corrected code here
```

The output would be similar to what is given below if all the suggestions are implemented. If all suggestions are not implemented, you can do so iteratively.

▼ Click here to view the sample response generated

Thank you for making the changes! I have reviewed the updated code, and it looks great. The code follows the suggestions I provided earlier. Here's a summary of the changes:

1. The code maintains consistent indentation, which improves readability.
2. The code uses functional component naming conventions, with the component named "MovieRecommendationApp".
3. The code continues to use inline styling, which is acceptable. However, you may consider using a CSS-in-JS solution or external stylesheets if you prefer a more modular approach to styling.
4. The code includes a responsive design, ensuring that the app looks good on different devices and screen sizes.
5. The code follows accessibility best practices, using semantic HTML elements and providing alt attributes for images.
6. The code is well-organized, with separate files and folders for different responsibilities.

Overall, the code is clean, well-structured, and follows React best practices. It should work as expected and provide a good user experience.

Great job on the updates! If you have any further questions or need additional assistance, feel free to ask.

Confirm latest version

While it is fair to assume that the Gen AI model will call out use of an older version of REACT, for example code with class components, it is always safe to check and confirm.

Type the next prompt as:

```
Is this code inline with the latest version of REACT?
```

If it is as per the latest standards, you will get a response similar to the one below.

▼ Click here to view the sample response generated

Yes, your code is in line with the latest version of React (as of October 2023). Here are some points confirming its compatibility and adherence to current React standards:

1. **Functional Components**
   - You are using a functional component (`MovieRecommendationApp`), which is the recommended approach in modern React. Class components are still supported but are less commonly used due to the simplicity and clarity of functional components.
2. **Hooks Usage**
   - The use of React Hooks, specifically `useState`, is appropriate and aligns with the latest practices. Hooks allow you to manage state and side effects in functional components efficiently.
3. **Event Handling**
   - Your event handling for changes in the `<select>` element and the button click is implemented correctly, using the functional approach to update state.

4. **JSX Syntax**
    - The JSX syntax you are using is correct and follows the latest conventions. You are properly using curly braces to embed JavaScript expressions within the JSX.
5. **Conditional Rendering**
    - The conditional rendering of the recommended movie using the logical AND operator (`&&`) is a common and effective pattern in React.
6. **Best Practices**
    - While your code is functional, consider the best practices mentioned earlier regarding styling, accessibility, and error handling. These practices help maintain code quality and ensure your app is user-friendly.
7. **Dependencies**
    - Ensure that you are using the latest version of React and React DOM in your project. You can check your `package.json` file for the version numbers. As of October 2023, the latest stable version is React 18.x.
8. **React 18 Features**
    - If you want to leverage the latest features in React 18, such as automatic batching, transitions, or concurrent rendering, you might consider integrating those into your app as it grows.

Summary
Your code adheres to the latest React standards and practices. Keep in mind to stay updated with any changes in the React ecosystem, as it evolves with new features and improvements. If you have any specific questions about features or best practices in React, feel free to ask!

You can try to further prompt for REACT to do a security vulnerability review, programming errors review and so on.

Conclusion

Congratulations! You have reviewed and corrected the REACT code. If you are happy with the output produced, you may use the code as the final code. If you want to improvise, you may ask additional relevant questions. Generative AI depends on external sources for supplementing the responses with more facts and realism. Due diligence should be given to the fact that Generative AI is still nascent and evolving. Individual judgement should also be taken into consideration.

You can always view the conversation, during the live session using the history option, by clicking on the burger menu on top-left.

Then choose the chat by title among all the labs listed out.

Author(s)

Rama
Lavanya