Implementation Steps

1. Locate the `analyzeText` function in the `App.js` component:

```
javascript
const analyzeText = async (text) => {
  setLoading(true);
  try {
    // API call would happen here
    // In a real implementation, this would be a fetch to your backend
    const response = await fetch(`/check/${encodeURIComponent(text)}/`, {
      method: 'POST',
    });
    const data = await response.json();
    setResult(data);
  } catch (error) {
    console.error('Error analyzing text:', error);
  } finally {
    setLoading(false);
  }
};
```

2. Replace any calls to the `simulateAnalysis` function with calls to this `analyzeText` function.

Expected API Response Format

Your backend endpoint must return a JSON response with the following structure:

```
json
{
  "prediction": "sql",  // One of: "sql", "xss", or "safe"
  "confidence": "0.95", // Confidence score as a string between 0-1
  "probabilities": "{\"sql\":0.95,\"xss\":0.03,\"safe\":0.02}"  // JSON string
with class probabilities
}
```

Component Interaction

- The `InputSection` component calls the analysis function when the user submits text
- The `App` component manages loading state and stores the result
- The `ResultsDisplay` component visualizes the analysis results

No other code changes are required to make the text analysis section work properly. The frontend components are already set up to handle loading states, display results, and process the response data from your API.

Troubleshooting

If you encounter issues:

1. Verify your backend endpoint is accessible at `/check/<text>/`
2. Confirm your backend returns the exact JSON structure expected
3. Check browser console for network or JavaScript errors
4. Ensure CORS is properly configured if your backend is on a different domain