



**COLLEGE CODE : 9504**

**COLLEGE NAME : DR.G.U.POPE COLLEGE OF ENGINEERING**

**DEPARTMENT : COMPUTER SCIENCE AND ENGINEERING**

**STUDENT NM ID : D1468F085E9D450B2A34825AF30855B5**

**ROLL NO : 11**

**DATE : 29.09.2025**

**Completed the Phase-04**

**PROJECT NAME : IBM-FE- CHAT APPLICATION UI**

**SUBMITTED BY,**

**HEPZIBAH E  
9843043546**

# ENHANCEMENT & DEPLOYMENT

## Additional Features

Building upon the solid MVP from Phase 3, this phase introduced key enhancements to improve functionality and user experience.

**Real-Time Communication:** Integrated the socket.io-client library to replace mocked message sending/receiving.

- Implemented the useSocket custom hook to manage the socket connection within the AuthContext, establishing a connection upon login and disconnecting on logout.
- The client emits send\_message events when a user sends a message.
- The client listens for new\_message events, automatically appending incoming messages to the active conversation or updating the conversation list if the chat is not open.
- Implemented typing indicators by emitting and listening for user\_typing events.

**Message Status Indicators:** Enhanced the MessageBubble component and the optimistic update logic to visually display message states (sending, sent, delivered). This provides users with clear feedback on their communication.

**Message History Persistence:** Integrated the frontend with a mock backend service (e.g., JSON Server or a more advanced MSW handler) to simulate the persistent storage and retrieval of message history, ensuring users see their full conversation context upon reload.

**Search & Filter Functionality:** Enhanced the SearchBar molecule to allow users to search through their conversation list by participant name or group name, improving navigability as the number of chats grows.

## UI/UX Improvements

A focus on polish and accessibility ensured the application is not only functional but also pleasant to use.

**Loading States & Skeleton Screens:** Replaced basic loading spinners with skeleton screens for the ConversationList and MessageList, providing a more seamless perceived performance.

**Empty States:** Designed and implemented friendly empty state components for when there are no conversations or no messages in a chat, guiding the user on what to do next.

**Micro-interactions:** Added subtle transitions and animations using Tailwind CSS or Framer Motion for a smoother feel, such as when opening modals, sending messages, and switching conversations.

### **Enhanced Accessibility (a11y):**

- Improved color contrast ratios to meet WCAG guidelines.
- Added proper ARIA labels to interactive elements.
- Ensured all functionality is accessible via keyboard navigation.

Theme Consistency: Conducted a full review of the UI to ensure consistent spacing, font sizes, and color usage across all components, solidifying the design system.

### **API Enhancements**

The frontend's interaction with backend services was made more robust and realistic.

**Error Handling:** Enhanced the useApi hook and axios interceptors to gracefully handle API errors (e.g., network issues, 4xx/5xx responses) by displaying user-friendly error messages without crashing the UI.

**Authentication Integration:** Updated the AuthContext and api.js service to automatically attach the authentication token (from a successful mock login) to the headers of all subsequent API requests, simulating a real-world authentication flow.

**Pagination for Messages:** Implemented infinite scroll or a "Load More" button in the MessageList component, triggering calls to a paginated mock API endpoint (e.g., GET /conversations/:id/messages?page=1) to handle large message histories efficiently.

### **Performance & Security Checks**

Proactive measures were taken to ensure the application is fast, reliable, and secure.

#### **Performance Audit:**

- Used the React DevTools Profiler to identify and memoize expensive components with React.memo and useCallback to prevent unnecessary re-renders.
- Implemented code splitting using React.lazy() and Suspense for the LoginPage and ChatPage, reducing the initial bundle size.
- Verified that images and avatars are optimized.

#### **Security Review:**

- Sanitized user input in the MessageInput to prevent potential XSS attacks.
- Ensured that no sensitive data (like mock tokens) is logged to the console in production.
- Validated that environment variables are used for configuration keeping deployment-specific secrets out of the codebase.

## Testing of Enhancements

The new features and improvements were rigorously tested to maintain quality.

**Integration Testing:** Wrote tests to verify the interaction between components and the Socket.IO connection, ensuring messages are sent and received correctly.

**User Flow Testing:** Manually tested complete user journeys, such as: User A sends a message -> User B's interface updates in real-time and shows a notification.

**Cross-Browser & Device Testing:** Verified that all enhancements work consistently across major browsers (Chrome, Firefox, Safari) and on various device sizes.

## Deployment (Netlify/Vercel)

The application was successfully deployed to a live environment, making it accessible online.

**Platform Selection:** Chose Netlify for its simplicity, excellent support for SPAs, and seamless Git integration.

### Build Configuration:

- Created a netlify.toml file to specify the build command (npm run build) and publish directory (dist).
- Configured build environment variables in the Netlify dashboard.

### Deployment Process:

- Connected the GitHub repository to Netlify.
- Pushing code to the main branch automatically triggers a new build and deployment.
- Netlify provides a unique, shareable URL for the live application (e.g., <https://graceful-pastelito-123abc.netlify.app>).

### Post-Deployment Verification:

- Thoroughly tested all core features and enhancements on the live URL to ensure they function as expected in the production environment.
- Verified that the site is served over HTTPS.