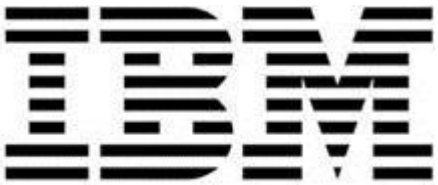

COLLEGE CODE : 9504

COLLEGENAME : DR.G.U.POPECOLLEGE OF ENGINEERING

DEPARTMENT : COMPUTERSCIENCE AND ENGINERRING

STUDENT NM ID : D1468F085E9D450B2A34825AF30855B5

ROLL NO : 11

DATE : 22.09.2025

Completed the Phase-03

PROJECT NAME : IBM-FE- CHAT APPLICATION UI

SUBMITTED BY,
HEPZIBAH E
9843043546

MVP IMPLEMENTATION

Project Setup

This phase began with the initial setup of the project repository and development environment, aligning with the tech stack defined in Phase 2.

Repository Initialization:

A new GitHub repository (ibm-fe-chat-app) was created and initialized with a README.md file detailing the project, tech stack, and setup instructions.

Development Environment:

- Node.js & npm: Ensured the correct LTS version of Node.js was installed.
- Vite Project Scaffolding: Executed `npm create vite@latest . -- --template react` to initialize a new React project with Vite for its fast development server and optimized build tooling.
- Dependency Installation:

  Installed the core libraries defined in the architecture:
  - ✧ react, react-dom
  - ✧ tailwindcss (with PostCSS and Autoprefixer)
  - ✧ axios for API calls
  - ✧ socket.io-client for real-time communication
  - ✧ msw (Mock Service Worker) for mocking backend APIs

Project Structure:

Implemented the planned atomic folder structure within the src directory, creating folders for components/atoms, components/molecules, components/organisms, contexts, hooks, pages, and services.

Core Features Implementation

The implementation focused on delivering the MVP features as per the user stories and acceptance criteria.

- **User Authentication UI (US-01):**
  - Built LoginPage and RegistrationPage components with form validation.
  - Implemented AuthContext to globally manage the user's authentication state (token, user data).
  - Used MSW to mock successful and failed responses from the /auth/login endpoint (AC1, AC3).
  - Upon successful mock login, the user is redirected to the ChatPage (AC2).
- **Conversations List View (US-01, US-07):**
  - Developed the ConversationList organism and ConversationListItem molecule.
  - Fetched mock conversation data from the mocked /conversations endpoint using axios inside a useEffect hook (AC4).
  - Each list item successfully displays the conversation name, last message preview, timestamp, and an unread count badge (AC5).
  - Styled the list to highlight conversations with unread messages.
- **1-on-1 & Group Chat Interface (US-03, US-04, US-05):**
  - Built the MessageList organism and MessageBubble molecule. Sent and received messages have distinct visual styling (aligned right/left, different background colors) (AC7).
  - Implemented the MessageInput component with a form handler.
  - Optimistic Updates: Implemented logic where a sent message is immediately added to the local UI state with a status: 'sending' before the API/Socket call is made (AC8). The status updates to 'sent' upon a successful mock response.
  - Mocked the API endpoint POST /conversations/:id/messages with MSW.
- **Group Chat Creation (US-05, US-06):**
  - Created a modal component for group creation.
  - The modal includes a form with a group name field and a searchable list of users (fetched from the mocked GET /users endpoint) (AC9).
  - Upon form submission, a POST request is sent to the mocked /conversations/group endpoint (AC10).

- **Responsive Design (US-08):**

  - Utilized Tailwind CSS's responsive utility classes (e.g., flex, flex-col, md:flex-row).
  - Achieved a side-by-side two-panel layout on desktop viewports (> 1024px) (AC11).
  - Implemented a conditional rendering logic where on mobile viewports (< 768px), the conversation list and chat view are displayed as separate full-screen pages, toggled based on the selected conversation state (AC12).

Data Storage (Local State / Database)

As this is the UI phase, data persistence is simulated. The architecture for handling data flow was implemented as designed.

UI State:

- Managed using React's useState and useReducer hooks.
- Input field values, modal open/close states, and loading indicators are local component state.
- The currently selected conversation ID is stored in the ConversationsContext.

Server State (Mocked):

- **Global State:** The AuthContext stores the authenticated user's token and profile. The ConversationsContext stores the array of conversation objects.
- **Local Component State:** Messages for the currently selected conversation are fetched and stored in the ChatPage component's state using useState to avoid global bloat.
- **Mocking:** MSW intercepts all API calls defined in the Phase 1 document (/auth/login, /conversations, /messages, etc.) and returns static, hard-coded JSON data matching the API schema from Phase 2. This allows for full frontend functionality without a backend.

Testing Core Features

A testing strategy was employed to ensure feature reliability.

Manual Testing: Each user story and acceptance criterion was manually tested:

- ✓ Successful and failed login attempts.
- ✓ Display and update of the conversation list.
- ✓ Selecting conversations and loading respective messages.
- ✓ Sending messages and seeing optimistic updates.
- ✓ Creating a new group chat.
- ✓ Verifying responsive design on various screen sizes.

Tooling Setup: Installed and configured jest and @testing-library/react for component testing. Wrote initial unit tests for key utility functions and basic rendering tests for core components like the MessageBubble and ConversationListItem.

Version Control (GitHub)

Git and GitHub were used extensively for version control and collaboration throughout this phase.

Branching Strategy:

Utilized feature-based branching.

- main branch always holds the production-ready, stable code.
- New features were developed in isolated branches (e.g., feature/auth-login, feature/chat-interface).

Commit Hygiene: Followed conventional commit messages (e.g., feat: add login page component, fix: resolve responsive issue on mobile). Commits were small and atomic, focusing on a single change.

Pull Requests (PRs): Each feature branch was merged into main via a Pull Request. PR descriptions included summaries of changes, screenshots of the new UI, and a checklist of completed acceptance criteria, facilitating code review.

Repository Link: The complete source code for this phase has been pushed to the GitHub repository