Simplified Messaging App System Design

1. Functional Requirements
- User Management: Sign up / log in (email + password), Profile (username, avatar URL)
- Messaging: One-to-one chat, Message history (persisted)
- Real-Time Updates: New messages delivered instantly via WebSocket

2. High-Level Architecture
[React SPA] ↕ HTTPS/REST + WSS [Node.js + Express + Socket.io] ↕ [MongoDB]

3. Simplified Data Model
User: _id, username, email, passwordHash, avatarUrl, lastSeen
Conversation: _id, participants (two user IDs), createdAt
Message: _id, conversationId, sender, content, createdAt

4. Core API Endpoints
Auth: POST /api/auth/signup → {username, email, password} → {token, user}
POST /api/auth/login → {email, password} → {token, user}
Conversations & Messages:
GET /api/conversations → list of {conversation, lastMessage}
POST /api/conversations → {participants: [id, id]} → new conversation
GET /api/conversations/:id/messages → full message list
POST /api/conversations/:id/messages → {content} → new message

5. Real-Time Flow (Socket.io)
```
io.on('connection', socket => { … authenticate JWT, join rooms,
  socket.on('send_message', … save to DB, emit 'new_message');
});
```

6. Tech Stack Summary
Front-end: React, React Router, Axios
Real-time Layer: Socket.io
Back-end API: Node.js, Express
Database: MongoDB (Mongoose)
Auth & Security: JWT, bcrypt
DevOps: Docker, GitHub Actions CI/CD
Hosting: Heroku / DigitalOcean / AWS EC2