Design Choices

1.State Management

React's Context API and useReducer were selected for global state management. This provides:

- A centralized way to manage the app's state.
- A clear and scalable structure for handling complex state updates like adding messages or selecting contacts.

2.UI Design

- Tailwind CSS was used for rapid styling and responsiveness.
- Components such as ChatWindow and ContactList were modularized to ensure reusability and maintainability

Challenges Faced

Responsive Design

Ensuring the chat interface and contact list were fully responsive across all screen sizes required extensive testing and adjustments.

Usage of Key Features

React Hooks

useState

- Used for managing local component-specific states.
- Example: The `searchQuery` state in `ContactList` stores the user's search input.

```
const [searchQuery, setSearchQuery] = useState("");
```

useEffect

- Handles side effects like loading data from IndexedDB and syncing updates.
- Example: Fetching contacts and messages when the app initializes.

```
useEffect(() => {
  loadFromIndexedDB(dispatch);
}, []);
```

React Context

Purpose

- Centralized management of global state (e.g., user, contacts, messages).
- Provides state and dispatch to components via a `useAppContext` custom hook.

Example

```
const { appState, dispatch } = useAppContext();
```

What It Manages

- User Information: The logged-in user's details.
- Contacts: A list of saved contacts.
- Messages: Chat messages exchanged with selected contacts.
- Selected Contact: The currently selected contact for the chat window.

Custom Hooks

useAppContext

- Simplifies accessing and dispatching actions to the global state.
- Example Usage in `ChatWindow`:

```
const { appState, dispatch } = useAppContext();
const handleSignOut = () => {
  dispatch({ type: "LOGOUT" });
};
```

---

useReducer

Purpose

- Manages complex state logic with clearly defined actions.
- Used in the `AppContext` to handle global state updates.

Example Reducer Logic

```
const appReducer = (state, action) => {
  switch (action.type) {
    case "SET_CONTACTS":
      return { ...state, contacts: action.payload };
    case "ADD_MESSAGE":
      return { ...state, messages: [...state.messages,
action.payload] };
    default:
      return state;
  }
};
```

InstantDB

Purpose

- Simplifies querying data from IndexedDB.
- Example: Querying messages exchanged between two users.

```
const query = {
  messages: {
    $: {
      where: {
        and: [
          { or: [{ sender: user }, { sender: endUser }] },
          { or: [{ receiver: user }, { receiver: endUser }] },
        ],
      },
    },
  },
};
const { data, isLoading, error } = db.useQuery(query);
```