

Engineering Manager Take-Home: ChatFlow - A Messaging Application

Overview

Build a simplified WhatsApp-like messaging application called "ChatFlow" that demonstrates your technical skills, architectural thinking, and approach to building scalable systems.

Time Allocation:

Focus: Code quality, system design, and architectural decisions over feature completeness

Core Requirements

Functional Requirements

1. User Management

- User registration with email/username
- User authentication
- Basic user profiles (name, avatar optional)

2. Messaging

- Send text messages between users
- View conversation history
- Create group chats (3+ participants)
- Real-time message delivery (when both users online)

3. Chat Management

- List all conversations for a user
- Create new conversations
- Basic message timestamps

Technical Requirements

- RESTful API backend
- Simple frontend (can be web, mobile, or even CLI)
- Database for data persistence
- Real-time communication (WebSockets, Server-Sent Events, or polling)
- Basic error handling and validation

Technical Constraints

- Choose your preferred tech stack
- Include a simple deployment/setup process
- Write tests for core functionality
- Document your architectural decisions

Deliverables

1. Working Application

- Backend API with core endpoints
- Frontend interface (basic UI acceptable)
- Database schema and setup
- README with setup instructions

2. Technical Documentation

- **Architecture Overview**
 - High-level system design
 - Technology choices and rationale
 - Database schema design
 - API design decisions
- **Scaling Considerations**
 - How would you scale this to 10K concurrent users?
 - What bottlenecks do you anticipate?
 - What would you change in the architecture?
- **Trade-offs and Decisions** (1 page)
 - Key technical decisions made
 - Alternative approaches considered
 - What you'd do differently with more time

3. Code Organization

- Clear project structure
- Separation of concerns
- Basic error handling
- At least 3-5 meaningful tests

Evaluation Criteria

Technical Execution (40%)

- Code quality and organization
- API design and RESTful principles
- Database design and queries
- Error handling and edge cases
- Test coverage of core paths

System Design Thinking (35%)

- Architectural decisions
- Scalability considerations
- Technology choice rationale
- Understanding of real-time communication challenges

Engineering Leadership (25%)

- Documentation quality
- Setup and deployment clarity
- Trade-off analysis
- Communication of technical decisions

Bonus Points (if time permits)

- Message status indicators (sent, delivered)
- User online/offline status
- Message search functionality
- Basic message encryption
- Docker containerization
- API rate limiting

API Endpoints (Suggested Minimum)

Unset

POST /auth/register

POST /auth/login

GET /users/me

GET /conversations

POST /conversations

GET /conversations/:id/messages

POST /conversations/:id/messages

WebSocket /ws (for real-time updates)

Submission Guidelines

1. Host code on GitHub (public or share access)
2. Include comprehensive README with:
 - Setup instructions
 - API documentation
 - Architecture overview
3. Ensure application can be run locally
4. Include your technical documentation files

What We're NOT Looking For

- Perfect UI/UX design
- Production-ready security implementation
- Complex deployment infrastructure
- Feature completeness over code quality





What We ARE Looking For

- Clean, readable code
- Thoughtful architectural decisions
- Clear documentation and communication
- Understanding of system design principles
- Practical trade-off considerations
- Leadership-level technical thinking

AI Coding Assistants - Encouraged!

We actively encourage the use of AI coding assistants (GitHub Copilot, ChatGPT, Claude, etc.) to help with this assignment. Modern engineering teams leverage these tools, and we want to see how effectively you can work with them.

What This Means:

-  Use AI to help write boilerplate code, generate tests, or explain concepts
-  Leverage AI for debugging and optimization suggestions
-  Get help with documentation and README creation
-  Use AI to explore different architectural approaches

Critical Requirement:

You MUST deeply understand every line of code you submit. During the follow-up review interview, we will:

- Ask you to walk through your code architecture live
- Request possible modifications and extensions to your solution
- Discuss alternative implementation approaches
- Ask detailed questions about your technical decisions
- Have you debug or extend functionality in real-time

Pro Tips for AI Usage:

- Review and understand all AI-generated code before including it
- Test AI suggestions thoroughly
- Be prepared to explain why you chose AI-generated solutions
- Document any significant AI assistance in your README

Follow-Up Review Interview

After submission, we'll schedule a 60-90 minute technical review where you'll:

1. **Code Walkthrough** (20 mins): Present your architecture and key decisions
2. **Live Coding** (30 mins): Implement additional features or modifications
3. **System Design Discussion** (20 mins): Explore scaling and architectural questions
4. **Technical Q&A** (15 mins): Deep dive into specific implementation choices

Example Follow-up Challenges:

- "Add message reactions functionality"
- "How would you implement message search?"
- "Debug this real-time connection issue"
- "Modify the API to support message threading"

Questions?

Feel free to ask clarifying questions. We encourage questions about scope, requirements, or technical approaches as they demonstrate good engineering judgment and communication skills.
