Product Requirements Document (PRD)

Table of Contents

- ChatFlow MVP - Team Communication Platform
- Table of Contents
- Document Metadata
- 1. Executive Summary
- 2. Problem Statement
- 3. Vision & Strategic Objectives
- 4. Target Users & Personas
- 5. Core Features - MVP Scope
- 6. User Stories & Use Cases
- 7. Technical Architecture (Lab Deployment)
- SSH into lab VM
- Password: slackteam123
- Verify Node.js version
- Install system dependencies
- Start services
- Create PostgreSQL database
- Test connection
- Clone repository
- Install dependencies
- Create .env file
- Run migrations
- Start backend
- Or use PM2 for process management:
- Build React app
- Copy to Nginx
- OR: Setup Nginx config for SPA routing
- Test frontend
- Test backend
- Monitor logs
  - 8. Database Schema (PostgreSQL)
  - 9. API Specification (REST + WebSocket)
  - 10. Non-Functional Requirements
  - 11. Release Timeline & Milestones
  - 12. Success Criteria (QA Acceptance)
  - 13. Known Limitations & Future Work
  - 14. Sign-Off & Approvals
  - 15. Appendices

ChatFlow MVP - Team Communication Platform

Document Status: Final for Development & Lab Deployment
Last Updated: November 19, 2025
Next Review: Upon MVP Milestone 1 completion (Week 1 end)

Table of Contents

1. Executive Summary
2. Problem Statement
3. Vision & Strategic Objectives
4. Target Users & Personas
5. Core Features - MVP Scope
6. User Stories & Use Cases
7. Technical Architecture (Lab Deployment)
8. Database Schema (PostgreSQL)
9. API Specification (REST + WebSocket)
10. Non-Functional Requirements
11. Release Timeline & Milestones
12. Success Criteria (QA Acceptance)
13. Known Limitations & Future Work
14. Sign-Off & Approvals
15. Appendices

Document Metadata

Attribute	Value
Project Name	ChatFlow MVP - Team Communication Platform
Version	2.0 (Lab-Optimized)
Date	November 19, 2025
Status	Final for Development & Deployment
Author	Senior Software Architect
Team	Slack Team (slackteam.lab.home.lucasacchi.net)
Environment	Lab/Development (Production-ready)

Deployment Infrastructure Specification

Parameter	Configuration
Hostname	slackteam.lab.home.lucasacchi.net
SSH User	slackteam
SSH Password	slackteam123
Frontend Port	8282
Backend Port	4000
Node.js Version	24.11.1 (LTS)

Parameter	Configuration
Python Version	3.11.2
Deployment Type	Single VM (Monolithic)
Target Users	5-200 concurrent
Message Throughput	10K-100K messages/day

1. Executive Summary

ChatFlow MVP is a real-time communication platform for teams, designed to replace fragmented communication tools (email, WhatsApp, SMS) with a low-cost alternative.

Value Proposition

✅ Real-time messaging with <500ms latency
✅ Channel-based organization
✅ Direct messaging (1-on-1 and group)
✅ Full-text search on messages
✅ File sharing (50MB limit)
✅ User presence & status
✅ Markdown + @mentions + emoji reactions
✅ Deploy on lab infrastructure in <1 week

MVP Scope: Core messaging, channel management, authentication, notifications

Target Launch: Week 2 November 2025 (Production Ready)

Success Metric: 1,000+ messages/day, 99.5% uptime, <500ms message latency

Primary Goals (MVP Phase)

Goal	Target	Measurement
User Adoption	50+ active team members in lab	Daily Active Users (DAU)
Core Feature Completion	95% reliability for messaging	Automated test coverage >80%
Platform Performance	<500ms message delivery latency (p99)	APM monitoring (Datadog/New Relic)
Uptime	99.5% (43 min downtime/month max)	Status page + monitoring alerts
User Satisfaction	NPS >40	Post-launch survey
Development Velocity	Feature complete in 4 weeks	Sprint burndown tracking

Key Capabilities

- **Single source of truth** for team communication
- **Searchable message history** (48h MVP → 30/90 days)
- **Async-first design** (perfect for remote teams)
- **Self-hosted option** (data privacy + cost control)
- **60% lower cost** vs Slack (~$3-5/user/month)

Vision Statement:
"Build a focused, performant, and developer-friendly team communication hub that replaces email and fragmented chat tools, enabling seamless async-first collaboration at a fraction of traditional platform costs."

2. Problem Statement

Identified Problem

Geographically dispersed software teams face:

✕ **Fragmented communication** (email, expensive Slack)
✕ **Poor searchability** and knowledge retention
✕ **Excessive context-switching**
✕ **High platform costs** $8-15/user/month

ChatFlow Solution

Problem	ChatFlow Solution
Fragmented communication	Single platform for all conversations
Scarce searchability	Full-text search with filters (keyword, author, channel, date)
Excessive context-switching	Channel-based organization + async-first design
High costs	Self-hosted ($3-5/user/month vs. Slack $10-15)
Data privacy concerns	On-premise deployment option
Knowledge loss	48h message history (MVP) → 90-day retention (v1.1)

3. Vision & Strategic Objectives

Real-Time Dashboards (Grafana)

1. User Engagement

- Daily Active Users (DAU): 50+
- Message send rate: 100+ msg/day
- Average session duration: 30+ min

2. Technical Performance

- API response time (p95): <150ms
- WebSocket latency (p99): <500ms
- Database query time (p95): <100ms
- Platform uptime: 99.5%+

3. Quality Metrics

- Error rate (<1%)
- Test coverage (>80%)
- Security audit (0 critical CVEs)

4. Adoption & Satisfaction

- Feature adoption rate: >80%
- NPS score: 40+
- Support ticket volume: <5/week

4. Target Users & Personas

Persona 1: Alex (Development Team Lead)

Demographics: 28-35 years old, Tech Lead at 20-person startup

Pain Points:

- Team scattered across 3 time zones
- Slack costs $200/month (20 users × $10)
- Knowledge loss between messages

Goals:

- Reduce platform costs
- Improve team alignment
- Maintain audit trail

Usage Pattern: 4-6 hours/day on platform
Tech Proficiency: Very High (developers)

Persona 2: Sarah (Product Manager)

Demographics: 26-40 years old, PM at early-stage SaaS

Pain Points:

- Complex workflows in Slack
- Difficult to track decisions
- Limited integrations

Goals:

- Collaboration tool with structured workflows
- Searchable decision history
- Integration with Jira/GitHub

Usage Pattern: 3-4 hours/day
Tech Proficiency: Medium-High

Persona 3: Marcus (System Administrator)

Demographics: 35-50 years old, IT/Ops lead at SMB

Pain Points:

- Compliance requirements (GDPR, SOC2)
- Data residency concerns
- User management overhead

Goals:

- Self-hosted option (data privacy)
- Granular access controls
- Audit logging 90+ days

Usage Pattern: 1-2 hours/day (admin tasks)
Tech Proficiency: Very High (infrastructure)

5. Core Features - MVP Scope

Tier 1: MUST HAVE (P0 - Critical Path)

5.1 Authentication & User Management

Signup & Login:

- Self-service signup via email/password
- Email verification (24h link validity)
- OAuth integration (Google, GitHub) – v1.0 optional
- Password requirements: 8+ chars, 1 uppercase, 1 number, 1 special char
- Email/password login
- Session management (JWT tokens, 24h expiry)
- Refresh token support (30-day mobile sessions)
- Rate limiting: 5 failed attempts → 15min lockout

User Profiles:

- Avatar upload + crop
- Display name customization
- Bio (max 200 chars)
- Status setting (online/away/offline)
- Timezone per user

5.2 Workspace Management

Workspace Creation & Management:

- Create workspace (name + optional description)
- Unique workspace slug (URL-safe identifier)
- Creator becomes owner
- Default channels auto-created: #general, #random, #announcements
- Email-based invitations (bulk up to 50 per day)

Member Management:

- Invite link valid 7 days
- Auto-join if already registered
- Redirect to signup if new user
- View member list with roles
- Role types:
  - **Owner** (unlimited power)
  - **Admin** (create channels, manage users)
  - **Moderator** (manage own channels)
  - **Member** (basic)
- Promote/demote members
- Remove member (immediate access revocation)

5.3 Channel Management

Channel Features:

- Public or private channels
- Channel name (3-50 chars, unique per workspace)
- Channel description (optional, max 200 chars)
- Auto-generated slug
- Creator becomes moderator

Channel Operations:

- Public channels: join anytime
- Private channels: invite-only
- Leave channel anytime (except #general is mandatory)
- Archive channel (read-only, hidden from list)
- Soft-delete channel (preserves history)
- List all public channels
- Search channels by name/description
- Show member count + activity level
- Sorting: alphabetical, most active, recent

5.4 Messaging Engine (Core)

Message Composition:

- Type message (max 4,000 chars)
- Markdown support: bold, italic, code, code block
- @mention users (autocomplete dropdown)
- Send via Enter key or Send button

Message Delivery & Storage:

- Real-time delivery (<500ms latency, p99)
- Store in PostgreSQL (indexed for search)
- Timestamp: server-generated (UTC ISO 8601)
- Message ID: UUID (immutable reference)
- 48-hour history window (MVP)

Message Management:

- Edit own message within 1 hour of send
- Shows "Edited" label with timestamp
- Edit history tracked (immutable log)
- Edit broadcast to all channel members in real-time
- Author can delete anytime
- Soft-delete (marked as deleted, hidden in UI)
- Soft-deleted content preserved for compliance

5.5 Direct Messaging

1-on-1 DM:

- Start DM from user profile
- Persistent conversation thread
- Same feature parity as channels (edit, react, delete)
- Typing indicators
- Online/offline status

Group DM:

- Create DM with 3+ users
- Shared conversation history
- Member list
- Leave group DM (remains accessible to others)

5.6 Search & Discovery

Search Features:

- Search by keyword in all messages
- Filter by author (from:@user)
- Filter by channel (in:#channel)
- Filter by date range (before:2025-11-19, after:2025-11-15)
- Combine filters
- Results: 20 per page, <2s response time

Search Results Display:

- Author, channel, timestamp, 100-char snippet
- Highlight matching keywords
- Click → navigate to message in context

5.7 Notifications

In-App Notifications:

- Toast notifications for @mentions
- Unread message badges
- Typing indicators (other users typing)
- Click notification → jump to message
- Enable/disable per channel

Browser Notifications (Optional v1.0):

- Push notification for @mentions
- Requires user opt-in

5.8 File Sharing

File Upload & Management:

- Upload documents/images (max 10MB per file)
- Organize by channel/DM
- File metadata: name, size, upload time, uploader
- Delete file (removes from storage)

File Preview:

- Image inline preview
- Document link with file type icon
- Size displayed
- Download link

Tier 2: SHOULD HAVE (P1 - High Priority)

✅ Message threading (replies to specific message)
✅ Channel pinned messages
✅ User presence (last seen timestamp)
✅ Admin dashboard (activity, user stats)
✅ Message reactions (expanded emoji support)
✅ Rich text formatting (tables, lists in Markdown)
✅ Link previews (title + description)
✅ Mobile-responsive UI (tested on tablet)

Tier 3: COULD HAVE - Out of Scope (v1.1+)

✖ Voice/video calling
✖ Screen sharing
✖ Custom bots & workflows
✖ End-to-end encryption
✖ Native mobile apps (iOS/Android)
✖ Advanced AI features (search suggestions, auto-summarize)
✖ Enterprise SSO (SAML/OAuth2 provider)

6. User Stories & Use Cases

Epic 1: Onboarding & Authentication

Story 1.1: User Sign Up

As a new team member
I want to sign up with email and password
So that I can access ChatFlow and join my team

Acceptance Criteria:
✓ Sign up form: email, password, name, confirm password
✓ Client-side validation: email format, password strength
✓ Server-side validation: duplicate email check
✓ Confirmation email sent within 5 seconds
✓ Email link valid for 24 hours
✓ Click link → account created → auto-login
✓ Error messages: "Invalid email", "Email already exists", "Password too weak"
✓ Can resend email (max 5x per hour)

QA Acceptance Criteria:
✓ Load test: 100 concurrent signups
✓ Performance: signup form <2 seconds
✓ Email delivery: >99% success rate
✓ Security: no plaintext passwords in logs
✓ XSS prevention: test HTML injection in name

Story 1.2: User Login

As an existing user
I want to log in with email and password
So that I can access my workspace

Acceptance Criteria:
✓ Login form: email, password
✓ Correct credentials → session created → redirect to workspace
✓ Incorrect credentials → generic error (no email leak)
✓ Rate limiting: 5 failed attempts → 15min lockout
✓ JWT token issued (24h expiry)
✓ Remember me: optional (30-day refresh token)
✓ Logout: session invalidated

Edge Cases:
✓ Email not verified → error "Verify email first"
✓ Account locked → error "Try again in 15 minutes"
✓ Token expired → auto-redirect to login

Epic 2: Workspace & Channel Management

Story 2.1: Create Workspace

As a first-time user
I want to create a new workspace
So that my team can join and communicate

Acceptance Criteria:
✓ Workspace creation form: name (2-50 chars), optional description
✓ Unique workspace slug generated (handle duplicates with -1, -2, etc.)
✓ Creator becomes owner (admin role)
✓ Default channels auto-created: #general, #random, #announcements
✓ Creator auto-joined to all default channels
✓ Workspace URL: https://chatflow.app/ws/{slug}
✓ Success message: "Workspace created! Invite your team."

Performance:
✓ Workspace creation: <500ms
✓ Database: 1 workspace + 3 channels + 1 user inserted

Story 2.2: Invite Team Members

As workspace owner
I want to invite team members via email
So that they can join the workspace

Acceptance Criteria:
✓ Invite form: email address(es)
✓ Single invite or bulk (up to 50 per day)
✓ Invitation email sent within 5 seconds
✓ Email includes: workspace name, sender name, join link
✓ Invite link: unique, valid 7 days
✓ Click link:

- If recipient registered → auto-join workspace
- If not registered → redirect to signup (email pre-filled, workspace auto-selected)
  ✓ Invitation status visible to owner: pending, accepted, expired
  ✓ Can resend invite if expired

Edge Cases:
✓ Duplicate invite → show "Invite pending"
✓ Already team member → show "Already a member"
✓ Invalid email → highlight error
✓ Email delivery fails → notification to owner

Epic 3: Messaging (Core)

Story 3.1: Send Message in Channel

As a team member
I want to send messages in channels
So that I can communicate with my team

Acceptance Criteria (Functional):
✓ Message input field in channel
✓ Type message (max 4,000 chars)
✓ Send via Enter key or Send button
✓ Message appears immediately (optimistic UI)
✓ Markdown formatting: **bold**, *italic*, `code`, `block code`
✓ @mention autocomplete (type @ → dropdown of team members)
✓ Message persisted to database
✓ Timestamp: server-generated (UTC)
✓ Other channel members see message in real-time (<500ms)

Acceptance Criteria (Non-Functional):
✓ P50 latency: <100ms
✓ P95 latency: <300ms
✓ P99 latency: <500ms

✓ Throughput: 100+ msg/sec (load test)
✓ No message loss (ack from server)

Edge Cases:
✓ Empty message → rejected "Message cannot be empty"
✓ >4,000 chars → rejected "Message too long"
✓ Network disconnect → retry mechanism
✓ Duplicate send prevention (idempotency key)

Story 3.2: Search Messages

As a team member
I want to search messages
So that I can find past conversations

Acceptance Criteria:
✓ Search box in header (global)
✓ Type keyword → real-time results (dropdown)
✓ Advanced syntax:

- from:@user → filter by author
- in:#channel → filter by channel
- before:2025-11-19 → filter by date
- after:2025-11-15 → filter by date
- "exact phrase" → exact match
  ✓ Results sorted by relevance
  ✓ Pagination: 20 per page
  ✓ Snippet: 100 chars with highlight
  ✓ Click result → jump to message in channel

Performance:
✓ Search latency: <2 seconds (p95)
✓ Support: 100 concurrent searches
✓ Index freshness: <5 seconds lag

7. Technical Architecture (Lab Deployment)

7.1 Tech Stack - Lab Specific

Frontend:

- Framework: React 19 + TypeScript
- State: Zustand
- Real-time: Socket.IO client
- Styling: Tailwind CSS
- Build: Vite
- Deploy: Static files → Nginx

Backend:

- Runtime: Node.js 24.11.1 (LTS - REQUIRED)
- Framework: Express.js or Fastify
- Language: TypeScript
- Real-time: Socket.IO server
- Port: 4000 (as specified)

Database:

- Primary: PostgreSQL 15+ (on same VM)
- Cache: Redis (optional, for session store)
- Search: Elasticsearch (optional, can use DB FTS)

Infrastructure (Lab):

- Host: slackteam.lab.home.lucasacchi.net
- SSH: slackteam / slackteam123
- Frontend Port: 8282 (Nginx reverse proxy)
- Backend Port: 4000 (Node.js)
- OS: Linux (Ubuntu/Debian assumed)

7.2 Architecture Diagram (Lab Setup)

```
┌─────────────────────────────────────────────────────┐
│           User Browser (Remote/Local)              │
│  http://slackteam.lab.home.lucasacchi.net:8282     │
└──────────────────────────┬──────────────────────────┘
                         │ HTTP/HTTPS
                         ↓
        ┌─────────────────────────────┐
        │   Nginx (Port 8282)        │
        │   (Reverse Proxy)          │
        │   ├─ Static: /app           │
        │   └─ API Proxy: /api → :4000
        └─────────────────┬───────────┘
                      │
        ┌─────────────────┴─────────────────┐
        │                            │
        ↓                            ↓
    ┌─────────────┐          ┌──────────────────┐
    │ React App   │          │ Express/Fastify  │
    │ (Port 8282) │          │ (Port 4000)      │
    │             │          │ Node.js 24.11.1  │
    │ • UI        │          │                  │
    │ • WebSocket │          │ • REST API       │
    │ • State     │          │ • WebSocket Svr  │
    │ • Auth      │          │ • Business Logic │
    └─────────────┘          └────────────┬───────────┘
                                 │
              ┌─────────────────┼─────────────────┐
              │                  │              │
              ↓                  ↓              ↓
       ┌───────────────┐   ┌────────────┐  ┌─────────────────┐
       │  PostgreSQL   │   │Redis Cache │  │ File Storage │
       │  (Messages,   │   │(Sessions,  │  │  (S3 or      │
       │   Users,      │   │ Presence)  │  │   Local /tmp)│
       │   Channels)   │   └────────────┘  └──────────────┘
       └───────────────┘
```

7.3 Deployment Steps (Lab)

Step 1: VM Setup

```
# SSH into lab VM<a></a>
ssh slackteam@slackteam.lab.home.lucasacchi.net
# Password: slackteam123<a></a>

# Verify Node.js version<a></a>
node -v  # Expected: v24.11.1
python3 --version  # Expected: 3.11.2

# Install system dependencies<a></a>
sudo apt update
sudo apt install -y postgresql postgresql-contrib redis-server nginx git

# Start services<a></a>
sudo systemctl start postgresql redis-server nginx
sudo systemctl enable postgresql redis-server nginx
```

Step 2: Database Setup

```
# Create PostgreSQL database<a></a>
sudo -u postgres psql &lt;&lt; EOF
CREATE DATABASE chatflow_dev;
CREATE USER chatflow WITH PASSWORD 'dev_password_secure';
ALTER ROLE chatflow SET client_encoding TO 'utf8';
ALTER ROLE chatflow SET default_transaction_isolation TO 'read committed';
ALTER ROLE chatflow SET default_transaction_deferrable TO on;
ALTER ROLE chatflow SET default_transaction_read_only TO off;
GRANT ALL PRIVILEGES ON DATABASE chatflow_dev TO chatflow;
EOF

# Test connection<a></a>
psql -U chatflow -d chatflow_dev -h localhost
```

Step 3: Backend Deployment

```
# Clone repository<a></a>
git clone https://github.com/your-repo/chatflow.git ~/chatflow
cd ~/chatflow

# Install dependencies<a></a>
npm install

# Create .env file<a></a>
cat &gt; .env &lt;&lt; EOF
NODE_ENV=development
NODE_VERSION=24.11.1
PORT=4000
DB_HOST=localhost
DB_USER=chatflow
DB_PASSWORD=dev_password_secure
DB_NAME=chatflow_dev
REDIS_URL=redis://localhost:6379
JWT_SECRET=your-secret-key-change-this
EOF

# Run migrations<a></a>
npm run migrate:up

# Start backend<a></a>
npm start  # Runs on port 4000

# Or use PM2 for process management:<a></a>
npm install -g pm2
pm2 start npm --name chatflow -- start
```

Step 4: Frontend Deployment

```
# Build React app<a></a>
cd ~/chatflow/frontend
npm install
npm run build  # Outputs to dist/

# Copy to Nginx<a></a>
sudo cp -r dist/* /var/www/html/chatflow/
```

Step 5: Nginx Configuration

```
# OR: Setup Nginx config for SPA routing<a></a>
sudo nano /etc/nginx/sites-available/chatflow

server {
    listen 8282;
    server_name slackteam.lab.home.lucasacchi.net;

    # Frontend static files
    location / {
        root /var/www/html/chatflow;
        index index.html;
        try_files $uri $uri/ /index.html;  # SPA routing
    }

    # API proxy to backend
    location /api/ {
        proxy_pass http://localhost:4000/;
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection "Upgrade";
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
    }

    # WebSocket support
    location /socket.io {
        proxy_pass http://localhost:4000/socket.io;
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection "Upgrade";
        proxy_buffering off;
        proxy_cache_bypass $http_upgrade;
    }
}
```

Step 6: Verify Deployment

```
# Test frontend<a></a>
curl http://slackteam.lab.home.lucasacchi.net:8282

# Test backend<a></a>
curl http://slackteam.lab.home.lucasacchi.net:8282/api/health

# Monitor logs<a></a>
tail -f /var/log/nginx/access.log
tail -f ~/.pm2/logs/chatflow-out.log
```

8. Database Schema (PostgreSQL)

Core Tables

```
-- Users table
CREATE TABLE users (
    id UUID PRIMARY KEY,
    email VARCHAR(255) UNIQUE NOT NULL,
    password_hash VARCHAR(255),  -- bcrypt hash
    display_name VARCHAR(100) NOT NULL,
    avatar_url VARCHAR(500),
    bio TEXT,
    timezone VARCHAR(50) DEFAULT 'UTC',
    status VARCHAR(20) DEFAULT 'offline',  -- online, away, offline
    email_verified BOOLEAN DEFAULT false,
    created_at TIMESTAMP DEFAULT NOW(),
    updated_at TIMESTAMP DEFAULT NOW(),
    last_login TIMESTAMP
);

-- Workspaces table
CREATE TABLE workspaces (
    id UUID PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    slug VARCHAR(100) UNIQUE NOT NULL,
    description TEXT,
    owner_id UUID NOT NULL REFERENCES users(id),
    plan VARCHAR(20) DEFAULT 'free',  -- free, pro, enterprise
    member_limit INT DEFAULT 30,
    member_count INT DEFAULT 1,
    created_at TIMESTAMP DEFAULT NOW(),
    updated_at TIMESTAMP DEFAULT NOW()
);

-- User-Workspace membership
CREATE TABLE user_workspace_members (
```

```sql
    id UUID PRIMARY KEY,
    workspace_id UUID NOT NULL REFERENCES workspaces(id),
    user_id UUID NOT NULL REFERENCES users(id),
    role VARCHAR(20) DEFAULT 'member',  -- owner, admin, moderator, member
    joined_at TIMESTAMP DEFAULT NOW(),
    status VARCHAR(20) DEFAULT 'active',  -- active, invited, left
    UNIQUE(workspace_id, user_id)
);

-- Channels table
CREATE TABLE channels (
    id UUID PRIMARY KEY,
    workspace_id UUID NOT NULL REFERENCES workspaces(id),
    name VARCHAR(80) NOT NULL,
    slug VARCHAR(80) NOT NULL,
    type VARCHAR(20) DEFAULT 'public',  -- public, private
    description TEXT,
    created_by UUID NOT NULL REFERENCES users(id),
    created_at TIMESTAMP DEFAULT NOW(),
    archived BOOLEAN DEFAULT false,
    deleted BOOLEAN DEFAULT false,
    deleted_at TIMESTAMP,
    message_count INT DEFAULT 0,
    UNIQUE(workspace_id, slug)
);

-- Channel members
CREATE TABLE channel_members (
    id UUID PRIMARY KEY,
    channel_id UUID NOT NULL REFERENCES channels(id),
    user_id UUID NOT NULL REFERENCES users(id),
    role VARCHAR(20) DEFAULT 'member',  -- moderator, member
    joined_at TIMESTAMP DEFAULT NOW(),
    UNIQUE(channel_id, user_id)
);

-- Messages table
CREATE TABLE messages (
    id UUID PRIMARY KEY,
    channel_id UUID NOT NULL REFERENCES channels(id),
    user_id UUID NOT NULL REFERENCES users(id),
    content TEXT NOT NULL,
    thread_id UUID REFERENCES messages(id),  -- For threaded replies
    edited_at TIMESTAMP,
    deleted_at TIMESTAMP,
    created_at TIMESTAMP DEFAULT NOW(),
    updated_at TIMESTAMP DEFAULT NOW(),
    INDEX idx_messages_channel_created (channel_id, created_at DESC),
    INDEX idx_messages_thread (thread_id)
);

-- Message edit history
CREATE TABLE message_edit_history (
    id UUID PRIMARY KEY,
    message_id UUID NOT NULL REFERENCES messages(id),
    previous_content TEXT NOT NULL,
    new_content TEXT NOT NULL,
    edited_by UUID NOT NULL REFERENCES users(id),
    edited_at TIMESTAMP DEFAULT NOW()
);

-- Reactions
CREATE TABLE reactions (
    id UUID PRIMARY KEY,
    message_id UUID NOT NULL REFERENCES messages(id),
    user_id UUID NOT NULL REFERENCES users(id),
    emoji VARCHAR(10) NOT NULL,
    created_at TIMESTAMP DEFAULT NOW(),
    UNIQUE(message_id, user_id, emoji)
);

-- Direct messages
CREATE TABLE direct_messages (
    id UUID PRIMARY KEY,
    sender_id UUID NOT NULL REFERENCES users(id),
    recipient_id UUID NOT NULL REFERENCES users(id),
    content TEXT NOT NULL,
    edited_at TIMESTAMP,
    deleted_at TIMESTAMP,
    created_at TIMESTAMP DEFAULT NOW(),
    INDEX idx_dms_participants (sender_id, recipient_id, created_at DESC)
);

-- Files
CREATE TABLE files (
    id UUID PRIMARY KEY,
    message_id UUID REFERENCES messages(id),
    dm_id UUID REFERENCES direct_messages(id),
    filename VARCHAR(255) NOT NULL,
    file_size INT NOT NULL,  -- bytes
    file_type VARCHAR(50),  -- MIME type
    storage_path VARCHAR(500),  -- S3 path or local path
    uploaded_by UUID NOT NULL REFERENCES users(id),
    uploaded_at TIMESTAMP DEFAULT NOW(),
    deleted_at TIMESTAMP
);

-- Notifications
CREATE TABLE notifications (
    id UUID PRIMARY KEY,
    user_id UUID NOT NULL REFERENCES users(id),
    type VARCHAR(50),  -- mention, channel_activity, dm
    channel_id UUID REFERENCES channels(id),
    message_id UUID REFERENCES messages(id),
    actor_id UUID REFERENCES users(id),  -- Who triggered notification
    created_at TIMESTAMP DEFAULT NOW(),
    read BOOLEAN DEFAULT false,
    read_at TIMESTAMP
);

-- Audit logs
CREATE TABLE audit_logs (
    id UUID PRIMARY KEY,
    workspace_id UUID NOT NULL REFERENCES workspaces(id),
    actor_id UUID NOT NULL REFERENCES users(id),
    action VARCHAR(100),  -- user_created, channel_deleted, message_edited
    resource_type VARCHAR(50),  -- user, channel, message
    resource_id UUID,
    details JSONB,  -- Additional context
    created_at TIMESTAMP DEFAULT NOW(),
    INDEX idx_audit_logs_workspace_created (workspace_id, created_at DESC)
);
```

Indexes for Performance

```
-- Message retrieval (critical path)
CREATE INDEX idx_messages_channel_created ON messages(channel_id, created_at DESC);
CREATE INDEX idx_messages_user ON messages(user_id, created_at DESC);

-- Search (full-text, if not using Elasticsearch)
CREATE INDEX idx_messages_content_fts ON messages USING GIN(to_tsvector('english', content));

-- Workspace queries
CREATE INDEX idx_user_workspaces ON user_workspace_members(user_id, workspace_id);
CREATE INDEX idx_workspace_channels ON channels(workspace_id);

-- DM queries
CREATE INDEX idx_direct_messages_pair ON direct_messages(sender_id, recipient_id);
```

9. API Specification (REST + WebSocket)

REST Endpoints

Authentication

```
POST   /api/auth/register
POST   /api/auth/login
POST   /api/auth/logout
POST   /api/auth/refresh
GET    /api/auth/me
```

Users

```
GET    /api/users/:id
PUT    /api/users/me
GET    /api/users  # List workspace members
```

Workspaces

```
POST   /api/workspaces
GET    /api/workspaces
GET    /api/workspaces/:id
PUT    /api/workspaces/:id
POST   /api/workspaces/:id/invite  # Send invitations
GET    /api/workspaces/:id/members
```

Channels

```
POST   /api/channels
GET    /api/channels  # List channels in workspace
GET    /api/channels/:id
PUT    /api/channels/:id
DELETE /api/channels/:id
POST   /api/channels/:id/members
DELETE /api/channels/:id/members/:userId
```

Messages

```
POST   /api/channels/:id/messages  # Send message
GET    /api/channels/:id/messages  # Get messages (paginated)
PUT    /api/messages/:id  # Edit message
DELETE /api/messages/:id  # Delete message
POST   /api/messages/:id/reactions  # Add reaction
GET    /api/search?q=keyword  # Search messages
```

Direct Messages

```
POST   /api/dms  # Create or get DM
GET    /api/dms  # List DM conversations
GET    /api/dms/:id/messages
POST   /api/dms/:id/messages  # Send DM message
```

WebSocket Events

Event	Direction	Payload	Latency
message:send	Client → Server	{content, channel_id}	-
message:received	Server → Clients	{message, channel_id}	<500ms
typing:start	Client → Server	{user_id, channel_id}	<100ms
typing:stop	Client → Server	{user_id, channel_id}	<100ms
presence:update	Client → Server	{user_id, status}	<100ms

Event	Direction	Payload	Latency
presence:broadcast	Server → Clients	{user_id, status}	<500ms
reaction:add	Client → Server	{message_id, emoji}	<200ms
message:edited	Server → Clients	{message_id, new_content}	<300ms
message:deleted	Server → Clients	{message_id}	<300ms

10. Non-Functional Requirements

Performance Requirements

Message Send Latency (from click to server ack):

- P50: <100ms
- P95: <300ms
- P99: <500ms
- Throughput: 100+ msg/sec at 500 concurrent users

API Response Time:

- GET /messages: P95 <100ms (50 messages, paginated)
- POST /messages: P95 <150ms
- GET /search: P95 <2 seconds
- All other endpoints: P95 <100ms

Database Performance:

- Query time (p95): <50ms
- Connection pool: 20 connections (single VM)
- Cache hit rate: 90%+ for frequently accessed data

File Upload/Download:

- Upload: <5 seconds for 10MB file
- Download: CDN or local storage <1 second

Scalability

MVP Phase:

- Concurrent users: 50-200
- Daily active users: 100+
- Message throughput: 10K-100K msg/day
- Database size: <10GB (should fit on VM)

Architecture: Monolithic (single VM)

- PostgreSQL: 20GB recommended disk
- Redis: 2GB RAM (optional, for sessions)
- Application: Node.js (single process or PM2 cluster)
- Static files: Nginx

Security

Authentication:

- JWT tokens (HS256)
- Access token: 24h expiry
- Refresh token: 30-day expiry (secure httpOnly cookie)
- Password: bcrypt cost factor 12

Transport:

- HTTPS/TLS 1.3 (certificate: Let's Encrypt or self-signed for lab)
- HSTS: 1-year max-age

Data Protection:

- At-rest: AES-256 for sensitive fields (passwords never stored plaintext)
- In-transit: TLS 1.3
- Audit logs: immutable append-only logs

Access Control:

- RBAC: owner > admin > moderator > member
- Channel-level ACL
- Workspace-level permissions

Reliability & Uptime

Target Uptime: 99.5% (43 min downtime/month max)

Monitoring:

- Uptime monitoring: UptimeRobot or similar
- Error tracking: Sentry or logs
- Performance monitoring: PM2 Plus or New Relic Lite

Backup & Recovery:

- Daily PostgreSQL backups (automated)
- Point-in-time recovery: 7-day retention
- RTO: <1 hour
- RPO: <15 minutes

11. Release Timeline & Milestones

4-Week Sprint Plan (MVP)

Week 1: Backend Foundation

- Day 1-2: Setup DB schema, table creation, indexes
- Day 3-4: Auth API (signup, login, JWT)
- Day 5: User management API
- Deliverable: Authentication system 100% working

Week 2: Core Messaging

- Day 1-2: Channel API (create, list, join/leave)
- Day 3-4: Message API (send, retrieve, pagination)
- Day 5: WebSocket real-time messaging
- Deliverable: Send/receive messages in channels

Week 3: Frontend + Enhancement

- Day 1-2: React UI (login, workspace, channels)

- Day 3-4: Message UI (send, display, real-time)
- Day 5: Search, DM, notifications
- Deliverable: Full-featured MVP

Week 4: Testing, Optimization, Deployment

- Day 1-2: Load testing (100+ concurrent users)
- Day 3: Performance optimization
- Day 4: Security audit + penetration testing
- Day 5: Deploy to lab, final testing
- Deliverable: Production-ready on lab.home

Key Dates

- Week 1 End: Backend auth + database ready (Milestone 1)
- Week 2 End: Core messaging working (Milestone 2)
- Week 3 End: Frontend + UI complete (Milestone 3)
- Week 4 End: Production deployment on lab (Milestone 4)

12. Success Criteria (QA Acceptance)

Functional Acceptance

- [ ] User can signup + verify email (end-to-end)
- [ ] User can login + logout
- [ ] Create workspace + invite 3+ team members
- [ ] Send message in channel (real-time display)
- [ ] Edit message (1-hour window works)
- [ ] Delete message (soft-delete hidden in UI)
- [ ] Search messages (filter by keyword, author, channel, date)
- [ ] 1-on-1 DM conversation works
- [ ] @mentions trigger notifications
- [ ] Emoji reactions work
- [ ] File upload/download works
- [ ] Channel archive/delete works
- [ ] User presence (online/offline) works

Performance Acceptance

- [ ] Message latency <500ms (p99) under 50 concurrent users
- [ ] API response time <150ms (p95) for all endpoints
- [ ] Page load time <2 seconds
- [ ] WebSocket connection establishes <1 second
- [ ] Search returns results <2 seconds

Security Acceptance

- [ ] No plaintext passwords in logs/DB
- [ ] XSS prevention (sanitize all user input)
- [ ] SQL injection prevention (parameterized queries)
- [ ] CSRF tokens on state-changing operations
- [ ] JWT tokens cannot be forged
- [ ] Rate limiting works (5 failed logins → lockout)

Reliability Acceptance

- [ ] Uptime: 99.5%+ over 7 days
- [ ] No message loss (all messages persisted)
- [ ] Database backup works (manual test)
- [ ] Point-in-time recovery works (manual test)
- [ ] Error logging comprehensive (all errors captured)

13. Known Limitations & Future Work

MVP Limitations

Limitation	Impact	v1.1 Plan
48-hour message history	Can't find older messages	Extend to 30/90 days
No mobile native app	Limited mobile access	iOS/Android apps
No video/voice calling	Limited rich communication	WebRTC integration
No message encryption	Privacy concern	E2E encryption option
No custom bots/workflows	Limited automation	Bot framework
Single VM deployment	Limited scalability	Kubernetes cluster
Monolithic architecture	Harder to scale independently	Microservices (later)

Roadmap (v1.1 → v2.0)

v1.1 (Q1 2026):
✅ Multi-workspace support (users can be in multiple workspaces)
✅ Message threading (nested replies)
✅ API & webhooks (custom integrations)
✅ Zapier integration
✅ Admin dashboard (analytics, user management)
✅ Extended message history (90 days)

v1.2 (Q2 2026):
✅ Native iOS/Android apps
✅ Voice/video calling (WebRTC)
✅ Screen sharing
✅ End-to-end encryption option
✅ Enterprise SSO (SAML/OAuth)
✅ SOC2 compliance

v2.0 (Q3-Q4 2026):
✅ AI-powered search (semantic)
✅ Auto-summarization
✅ Custom bots & workflows
✅ Advanced analytics
✅ White-label option

14. Sign-Off & Approvals

Stakeholder Review & Approval

Role	Name	Approval	Date
Product Manager	[To be assigned]	☐	____
Engineering Lead	[To be assigned]	☐	____
Architect	[To be assigned]	☐	____
DevOps/Infrastructure	slackteam	☐	____
QA Lead	[To be assigned]	☐	____

15. Appendices

Appendix A: Glossary

Term	Definition
DAU	Daily Active Users
MVP	Minimum Viable Product
RBAC	Role-Based Access Control
JWT	JSON Web Token
WebSocket	Persistent bidirectional communication protocol
Soft Delete	Mark as deleted without removing from DB
Idempotency	Operation safe to retry without side effects
P50/P95/P99	50th/95th/99th percentile latency

Appendix B: Reference Documentation

- joelparkerhenderson/functional-specifications-template
- hellopm.co - How to Create a PRD
- ChatFlow Functional Analysis Document
- Node.js 24.11.1 Documentation
- PostgreSQL 15 Documentation

Document Status: Final for Development & Lab Deployment
Last Updated: November 19, 2025
Next Review: Upon MVP Milestone 1 completion (Week 1 end)
[1]

⁂

1. ChatFlow_PRD_v2_Lab-Optimized.pdf