I'd be glad to assist you with sending a push of data from a Java Spring Boot application to a React presentation layer. Here's a comprehensive approach combining the best aspects of the provided responses and addressing potential shortcomings:

**1. Choose a Push Notification Technology:**

- **WebSockets:**
  - Real-time bidirectional communication between client and server.
  - Ideal for frequent updates or interactive applications.
  - Libraries like Spring WebSocket or SockJS can simplify implementation.
- **Server-Sent Events (SSE):**
  - One-way communication from server to client.
  - Suitable for less frequent updates or data streaming.
  - Spring Boot offers built-in SSE support.
- **Polling:**
  - Client periodically sends requests to the server for updates.
  - Less efficient than WebSockets or SSE for frequent updates.
  - Consider using a long polling mechanism to minimize overhead.

**2. Set Up Spring Boot Server:**

- Create a Spring Boot project and add dependencies for your chosen push notification technology (e.g., Spring WebSocket, Spring Boot Starter SSE).
- Configure the server-side endpoint to handle incoming connections or requests.
- Implement logic to send data updates to clients using the appropriate methods (e.g., `SimpleBrokerMessageHandler` for WebSockets, `SseEmitter` for SSE).

**3. Create React Client:**

- Set up a React project and add dependencies for handling push notifications (e.g., `sockjs-client` for WebSockets, `EventSource` for SSE).
- Establish a connection to the server-side endpoint using the appropriate methods.
- Implement event handlers to receive and process data updates from the server.
- Update the UI based on the received data.

**Example using WebSockets:**

**Server (Spring Boot):**

Java

```
@Configuration
@EnableWebSocketMessageBroker
public class WebSocketConfig implements WebSocketMessageBrokerConfigurer {

    @Override
    public void registerStompEndpoints(StompEndpointRegistry registry) {
        registry.addEndpoint("/ws").withSockJS();
    }

    @Override
    public void configureMessageBroker(MessageBrokerRegistry registry) {
        registry.setApplicationDestinationPrefixes("/app");
        registry.enableSimpleBroker("/topic");
    }
}

@Controller
public class GreetingController {

    @MessageMapping("/hello")
    public void handleMessage(Message message) {
        // Process the message and send a response
        messagingTemplate.convertAndSend("/topic/greetings", new
Greeting("Hello, " + message.getName() + "!"));
    }
}
```

**Client (React):**

JavaScript

```
import React, { useEffect, useState } from 'react';
import SockJS from 'sockjs-client';
import Stomp from 'stompjs';

function App() {
  const [greetings, setGreetings] = useState([]);

  useEffect(() => {
    const socket = new SockJS('http://localhost:8080/ws');
    const stompClient = Stomp.over(socket);

    stompClient.connect({}, (frame) => {
      console.log('Connected: ' + frame);
      stompClient.subscribe('/topic/greetings',   (message) => {
        const greeting = JSON.parse(message.body);
        setGreetings([...greetings, greeting]);
      });
    });

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

  return (
    <div>
      {greetings.map((greeting) => (
        <p key={greeting.id}>{greeting.content}</p>
      ))}
    </div>
  );
}

export default App;
```

Remember to adjust the code and configurations based on your specific requirements and chosen push notification technology.