EXP 3: Evaluation of Prompting Tools Across Diverse AI Platforms: ChatGPT, Claude, Bard, Cohere Command, and Meta

Aim:

To compare the performance, user experience, and response quality of different AI platforms (ChatGPT, Claude, Bard, Cohere Command, and Meta) within a specific use case, such as summarizing text or answering technical questions.

Scenario: Frontend Engineering for Health-Care management Website

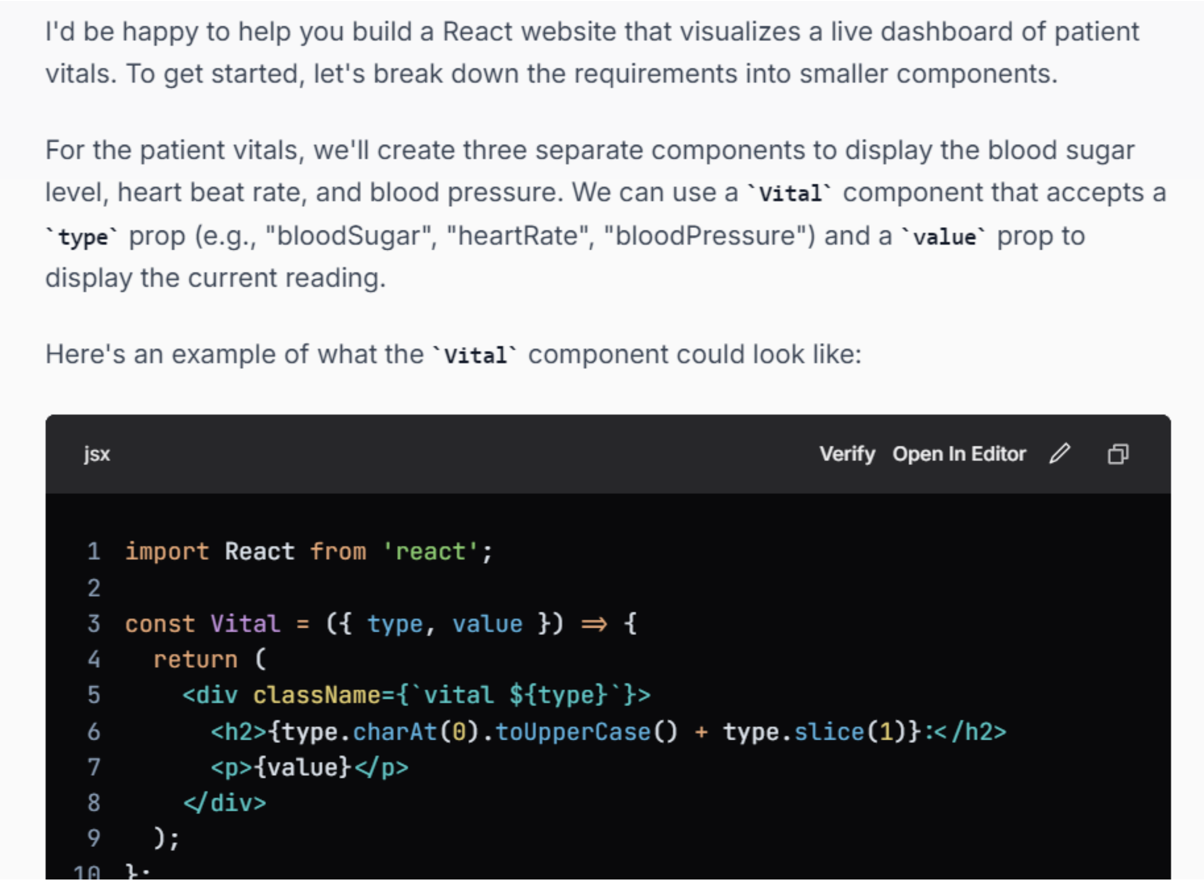
Prompt:

"You are expert Frontend engineer; can you help me build a website with React that will help me to visualize live dashboard of patient vitals. Here are the details you need to consider

Patient vitals: It should contains blood sugar level, heart beat rate and blood pressure.

Include proper modular CSS in react to style the components. Implement real-time updating of data in the dashboard using any technology like sockets or message queues."

Blackbox.ai [https://www.blackbox.ai] :

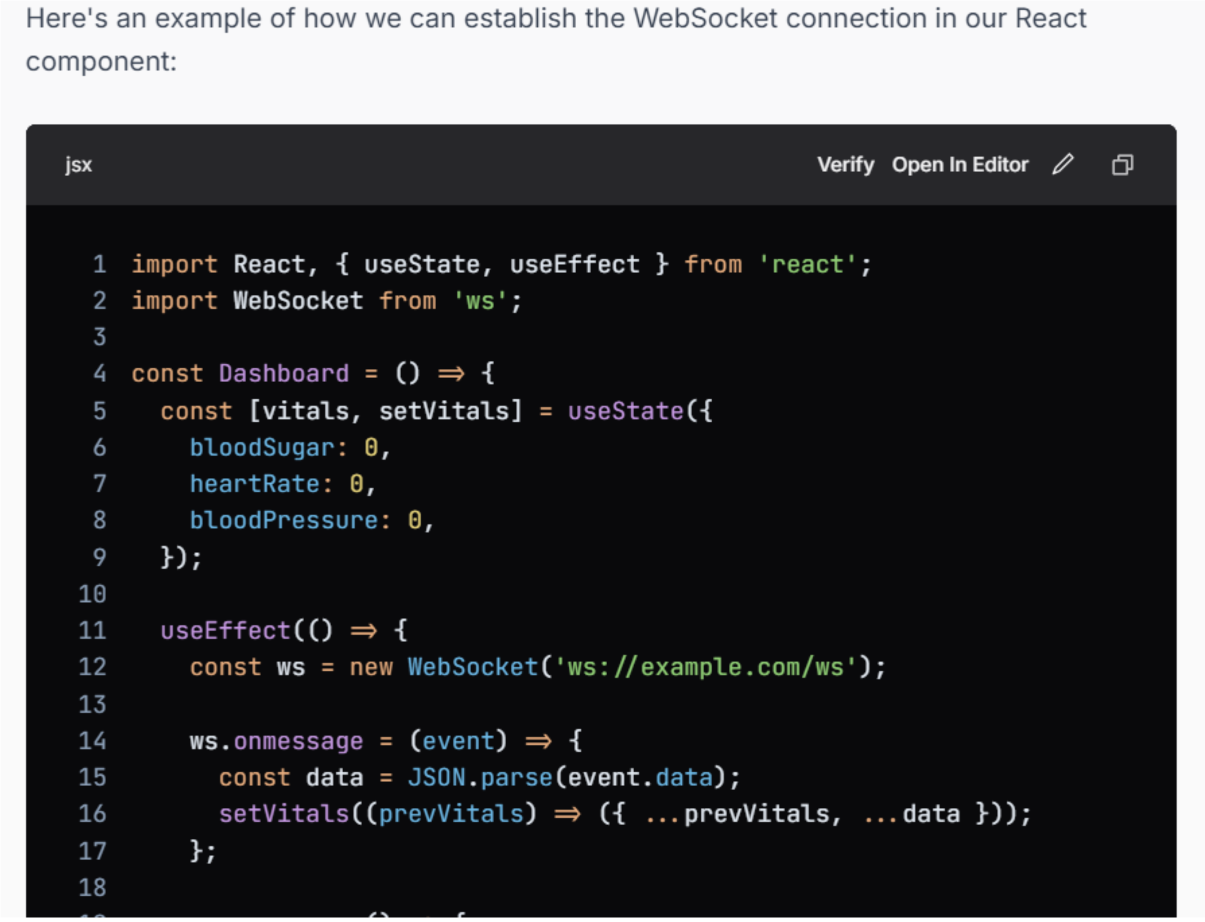


I'd be happy to help you build a React website that visualizes a live dashboard of patient vitals. To get started, let's break down the requirements into smaller components.

For the patient vitals, we'll create three separate components to display the blood sugar level, heart beat rate, and blood pressure. We can use a `Vital` component that accepts a `type` prop (e.g., "bloodSugar", "heartRate", "bloodPressure") and a `value` prop to display the current reading.

Here's an example of what the `Vital` component could look like:

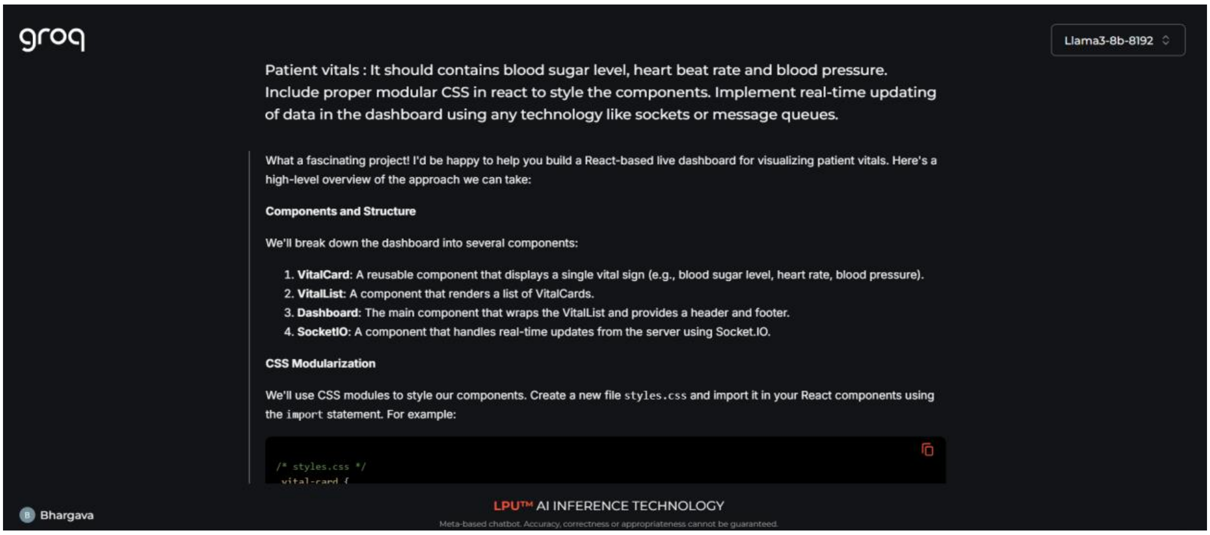
```jsx
import React from 'react';

const Vital = ({ type, value }) => {
  return (
    <div className={`vital ${type}`}>
      <h2>{type.charAt(0).toUpperCase() + type.slice(1)}:</h2>
      <p>{value}</p>
    </div>
  );
};
```



Here's an example of how we can establish the WebSocket connection in our React component:

```jsx
import React, { useState, useEffect } from 'react';
import WebSocket from 'ws';

const Dashboard = () => {
  const [vitals, setVitals] = useState({
    bloodSugar: 0,
    heartRate: 0,
    bloodPressure: 0,
  });

  useEffect(() => {
    const ws = new WebSocket('ws://example.com/ws');

    ws.onmessage = (event) => {
      const data = JSON.parse(event.data);
      setVitals((prevVitals) => ({ ...prevVitals, ...data }));
    };

```

Groq: [https://chat.groq.com/]



groq
Llama3-8b-8192
Bhargava
LPU™ AI INFERENCE TECHNOLOGY
Meta-based chatbot. Accuracy, correctness or appropriateness cannot be guaranteed.

Patient vitals : It should contains blood sugar level, heart beat rate and blood pressure. Include proper modular CSS in react to style the components. Implement real-time updating of data in the dashboard using any technology like sockets or message queues.

What a fascinating project! I'd be happy to help you build a React-based live dashboard for visualizing patient vitals. Here's a high-level overview of the approach we can take:

Components and Structure

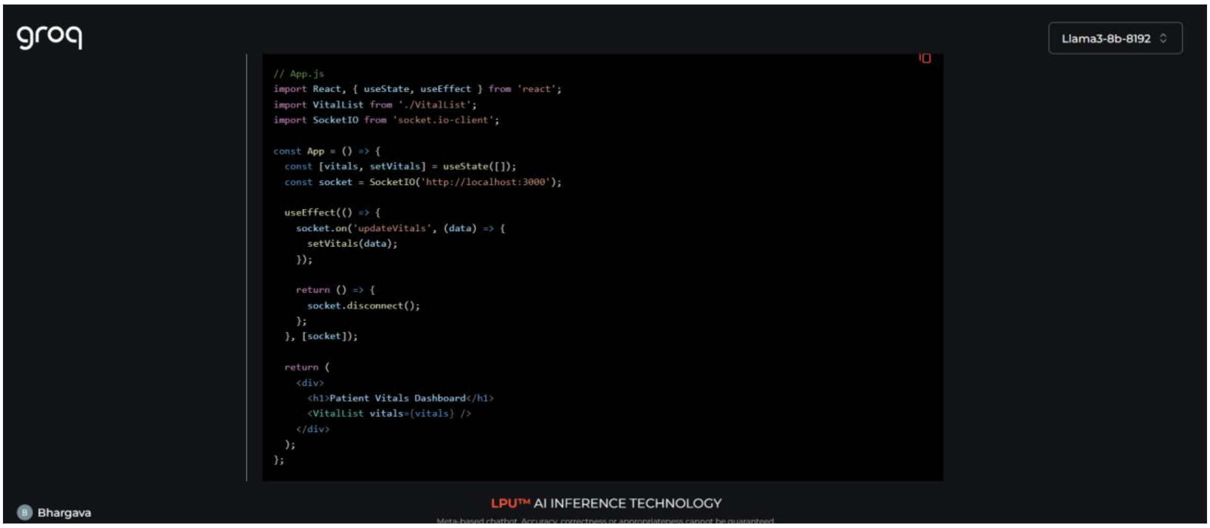
We'll break down the dashboard into several components:

1. **VitalCard**: A reusable component that displays a single vital sign (e.g., blood sugar level, heart rate, blood pressure).
2. **VitalList**: A component that renders a list of VitalCards.
3. **Dashboard**: The main component that wraps the VitalList and provides a header and footer.
4. **SocketIO**: A component that handles real-time updates from the server using Socket.IO.

CSS Modularization

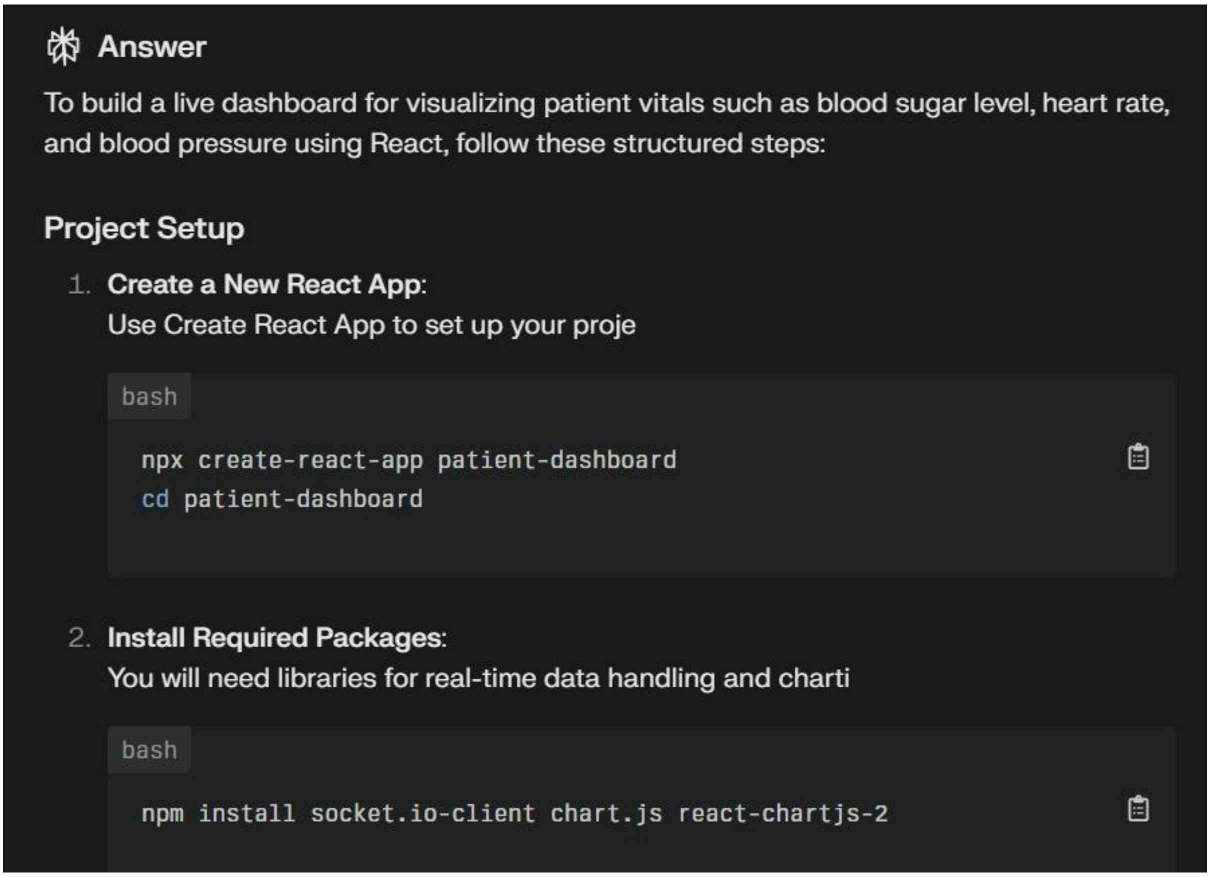
We'll use CSS modules to style our components. Create a new file `styles.css` and import it in your React components using the `import` statement. For example:

```css
/* styles.css */
```


groq
Llama3-8b-8192
Bhargava
LPU™ AI INFERENCE TECHNOLOGY
Meta-based chatbot. Accuracy, correctness or appropriateness cannot be guaranteed.

```
// App.js
import React, { useState, useEffect } from 'react';
import VitalList from './VitalList';
import SocketIO from 'socket.io-client';

const App = () => {
  const [vitals, setVitals] = useState([]);
  const socket = SocketIO('http://localhost:3000');

  useEffect(() => {
    socket.on('updateVitals', (data) => {
      setVitals(data);
    });

    return () => {
      socket.disconnect();
    };
  }, [socket]);

  return (
    <div>
      <h1>Patient Vitals Dashboard</h1>
      <VitalList vitals={vitals} />
    </div>
  );
};
```

Perplexity : [https://www.perplexity.ai/]



To build a live dashboard for visualizing patient vitals such as blood sugar level, heart rate, and blood pressure using React, follow these structured steps:

Project Setup

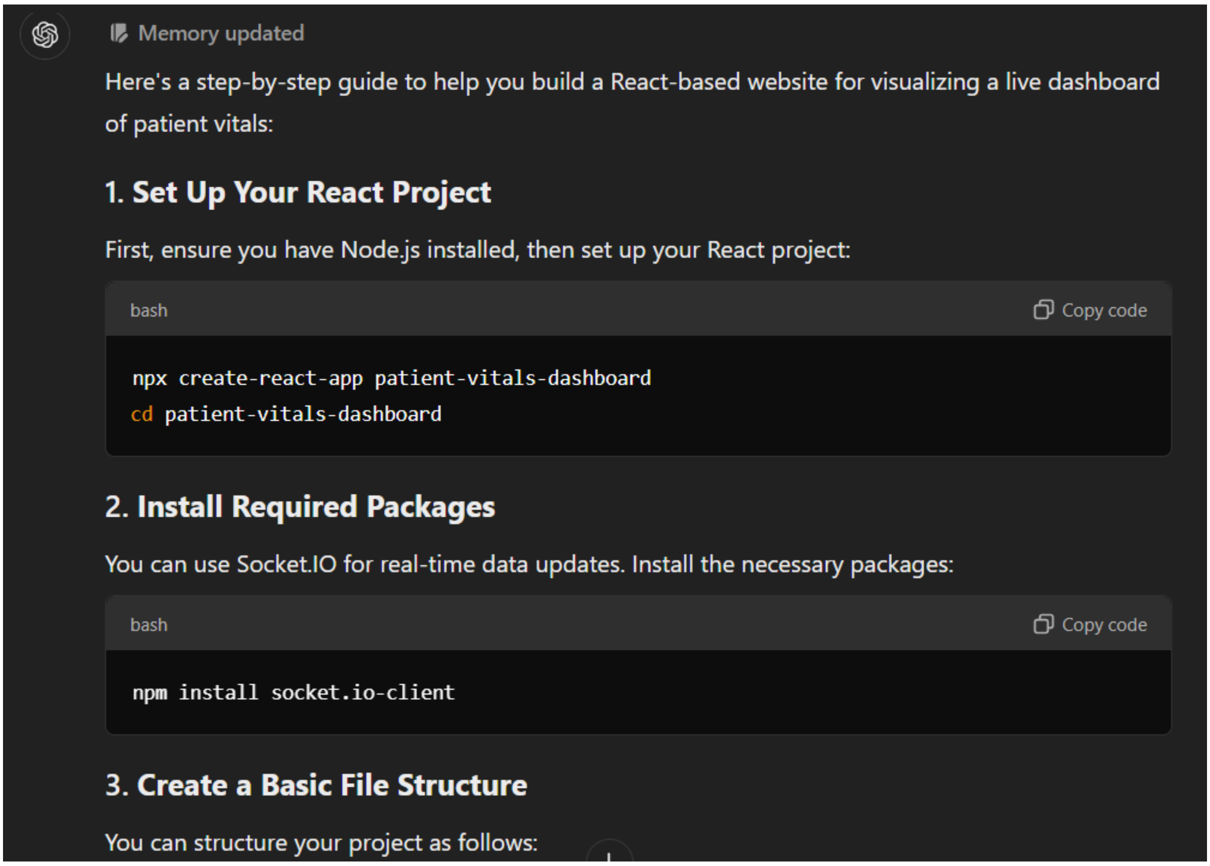
1. **Create a New React App**:
   Use Create React App to set up your proje

```bash
npx create-react-app patient-dashboard
cd patient-dashboard
```

2. **Install Required Packages**:
   You will need libraries for real-time data handling and charti

```bash
npm install socket.io-client chart.js react-chartjs-2
```

ChatGPT [https://chatgpt.com/]



Here's a step-by-step guide to help you build a React-based website for visualizing a live dashboard of patient vitals:

1. Set Up Your React Project

First, ensure you have Node.js installed, then set up your React project:

```bash
npx create-react-app patient-vitals-dashboard
cd patient-vitals-dashboard
```

2. Install Required Packages

You can use Socket.IO for real-time data updates. Install the necessary packages:

```bash
npm install socket.io-client
```

3. Create a Basic File Structure

You can structure your project as follows:

Platform Accuracy Clarity Depth Relevance

GROQ	Medium: While the response accurately outlines a structured approach, some details on implementation specifics, such as handling edge cases, are missing.	High: The response is well-organized and easy to follow, using headings and bullet points to clearly delineate different aspects of the solution.	Medium: The overview is comprehensive, but it lacks deeper insights into error handling, performance optimization, and specific implementation details that may be beneficial for developers.	High: The information provided is directly applicable to the project, addressing all key requirements such as real-time data updates and modular CSS.
Perplexity	High: The response provides a clear and accurate guide for setting up a live dashboard, including component design and real-time data	High: The structured format with clear headings and code snippets enhances readability and understanding, making it easy to	Medium: The response offers good guidance but could include more advanced topics like state management and testing, which are crucial for	High: The detailed instructions and components directly align with the project requirements, ensuring relevance

	handling with practical code examples.	follow the implementation steps.	scalable applications.	for building the application.
Blackbox AI	Medium: The response gives a basic overview but lacks detailed guidance on the server-side logic required for a fully functional application.	Medium: The explanation is straightforward, but some areas lack depth, making it harder for less experienced developers to fully grasp the implementation.	Low: While the response covers essential components and WebSocket integration, it does not provide enough detail on advanced features or error handling strategies.	Medium: The information is relevant but does not address all necessary components for a comprehensive solution, particularly in server implementation.
ChatGPT	High: The response provides a detailed step-by-step guide for building a React-based live dashboard, including setup, components, styling, and server-side integration for real-time updates.	High: The response is very clear and well-structured, using numbered steps and code examples to enhance understanding and usability for developers.	High: The depth is significant, covering everything from project setup to styling, real-time updates, and even a basic server implementation, offering a comprehensive framework for building the application.	High: The content is highly relevant to the project, addressing all aspects necessary for creating a functional dashboard with real-time capabilities.

Summary & Conclusion

In this comparison, we analyzed the responses from four different platforms regarding the development of a React-based live dashboard for visualizing patient vitals. Each platform provided valuable insights, but they differed significantly in accuracy, clarity, depth, and relevance.

1. **GROQ** offered a solid structure but lacked some details that could enhance the implementation process, particularly around edge cases and error handling.
2. **Perplexity** provided a clear guide but could have delved deeper into advanced topics like state management and testing for scalable applications.
3. **Blackbox AI** presented a basic overview, which was less comprehensive and lacked the necessary detail for server-side implementation, making it less useful for beginners.
4. **ChatGPT** stood out with a thorough, well-structured guide that effectively covered all critical aspects of building the application, making it the most useful response overall.

Overall, the analysis shows that while all platforms offered useful guidance, the depth and clarity of the information varied significantly, impacting their practical utility for developers looking to implement a robust solution. ChatGPT's response proved to be the most comprehensive and directly applicable to the project needs, making it the top choice for developers seeking detailed instructions for building a patient vitals dashboard.