Technical Design Document (TDD)

Table of Contents

ChatFlow MVP - Team Communication Platform

VERSION 1.0 - LAB OPTIMIZED
Language: English
Date: November 19, 2025
Author: Senior Software Architect

Document Metadata

Attribute	Value
Project Name	ChatFlow MVP - Real-Time Team Communication Platform
Version	1.0 (Lab-Optimized)
Date	November 19, 2025
Status	Final for Development & Architecture Review
Author	Senior Software Architect
Audience	Development Team, DevOps Engineers, Technical Architects
Lab Environment	slackteam.lab.home.lucasacchi.net
Node.js	v24.11.1 (LTS - Production Ready)
Python	3.11.2
Template Base	NotePlan Technical Design Document Template
Base Documents	PRD v2.1 (English), FAD v2.0 (Lab-Optimized)

Table of Contents

1. Executive Summary

This Technical Design Document (TDD) specifies the complete technical architecture for **ChatFlow MVP**, a real-time team communication platform designed for deployment on lab infrastructure (slackteam.lab.home.lucasacchi.net).

Purpose & Scope

The TDD provides:

- ✅ Complete system architecture (components, interactions, data flow)
- ✅ Technology stack rationale (why each tech chosen)
- ✅ Data model & database schema (normalized, indexed for performance)
- ✅ API design specifications (REST + WebSocket)
- ✅ Security architecture (authentication, authorization, encryption)
- ✅ Performance design (latency targets, scalability roadmap)
- ✅ Deployment procedures (step-by-step, automated scripts)
- ✅ DevOps & monitoring infrastructure
- ✅ Testing & quality assurance strategy

Key Technical Decisions

Decision	Rationale	Impact
Node.js 24.11.1	LTS, proven for real-time apps, JavaScript full-stack	✅ Fast development, single language (backend + frontend)
React 19 + TypeScript	Modern UI framework, type safety, large ecosystem	✅ Faster development, fewer bugs, better DX
Express.js + Socket.IO	Industry standard, WebSocket support out-of-box	✅ Fast implementation, proven in production
PostgreSQL 15	ACID compliance, JSON support, powerful indexing	✅ Data integrity, real-time search, complex queries
Redis Cache	Sub-millisecond latency, session management	✅ Reduces DB load, improves response times
Monolithic on Single VM	MVP phase, 50-100 concurrent users target	✅ Simpler deployment, easier debugging, cost-effective

2. Architecture Overview & Design Goals

2.1 Design Goals (Priority Order)

Goal	Target	Rationale
Performance	<500ms message latency (p99)	Core UX requirement for real-time chat
Reliability	99.5% uptime (43 min/month)	Business SLA target

Goal	Target	Rationale
Security	Zero critical CVEs, RBAC enforcement	Protect user data, audit compliance
Scalability	50→200 concurrent users (phase 2 → 500+)	Support growth without architectural rewrite
Maintainability	<30% technical debt, modular code	Enable rapid feature development
Observability	All critical paths instrumented	Fast troubleshooting and debugging

2.2 High-Level Architecture Diagram

```
┌─────────────────────────────────────────────────────────────────────┐
│                    User Browsers                               │
│              (Web, Mobile, Desktop)                            │
└───────────────────────┬─────────────────────────────────────────────┘
                     │ HTTPS/WSS
                     ↓
        ┌──────────────────────────────────────────┐
        │    Nginx Reverse Proxy (Port 8282) │
        │  (Load Balancer, SSL Termination) │
        ├──────────────────────────────────────────┤
        │ • Static files: /app → React SPA  │
        │ • API routes: /api/* → Node.js    │
        │ • WebSocket: /socket.io → Upgrade│
        └─────┬──────────────────┬────────────────┘
             │                 │
        ┌─────▼──────┐   ┌──────▼──────────┐
        │  React 19  │   │  Express.js    │
        │ TypeScript │   │  + Socket.IO   │
        │ (Port 8282)│   │  (Port 4000)   │
        │            │   │  Node 24.11.1  │
        │ • UI State │   │                │
        │ • WebSocket│   │ • REST API     │
        │ • Auth UI  │   │ • WebSocket Sv │
        │ • Pages    │   │ • Business Log │
        │ • Forms    │   │ • Auth Handler │
        │ • Cache    │   │ • Rate Limiter │
        └──────────────┘   └─────┬──────────────┘
                                │
        ┌───────────────────────────┼─────────────────┐
        │                        │                  │
        ↓                        ↓                  ↓
    ┌───────────────┐  ┌─────────────────────┐  ┌──────────────┐
    │ PostgreSQL   │  │ Redis Cache      │  │ File Store │
    │ 15           │  │ (Sessions, User  │  │ (Local /   │
    │              │  │ Presence, Rate   │  │  tmp or S3)│
    │ • Users      │  │ Limit State)     │  │            │
    │ • Channels   │  │                  │  │ • Files    │
    │ • Messages   │  │ • &lt;5ms latency   │  │ • Avatars  │
    │ • Roles      │  │ • 90%+ hit rate  │  │ • Images   │
    │ • Audit      │  │ • 2GB RAM        │  │            │
    │ • Indexes    │  │ • Pub/Sub        │  │ Max 50MB   │
    │              │  │ • Session store  │  │            │
    └───────────────┘  └─────────────────────┘  └──────────────┘
         │                  ↑
         │                  │
         └─────────────────────┘
       (Connection Pooling)
```

2.3 Component Responsibilities

Component	Role	Key Responsibilities
Nginx	Gateway/Proxy	Reverse proxy, SSL termination, static file serving, WebSocket upgrade
React SPA	Frontend	UI rendering, user interaction, real-time updates, offline queue
Express.js	API Server	HTTP handlers, validation, business logic orchestration
Socket.IO	Real-Time	WebSocket connections, event pub/sub, presence management
PostgreSQL	Primary Data Store	Message persistence, user/workspace metadata, audit logs
Redis	Cache/Session	Session storage, user presence, rate limit counters, message cache
File Storage	Media Store	File uploads, avatar storage, backup

3. Technology Stack & Justification

3.1 Frontend Technology Stack

```
// package.json (React frontend)
{
  "dependencies": {
    "react": "^19.0.0",
    "react-dom": "^19.0.0",
    "react-router-dom": "^6.x",
    "zustand": "^4.x",              // State management (lightweight)
    "socket.io-client": "^4.x",     // WebSocket client
    "typescript": "^5.x",           // Type safety
    "axios": "^1.x",                // HTTP client
    "date-fns": "^2.x",             // Date utilities
    "react-markdown": "^8.x",       // Markdown rendering
    "emoji-picker-react": "^4.x"    // Emoji support
  },
  "devDependencies": {
    "@vitejs/plugin-react": "^4.x",
    "vite": "^5.x",                 // Build tool (faster than webpack)
    "tailwindcss": "^3.x",          // CSS framework
    "@testing-library/react": "^14.x",
    "vitest": "^1.x"                // Unit testing
  }
}
```

Justification:

Tech	Why Chosen	Alternative	Reason
React 19	Modern, component-based, large ecosystem	Vue 3, Svelte	React dominates enterprise, better hiring pool
TypeScript	Type safety, refactoring confidence	JavaScript	Reduces bugs, improves code quality
Zustand	Minimal boilerplate, performant state	Redux, Jotai	Simple API, solves 80% of cases
Socket.IO	Fallback to polling, well-tested	ws (native), Pusher	Handles connection issues gracefully
Vite	Fast HMR, fast build, modern	webpack, parcel	10-100x faster than webpack
Tailwind	Utility-first CSS, consistent design	Material-UI, Bootstrap	Smaller bundle size, faster development

3.2 Backend Technology Stack

```
// package.json (Node.js backend)
{
  "engines": {
    "node": "^24.11.1",
    "npm": "^10.x"
  },
  "dependencies": {
    "express": "^4.18.x",           // Web framework
    "express-async-errors": "^3.x", // Async/await error handling
    "socket.io": "^4.x",            // WebSocket server
    "uuid": "^9.x",                 // UUID generation
    "bcryptjs": "^2.4.x",           // Password hashing (cost 12)
    "jsonwebtoken": "^9.x",         // JWT tokens
    "pg": "^8.x",                   // PostgreSQL client
    "pg-pool": "^3.x",              // Connection pooling
    "redis": "^4.x",                // Redis client
    "cors": "^2.x",                 // Cross-origin requests
    "helmet": "^7.x",               // Security headers
    "compression": "^1.x",          // Gzip compression
    "dotenv": "^16.x",              // Environment variables
    "pino": "^8.x",                 // Structured logging
    "joi": "^17.x"                  // Input validation (schema validation)
  },
  "devDependencies": {
    "typescript": "^5.x",
    "ts-node": "^10.x",
    "@types/express": "^4.x",
    "@types/node": "^20.x",
    "jest": "^29.x",                // Unit testing
    "supertest": "^6.x",            // HTTP testing
    "nodemon": "^3.x",              // Development auto-reload
    "eslint": "^8.x",               // Code linting
    "prettier": "^3.x"              // Code formatting
  }
}
```

Justification:

Tech	Why Chosen	Alternative	Reason
Node.js 24.11.1	LTS, JavaScript, async I/O	Go, Python, Java	Fast development, full-stack JavaScript
Express.js	Minimal, middleware-based	Fastify, Koa, Nest	Simple API, excellent documentation, proven at scale
Socket.IO	Proven real-time, fallback transport	ws (native), Pusher	Handles network issues, great DX
PostgreSQL	ACID, JSON, powerful indexing	MongoDB, MySQL	Data integrity critical for messages
Redis	Sub-millisecond latency, Pub/Sub	Memcached, in-memory	Essential for real-time presence/session
Pino	Structured logging, performant	winston, bunyan	Low overhead, JSON output for parsing
JWT	Stateless, scalable, no server lookup	Session cookies	MVP focus: stateless architecture

3.3 Database & Infrastructure

```
# docker-compose.yml (for reference)<a></a>
version: '3.8'
services:
  postgres:
    image: postgres:15-alpine
    environment:
      POSTGRES_DB: chatflow_dev
      POSTGRES_USER: chatflow
      POSTGRES_PASSWORD: secure_password
    volumes:
      - postgres_data:/var/lib/postgresql/data
    ports:
      - "5432:5432"
    healthcheck:
      test: ["CMD-SHELL", "pg_isready -U chatflow"]
      interval: 10s
      timeout: 5s
      retries: 5

  redis:
    image: redis:7-alpine
    ports:
      - "6379:6379"
    healthcheck:
      test: ["CMD", "redis-cli", "ping"]
      interval: 10s
      timeout: 5s
      retries: 5
```

Component	Version	Choice Rationale	Performance Impact
PostgreSQL	15 (Alpine)	Latest stable, security patches	Supports 1M+ messages in <10GB
Redis	7 (Alpine)	Latest stable, Streams support	<5ms latency for cache hits
Nginx	1.25+	Modern, HTTP/2 support	Reduces connection overhead
Linux	Ubuntu 22.04+ LTS	Stable, long support window	Security patches, stability

4. System Architecture & Components

4.1 Layered Architecture

```
┌─────────────────────────────────────────────────────────────────┐
│                   Presentation Layer                        │
│         React SPA (Port 8282, Nginx Static)                │
│  • TypeScript Components                                    │
│  • Zustand State Management                                 │
│  • Socket.IO Real-Time Updates                             │
└─────────────────────────────────────────────────────────────────┘
                         │ HTTPS
                         ↓
┌─────────────────────────────────────────────────────────────────┐
│                  API Gateway Layer                           │
│  Nginx Reverse Proxy (Port 8282 → 4000)                    │
│  • SSL/TLS Termination                                     │
│  • Request Routing                                          │
│  • WebSocket Upgrade                                        │
│  • Rate Limiting (Nginx module)                            │
└─────────────────────────────────────────────────────────────────┘
                         │
        ┌────────────────┴──────────────────┐
        │                                  │
  REST API Routes                   WebSocket Routes
```

```
    (HTTP/1.1)                        (WebSocket Upgrade)
         │                                  │
         ↓                                  ↓
┌───────────────────────────────────────────────────────────────────┐
│              Application/Business Logic Layer             │
│                   Express.js (Port 4000)                 │
│  ┌──────────────────────────────────────────────────────────┐   │
│  │ Middleware Stack                                 │  │
│  │ • cors() - Cross-origin requests                │  │
│  │ • helmet() - Security headers                   │  │
│  │ • compression() - Gzip                          │  │
│  │ • authenticateToken() - JWT validation          │  │
│  │ • logger - Request/response logging             │  │
│  └──────────────────────────────────────────────────────────┘   │
│  ┌──────────────────────────────────────────────────────────┐   │
│  │ Route Handlers &amp; Controllers                    │  │
│  │ • /api/auth/* - Auth operations                 │  │
│  │ • /api/workspaces/* - Workspace mgmt            │  │
│  │ • /api/channels/* - Channel operations          │  │
│  │ • /api/messages/* - Message CRUD                │  │
│  │ • /api/search - Full-text search                │  │
│  │ • /health - Health check endpoint               │  │
│  └──────────────────────────────────────────────────────────┘   │
│  ┌──────────────────────────────────────────────────────────┐   │
│  │ Business Services                               │  │
│  │ • AuthService - JWT, password hashing           │  │
│  │ • MessageService - Message CRUD, validation     │  │
│  │ • ChannelService - Channel membership           │  │
│  │ • NotificationService - @mention handling       │  │
│  │ • SearchService - Full-text indexing            │  │
│  │ • RateLimitService - Token bucket algorithm     │  │
│  └──────────────────────────────────────────────────────────┘   │
│  ┌──────────────────────────────────────────────────────────┐   │
│  │ Socket.IO Real-Time Handler                     │  │
│  │ • message:send - Receive &amp; broadcast             │  │
│  │ • typing:start - Typing indicators              │  │
│  │ • presence:update - User status broadcast       │  │
│  │ • reaction:add - Emoji reactions                │  │
│  └──────────────────────────────────────────────────────────┘   │
└───────────────────────────────────────────────────────────────────┘
                         │
     ┌───────────────────┼───────────────────────┐
     │                   │                   │
     ↓                   ↓                   ↓
┌──────────────────┐ ┌────────────────┐ ┌───────────────────┐
│   PostgreSQL  │ │ Redis Cache  │ │ File Storage   │
│   Data Layer  │ │ (Session)    │ │ (Local /tmp)   │
│               │ │              │ │                │
│ • Connection  │ │ • GET/SET    │ │ • Upload       │
│   Pool (20)   │ │ • TTL expiry │ │ • Download     │
│ • Prepared    │ │ • Pub/Sub    │ │ • Delete       │
│   Statements  │ │              │ │                │
│ • Transactions│ │ &lt;5ms latency │ │ Max 50MB/file  │
└──────────────────┘ └────────────────┘ └───────────────────┘
```

4.2 Core Components & Responsibilities

Authentication Service

```
// src/services/AuthService.ts
class AuthService {
  // Signup with email verification
  async signup(email: string, password: string, displayName: string) {
    // 1. Validate inputs (email format, password strength)
    // 2. Check duplicate email (case-insensitive)
    // 3. Hash password with bcrypt (cost factor 12)
    // 4. Create user (email_verified = false)
    // 5. Generate JWT verification token (24h TTL)
    // 6. Send verification email via SendGrid
    // 7. Return success message
  }

  // Email verification
  async verifyEmail(token: string) {
    // 1. Decode JWT verification token
    // 2. Check token expiry
    // 3. Find user by email
    // 4. Update email_verified = true
    // 5. Return success
  }

  // Login with rate limiting
  async login(email: string, password: string) {
    // 1. Check rate limit (5 attempts → 15min lockout)
    // 2. Fetch user (case-insensitive email)
    // 3. Verify email_verified flag
    // 4. Compare password with bcrypt
    // 5. Generate JWT access token (24h)
    // 6. Generate JWT refresh token (30d)
    // 7. Store session in Redis cache
    // 8. Update last_login timestamp
    // 9. Return tokens + user info
  }

  // Refresh access token
  async refresh(refreshToken: string) {
    // 1. Verify refresh token signature
    // 2. Check expiry (30 days)
    // 3. Generate new access token (24h)
    // 4. Return new token
  }

  // Logout
  async logout(userId: string) {
    // 1. Invalidate session from Redis
    // 2. Blacklist refresh token (optional)
    // 3. Return success
  }
}
```

Security Implementation:

```
// Password hashing (bcrypt cost 12 = ~250ms)
const passwordHash = await bcrypt.hash(password, 12);
// bcrypt.compare(plaintext, hash) // ~250ms constant time

// JWT token structure
const accessToken = jwt.sign(
  {
    user_id: user.id,
    email: user.email,
    workspace_ids: [...],
    type: "access_token",
    iat: now,
    exp: now + 24 * 3600  // 24 hours
  },
  process.env.JWT_SECRET,
  { algorithm: "HS256" }
);

// Rate limiting with Redis (token bucket)
async function checkRateLimit(email: string, maxAttempts: number = 5) {
  const key = `login_fails:${email}`;
  const count = await redis.incr(key);

  if (count === 1) {
    await redis.expire(key, 900);  // 15 minutes
  }

  if (count &gt; maxAttempts) {
    throw new Error("Account locked for 15 minutes");
  }
}
```

Message Service (Real-Time)

```
// src/services/MessageService.ts
class MessageService {
  // Send message with real-time broadcast
  async sendMessage(
    userId: string,
    channelId: string,
    content: string,
    threadId?: string
  ) {
    // 1. Validate permission (user is channel member)
    // 2. Validate content (not empty, &lt;4KB)
    // 3. Sanitize content (escape HTML, preserve Markdown)
    // 4. Parse mentions (@username)
    // 5. Create message record (database transaction)
    // 6. Create notifications for @mentions
    // 7. Index message for search (async)
    // 8. Broadcast via WebSocket (&lt;500ms)
    // 9. Return message_id + timestamp
  }

  // Edit message (1-hour window)
  async editMessage(messageId: string, userId: string, newContent: string) {
    // 1. Fetch message
    // 2. Check author (only owner can edit)
    // 3. Check 1-hour window (created_at + 1h &gt; now)
    // 4. Validate new content
    // 5. Append to edit_history table
    // 6. Update message.edited_at timestamp
    // 7. Broadcast edit event via WebSocket
    // 8. Re-index in search
  }

  // Delete message (soft delete)
  async deleteMessage(messageId: string, userId: string) {
    // 1. Fetch message
    // 2. Check permission (author or moderator)
    // 3. Set deleted_at timestamp (soft delete)
    // 4. Broadcast delete event via WebSocket
    // 5. Return success
  }

  // Fetch messages (paginated)
  async getMessages(
    channelId: string,
    userId: string,
    page: number = 1,
    limit: number = 50
  ) {
    // 1. Check permission (user is channel member)
    // 2. Query messages with pagination
    // 3. Include sender info (display_name, avatar)
    // 4. Exclude soft-deleted messages
    // 5. Include reactions count
    // 6. Order by created_at DESC
    // 7. Return paginated result
  }
}
```

Search Service

```
// src/services/SearchService.ts
class SearchService {
  // Full-text search with filters
  async search(
    query: string,
    filters: SearchFilters
  ) {
    // Query: "keyword"
    // Filters: { from: "@user", in: "#channel", before, after }

    // Option 1: PostgreSQL Full-Text Search (MVP)
    const sql = `
      SELECT m.id, m.content, m.channel_id, m.user_id, m.created_at,
             u.display_name, c.name as channel_name,
             ts_rank(to_tsvector('english', m.content), query) as rank
      FROM messages m
      JOIN users u ON m.user_id = u.id
      JOIN channels c ON m.channel_id = c.id
      WHERE to_tsvector('english', m.content) @@ plainto_tsquery('english', ?)
        AND m.deleted_at IS NULL
        AND c.workspace_id = ?
        AND (? IS NULL OR m.user_id = ?)  -- from: filter
        AND (? IS NULL OR m.channel_id = ?)  -- in: filter
        AND (? IS NULL OR m.created_at &gt; ?)  -- after: filter
        AND (? IS NULL OR m.created_at &lt; ?)  -- before: filter
      ORDER BY rank DESC
      LIMIT ? OFFSET ?
    `;

    // Option 2: Elasticsearch (v1.1)
    // await elasticsearch.search({
    //   index: "messages",
    //   query: {
    //     bool: {
    //       must: [
    //         { multi_match: { query, fields: ["content^2", "display_name"] } },
    //         { term: { workspace_id } }
    //       ],
    //       filter: [
    //         { term: { deleted: false } },
    //         ...(filters.from ? [{ term: { user_id: filters.from } }] : []),
    //         ...(filters.channel ? [{ term: { channel_id: filters.channel } }] : []),
    //         ...(filters.after ? [{ range: { created_at: { gte: filters.after } } }] : [])
    //       ]
    //     }
    //   },
    //   from: (page - 1) * limit,
    //   size: limit
    // })
  }

  // Index message (called async after message creation)
  async indexMessage(message: Message) {
    // PostgreSQL: automatic via to_tsvector trigger
    // Elasticsearch: POST /messages/_doc/{ message }
  }
}
```

5. Data Structures & Database Design

5.1 Entity-Relationship Diagram

```
users ←─→ user_workspace_members ←─→ workspaces
  │                                       │
  │                                       ├─→ channels ←─→ channel_members ←─→ users
  │                                       │                    │
  │                                       │                    └─→ messages
  │                                       │                         ├─ thread_id
  │                                       │                         ├─ message_edit_history
  │                                       │                         ├─ reactions
  │                                       │                         └─ files
  │
  ├─→ direct_messages (1-to-1 or group)
  │    ├─ files
  │    └─ reactions
  │
  ├─→ notifications
  │
  └─→ audit_logs
```

5.2 Complete Database Schema (PostgreSQL 15)

```
-- Enable UUID extension
CREATE EXTENSION IF NOT EXISTS "uuid-ossp";
CREATE EXTENSION IF NOT EXISTS "pg_trgm";  -- For fuzzy search

-- ============================================================================
-- USERS TABLE (Authentication &amp; Identity)
-- ============================================================================
CREATE TABLE users (
    id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
    email VARCHAR(255) UNIQUE NOT NULL,
    password_hash VARCHAR(255),  -- bcrypt hash: $2b$12$...
    display_name VARCHAR(100) NOT NULL,
    avatar_url VARCHAR(500),     -- Nullable, uploaded to storage
    bio TEXT,
    timezone VARCHAR(50) DEFAULT 'UTC',
    status VARCHAR(20) DEFAULT 'offline',  -- online, away, offline, dnd
    status_message VARCHAR(100),
    email_verified BOOLEAN DEFAULT false,
    created_at TIMESTAMP WITH TIME ZONE DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP WITH TIME ZONE DEFAULT CURRENT_TIMESTAMP,
    last_login TIMESTAMP WITH TIME ZONE,
    last_seen_at TIMESTAMP WITH TIME ZONE,
    deleted_at TIMESTAMP WITH TIME ZONE,

    -- Constraints
    CONSTRAINT email_format CHECK (email ~ '^[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Za-z]{2,}$'),
    CONSTRAINT status_valid CHECK (status IN ('online', 'away', 'offline', 'dnd')),
    CONSTRAINT display_name_length CHECK (char_length(display_name) BETWEEN 2 AND 100)
);

CREATE INDEX idx_users_email_lower ON users(LOWER(email));
CREATE INDEX idx_users_display_name_trgm ON users USING GIN(display_name GIN_TRGM_OPS);

-- ============================================================================
-- WORKSPACES TABLE (Multi-Tenancy)
-- ============================================================================
CREATE TABLE workspaces (
    id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
    name VARCHAR(100) NOT NULL,
    slug VARCHAR(100) UNIQUE NOT NULL,
    description TEXT,
    owner_id UUID NOT NULL REFERENCES users(id) ON DELETE RESTRICT,
    plan VARCHAR(20) DEFAULT 'free',  -- free, pro, enterprise
    member_limit INT DEFAULT 30,
    member_count INT DEFAULT 1,
    features JSONB DEFAULT '{"messaging": true, "search": true}',
    created_at TIMESTAMP WITH TIME ZONE DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP WITH TIME ZONE DEFAULT CURRENT_TIMESTAMP,
    deleted_at TIMESTAMP WITH TIME ZONE,

    CONSTRAINT plan_valid CHECK (plan IN ('free', 'pro', 'enterprise')),
    CONSTRAINT member_limit_positive CHECK (member_limit &gt; 0),
    CONSTRAINT slug_format CHECK (slug ~ '^[a-z0-9-]+$')
);

CREATE INDEX idx_workspaces_owner ON workspaces(owner_id);
CREATE INDEX idx_workspaces_plan ON workspaces(plan);

-- ============================================================================
-- USER_WORKSPACE_MEMBERS TABLE (Many-to-Many with Roles)
-- ============================================================================
CREATE TABLE user_workspace_members (
    id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
    workspace_id UUID NOT NULL REFERENCES workspaces(id) ON DELETE CASCADE,
    user_id UUID NOT NULL REFERENCES users(id) ON DELETE CASCADE,
    role VARCHAR(20) DEFAULT 'member',  -- owner, admin, moderator, member
    joined_at TIMESTAMP WITH TIME ZONE DEFAULT CURRENT_TIMESTAMP,
    status VARCHAR(20) DEFAULT 'active',  -- active, invited, left, removed
    last_active TIMESTAMP WITH TIME ZONE,

    UNIQUE(workspace_id, user_id),
    CONSTRAINT role_valid CHECK (role IN ('owner', 'admin', 'moderator', 'member')),
    CONSTRAINT status_valid CHECK (status IN ('active', 'invited', 'left', 'removed'))
);

CREATE INDEX idx_user_workspace_members_user ON user_workspace_members(user_id);
CREATE INDEX idx_user_workspace_members_workspace ON user_workspace_members(workspace_id);
CREATE INDEX idx_user_workspace_members_status ON user_workspace_members(status);

-- ============================================================================
-- CHANNELS TABLE (Conversation Organization)
-- ============================================================================
CREATE TABLE channels (
    id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
    workspace_id UUID NOT NULL REFERENCES workspaces(id) ON DELETE CASCADE,
    name VARCHAR(80) NOT NULL,
    slug VARCHAR(80) NOT NULL,
    type VARCHAR(20) DEFAULT 'public',  -- public, private, direct, group_dm
    description TEXT,
    topic VARCHAR(500),
    created_by UUID NOT NULL REFERENCES users(id) ON DELETE SET NULL,
    created_at TIMESTAMP WITH TIME ZONE DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP WITH TIME ZONE DEFAULT CURRENT_TIMESTAMP,
    archived BOOLEAN DEFAULT false,
    archived_at TIMESTAMP WITH TIME ZONE,
    deleted_at TIMESTAMP WITH TIME ZONE,
    message_count INT DEFAULT 0,
    last_message_at TIMESTAMP WITH TIME ZONE,

    UNIQUE(workspace_id, slug),
    CONSTRAINT type_valid CHECK (type IN ('public', 'private', 'direct', 'group_dm')),
    CONSTRAINT message_count_positive CHECK (message_count &gt;= 0),
    CONSTRAINT name_length CHECK (char_length(name) BETWEEN 3 AND 80),
    CONSTRAINT slug_format CHECK (slug ~ '^[a-z0-9-]+$')
);

CREATE INDEX idx_channels_workspace ON channels(workspace_id);
CREATE INDEX idx_channels_type ON channels(type);
CREATE INDEX idx_channels_created_by ON channels(created_by);
CREATE INDEX idx_channels_last_message ON channels(last_message_at DESC);

-- ============================================================================
-- CHANNEL_MEMBERS TABLE (Channel Access Control)
-- ============================================================================
CREATE TABLE channel_members (
    id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
    channel_id UUID NOT NULL REFERENCES channels(id) ON DELETE CASCADE,
    user_id UUID NOT NULL REFERENCES users(id) ON DELETE CASCADE,
    role VARCHAR(20) DEFAULT 'member',  -- moderator, member
```

```sql
    joined_at TIMESTAMP WITH TIME ZONE DEFAULT CURRENT_TIMESTAMP,
    last_read_message_id UUID,
    last_read_at TIMESTAMP WITH TIME ZONE,

    UNIQUE(channel_id, user_id),
    CONSTRAINT role_valid CHECK (role IN ('moderator', 'member'))
);

CREATE INDEX idx_channel_members_user ON channel_members(user_id);
CREATE INDEX idx_channel_members_channel ON channel_members(channel_id);

-- ============================================================================
-- MESSAGES TABLE (Core Entity - Optimized for Real-Time)
-- ============================================================================
CREATE TABLE messages (
    id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
    channel_id UUID NOT NULL REFERENCES channels(id) ON DELETE CASCADE,
    user_id UUID REFERENCES users(id) ON DELETE SET NULL,
    content TEXT NOT NULL,
    thread_id UUID REFERENCES messages(id) ON DELETE CASCADE,  -- For threaded replies
    edited_at TIMESTAMP WITH TIME ZONE,
    deleted_at TIMESTAMP WITH TIME ZONE,
    created_at TIMESTAMP WITH TIME ZONE DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP WITH TIME ZONE DEFAULT CURRENT_TIMESTAMP,
    metadata JSONB DEFAULT '{}',  -- Flexible metadata (links, mentions, etc.)

    CONSTRAINT content_not_empty CHECK (length(trim(content)) &gt; 0),
    CONSTRAINT content_max_length CHECK (octet_length(content) &lt;= 4000),
    CONSTRAINT valid_content_encoding CHECK (content ~ '[A-Za-z0-9\s\-._~:/?#\[\]@!$&amp;'"'"'()*+,;=%\n\r
);

-- Critical indexes for message retrieval
CREATE INDEX idx_messages_channel_created ON messages(channel_id, created_at DESC)
    WHERE deleted_at IS NULL;

CREATE INDEX idx_messages_channel_id_created_id ON messages(channel_id, created_at DESC, id)
    WHERE deleted_at IS NULL;

CREATE INDEX idx_messages_thread ON messages(thread_id, created_at DESC)
    WHERE thread_id IS NOT NULL AND deleted_at IS NULL;

CREATE INDEX idx_messages_user ON messages(user_id, created_at DESC)
    WHERE deleted_at IS NULL;

-- Full-text search index
CREATE INDEX idx_messages_content_fts ON messages USING GIN(to_tsvector('english', content))
    WHERE deleted_at IS NULL;

-- Trigram index for fuzzy search
CREATE INDEX idx_messages_content_trgm ON messages USING GIN(content GIN_TRGM_OPS);

-- ============================================================================
-- MESSAGE_EDIT_HISTORY TABLE (Audit Trail)
-- ============================================================================
CREATE TABLE message_edit_history (
    id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
    message_id UUID NOT NULL REFERENCES messages(id) ON DELETE CASCADE,
    previous_content TEXT NOT NULL,
    edited_by UUID NOT NULL REFERENCES users(id) ON DELETE SET NULL,
    edited_at TIMESTAMP WITH TIME ZONE DEFAULT CURRENT_TIMESTAMP
);

CREATE INDEX idx_message_edit_history_message ON message_edit_history(message_id);

-- ============================================================================
-- REACTIONS TABLE (Emoji Reactions)
-- ============================================================================
CREATE TABLE reactions (
    id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
    message_id UUID NOT NULL REFERENCES messages(id) ON DELETE CASCADE,
    user_id UUID NOT NULL REFERENCES users(id) ON DELETE CASCADE,
    emoji VARCHAR(10) NOT NULL,  -- Unicode emoji
    created_at TIMESTAMP WITH TIME ZONE DEFAULT CURRENT_TIMESTAMP,

    UNIQUE(message_id, user_id, emoji),
    CONSTRAINT emoji_valid CHECK (emoji ~ '^[\p{Emoji}]+$')
);

CREATE INDEX idx_reactions_message ON reactions(message_id);
CREATE INDEX idx_reactions_user ON reactions(user_id);

-- ============================================================================
-- DIRECT_MESSAGES TABLE (1-on-1 &amp; Group Conversations)
-- ============================================================================
CREATE TABLE direct_messages (
    id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
    sender_id UUID NOT NULL REFERENCES users(id) ON DELETE SET NULL,
    recipient_id UUID NOT NULL REFERENCES users(id) ON DELETE SET NULL,
    content TEXT NOT NULL,
    edited_at TIMESTAMP WITH TIME ZONE,
    deleted_at TIMESTAMP WITH TIME ZONE,
    created_at TIMESTAMP WITH TIME ZONE DEFAULT CURRENT_TIMESTAMP,

    CONSTRAINT content_not_empty CHECK (length(trim(content)) &gt; 0),
    CONSTRAINT different_users CHECK (sender_id != recipient_id)
);

CREATE INDEX idx_direct_messages_pair ON direct_messages(sender_id, recipient_id, created_at DESC)
    WHERE deleted_at IS NULL;

CREATE INDEX idx_direct_messages_recipient ON direct_messages(recipient_id, created_at DESC)
    WHERE deleted_at IS NULL;

-- ============================================================================
-- FILES TABLE (Upload &amp; Storage Metadata)
-- ============================================================================
CREATE TABLE files (
    id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
    message_id UUID REFERENCES messages(id) ON DELETE SET NULL,
    dm_id UUID REFERENCES direct_messages(id) ON DELETE SET NULL,
    filename VARCHAR(255) NOT NULL,
    file_size INT NOT NULL,  -- bytes
    file_type VARCHAR(50),   -- MIME type: image/jpeg, application/pdf
    storage_path VARCHAR(500) NOT NULL,
    uploaded_by UUID NOT NULL REFERENCES users(id) ON DELETE SET NULL,
    uploaded_at TIMESTAMP WITH TIME ZONE DEFAULT CURRENT_TIMESTAMP,
    deleted_at TIMESTAMP WITH TIME ZONE,
    metadata JSONB DEFAULT '{}',  -- {width, height, duration, etc}

    CONSTRAINT exactly_one_parent CHECK (
        (message_id IS NOT NULL AND dm_id IS NULL) OR
        (message_id IS NULL AND dm_id IS NOT NULL)
    ),
    CONSTRAINT positive_file_size CHECK (file_size &gt; 0 AND file_size &lt;= 52428800)  -- 50MB
);

CREATE INDEX idx_files_message ON files(message_id);
CREATE INDEX idx_files_dm ON files(dm_id);
CREATE INDEX idx_files_uploaded_by ON files(uploaded_by);

-- ============================================================================
-- NOTIFICATIONS TABLE (User Notifications)
-- ============================================================================
CREATE TABLE notifications (
    id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
    user_id UUID NOT NULL REFERENCES users(id) ON DELETE CASCADE,
    type VARCHAR(50) NOT NULL,  -- mention, reply, reaction, channel_activity, dm
    channel_id UUID REFERENCES channels(id) ON DELETE CASCADE,
    message_id UUID REFERENCES messages(id) ON DELETE CASCADE,
    actor_id UUID REFERENCES users(id) ON DELETE SET NULL,
    metadata JSONB DEFAULT '{}',
    created_at TIMESTAMP WITH TIME ZONE DEFAULT CURRENT_TIMESTAMP,
    read BOOLEAN DEFAULT false,
    read_at TIMESTAMP WITH TIME ZONE,
```

```sql
    CONSTRAINT type_valid CHECK (type IN ('mention', 'reply', 'reaction', 'channel_activity', 'dm'))
);

CREATE INDEX idx_notifications_user ON notifications(user_id, created_at DESC);
CREATE INDEX idx_notifications_read ON notifications(user_id, read) WHERE read = false;

-- ============================================================================
-- AUDIT_LOGS TABLE (Compliance &amp; Security)
-- ============================================================================
CREATE TABLE audit_logs (
    id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
    workspace_id UUID NOT NULL REFERENCES workspaces(id) ON DELETE CASCADE,
    actor_id UUID REFERENCES users(id) ON DELETE SET NULL,
    action VARCHAR(100) NOT NULL,  -- user_signup, channel_created, message_edited, member_removed
    resource_type VARCHAR(50),     -- user, channel, message, workspace
    resource_id UUID,
    details JSONB DEFAULT '{}',    -- {ip, user_agent, changes, reason, ...}
    created_at TIMESTAMP WITH TIME ZONE DEFAULT CURRENT_TIMESTAMP
);

CREATE INDEX idx_audit_logs_workspace ON audit_logs(workspace_id, created_at DESC);
CREATE INDEX idx_audit_logs_actor ON audit_logs(actor_id, created_at DESC);
CREATE INDEX idx_audit_logs_action ON audit_logs(action, created_at DESC);

-- ============================================================================
-- MATERIALIZED VIEWS (Optimizations)
-- ============================================================================

-- Channel statistics
CREATE MATERIALIZED VIEW v_channel_stats AS
SELECT
    c.id,
    c.workspace_id,
    COUNT(m.id) as total_messages,
    COUNT(DISTINCT m.user_id) as unique_senders,
    MAX(m.created_at) as last_message_at,
    COUNT(DISTINCT cm.user_id) as member_count
FROM channels c
LEFT JOIN messages m ON c.id = m.channel_id AND m.deleted_at IS NULL
LEFT JOIN channel_members cm ON c.id = cm.channel_id
WHERE c.deleted_at IS NULL
GROUP BY c.id, c.workspace_id;

CREATE UNIQUE INDEX idx_v_channel_stats_id ON v_channel_stats(id);

-- User activity summary
CREATE MATERIALIZED VIEW v_user_activity AS
SELECT
    u.id,
    u.workspace_id,
    COUNT(m.id) as messages_sent,
    COUNT(r.id) as reactions_added,
    MAX(m.created_at) as last_message_at,
    MAX(u.last_login) as last_login
FROM user_workspace_members u
LEFT JOIN messages m ON u.user_id = m.user_id AND m.deleted_at IS NULL
LEFT JOIN reactions r ON u.user_id = r.user_id
WHERE u.status = 'active'
GROUP BY u.id, u.workspace_id;

-- Refresh: REFRESH MATERIALIZED VIEW CONCURRENTLY v_user_activity;
```

5.3 Data Access Patterns & Query Performance

```
// src/db/queries.ts - Optimized SQL queries

// Pattern 1: Message retrieval (most common)
// Query: Load 50 messages for a channel, sorted by newest first
const getMessages = async (channelId, page = 1, limit = 50) =&gt; {
  const offset = (page - 1) * limit;
  return db.query(`
    SELECT
      m.id, m.channel_id, m.user_id, m.content, m.thread_id,
      m.created_at, m.edited_at, m.deleted_at,
      u.display_name, u.avatar_url,
      COALESCE(json_agg(r.*) FILTER (WHERE r.id IS NOT NULL), '[]') as reactions,
      COUNT(*) OVER () as total_count
    FROM messages m
    LEFT JOIN users u ON m.user_id = u.id
    LEFT JOIN reactions r ON m.id = r.message_id
    WHERE m.channel_id = $1 AND m.deleted_at IS NULL AND m.thread_id IS NULL
    GROUP BY m.id, u.id
    ORDER BY m.created_at DESC
    LIMIT $2 OFFSET $3
  `, [channelId, limit, offset]);
};

// Index: idx_messages_channel_created
// Execution: &lt;50ms for 1M messages with proper index

// Pattern 2: Full-text search
// Query: Search messages by keyword with filters
const searchMessages = async (workspaceId, query, filters) =&gt; {
  return db.query(`
    SELECT
      m.id, m.content, m.channel_id,
      u.display_name,
      c.name as channel_name,
      ts_rank(to_tsvector('english', m.content), query) as rank
    FROM messages m
    JOIN users u ON m.user_id = u.id
    JOIN channels c ON m.channel_id = c.id
    WHERE to_tsvector('english', m.content) @@ $1
      AND c.workspace_id = $2
      AND m.deleted_at IS NULL
      ${filters.userId ? 'AND m.user_id = $3' : ''}
      ${filters.channelId ? 'AND m.channel_id = $4' : ''}
      ${filters.before ? 'AND m.created_at &lt; $5' : ''}
      ${filters.after ? 'AND m.created_at &gt; $6' : ''}
    ORDER BY rank DESC
    LIMIT 20 OFFSET $7
  `, queryParams);
};

// Index: idx_messages_content_fts
// Execution: &lt;2s for 1M messages

// Pattern 3: User presence (cache-heavy)
// Query: Get online users in workspace
// Most data from Redis, fallback to PostgreSQL for verification
const getOnlineUsers = async (workspaceId) =&gt; {
  // 1. Check Redis: presence:${workspaceId}:online
  // 2. If miss, query: SELECT users with last_seen_at &gt; now() - 5min
  // 3. Cache in Redis for 5 minutes
};

// Pattern 4: Notification fetch
const getNotifications = async (userId, unreadOnly = false) =&gt; {
  return db.query(`
    SELECT *
    FROM notifications
    WHERE user_id = $1
      ${unreadOnly ? 'AND read = false' : ''}
    ORDER BY created_at DESC
    LIMIT 20
  `, [userId]);
};

// Index: idx_notifications_read (for unread queries)
// Execution: &lt;10ms with index
```

6. API Architecture & Design Patterns

6.1 REST API Design

Principles:

- ✅ Resource-oriented (nouns, not verbs)
- ✅ Stateless (no server-side sessions required)
- ✅ Version-less (API stability via backwards compatibility)
- ✅ Content negotiation (Accept header for JSON/XML)
- ✅ Proper HTTP status codes (2xx, 4xx, 5xx)

Endpoint Structure:

```
API Base: https://slackteam.lab.home.lucasacchi.net:8282/api

Authentication:
  POST   /auth/register              # Signup
  POST   /auth/login                 # Login
  POST   /auth/logout                # Logout
  POST   /auth/refresh               # Refresh token
  GET    /auth/me                    # Current user

Users:
  GET    /users/:id                  # Get user
  PUT    /users/me                   # Update profile
  GET    /users/me/workspaces        # List user workspaces

Workspaces:
  POST   /workspaces                 # Create
  GET    /workspaces                 # List
  GET    /workspaces/:id             # Get
  PUT    /workspaces/:id             # Update
  DELETE /workspaces/:id             # Delete
  POST   /workspaces/:id/invite      # Invite members
  GET    /workspaces/:id/members     # List members
  PUT    /workspaces/:id/members/:uid # Update role

Channels:
  POST   /channels                   # Create
  GET    /channels                   # List
  GET    /channels/:id               # Get
  PUT    /channels/:id               # Update
  DELETE /channels/:id               # Delete (soft)
  POST   /channels/:id/members       # Add member
  DELETE /channels/:id/members/:uid  # Remove member

Messages:
  POST   /channels/:id/messages      # Send
  GET    /channels/:id/messages      # List (paginated)
  PUT    /messages/:id               # Edit (1-hour window)
  DELETE /messages/:id               # Delete (soft)

Reactions:
  POST   /messages/:id/reactions     # Add reaction
  DELETE /messages/:id/reactions/:emoji # Remove reaction

Search:
  GET    /search?q=keyword&amp;from=@user&amp;in=#channel

Direct Messages:
  POST   /dms                       # Create or get DM
  GET    /dms                       # List conversations
  GET    /dms/:id/messages          # Get messages
  POST   /dms/:id/messages          # Send DM

Utilities:
  GET    /health                    # Health check
```

6.2 Request/Response Format

Request Header:

```
POST /api/channels/abc-123/messages HTTP/1.1
Host: slackteam.lab.home.lucasacchi.net:8282
Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...
Content-Type: application/json
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)...
Content-Length: 245

{
  "content": "Hello @team! Check out this feature.",
  "thread_id": null
}
```

Success Response:

```
HTTP/1.1 201 Created
Content-Type: application/json
Content-Length: 342
X-Request-ID: req_12345abcde

{
  "success": true,
  "data": {
    "id": "msg-550e8400-e29b-41d4-a716-446655440000",
    "channel_id": "ch-abc-123",
    "user_id": "user-001",
    "content": "Hello @team! Check out this feature.",
    "created_at": "2025-11-19T14:30:45.123Z",
    "status": "sent"
  },
  "meta": {
    "request_id": "req_12345abcde",
    "timestamp": "2025-11-19T14:30:45.123Z"
  }
}
```

Error Response:

```
HTTP/1.1 400 Bad Request
Content-Type: application/json
X-Request-ID: req_12345abcde

{
  "success": false,
  "error": {
    "code": "INVALID_REQUEST",
    "message": "Message cannot be empty",
    "details": {
      "field": "content",
      "constraint": "required"
    }
  },
  "meta": {
    "request_id": "req_12345abcde",
    "timestamp": "2025-11-19T14:30:45.123Z"
  }
}
```

6.3 Error Codes & Handling

```
// src/utils/errors.ts
enum ErrorCode {
  // Authentication (4xx)
  INVALID_CREDENTIALS = 'INVALID_CREDENTIALS',  // 401
  TOKEN_EXPIRED = 'TOKEN_EXPIRED',              // 401
  TOKEN_INVALID = 'TOKEN_INVALID',              // 403
  UNAUTHORIZED = 'UNAUTHORIZED',                // 403
  FORBIDDEN = 'FORBIDDEN',                      // 403

  // Client Errors (4xx)
  INVALID_REQUEST = 'INVALID_REQUEST',          // 400
  NOT_FOUND = 'NOT_FOUND',                      // 404
  CONFLICT = 'CONFLICT',                        // 409
  UNPROCESSABLE = 'UNPROCESSABLE',              // 422
  RATE_LIMITED = 'RATE_LIMITED',                // 429

  // Server Errors (5xx)
  INTERNAL_ERROR = 'INTERNAL_ERROR',            // 500
  DATABASE_ERROR = 'DATABASE_ERROR',            // 500
  SERVICE_UNAVAILABLE = 'SERVICE_UNAVAILABLE',  // 503
}

// Centralized error handler
app.use((err, req, res, next) =&gt; {
  const requestId = req.header('X-Request-ID') || generateId();

  logger.error({
    error: err.message,
    code: err.code,
    requestId,
    path: req.path,
    method: req.method,
    userId: req.user?.id,
    stack: err.stack
  });

  const status = HTTP_STATUS_MAP[err.code] || 500;

  res.status(status).json({
    success: false,
    error: {
      code: err.code || 'INTERNAL_ERROR',
      message: err.message,
      ...(process.env.NODE_ENV === 'development' &amp;&amp; { stack: err.stack })
    },
    meta: { requestId, timestamp: new Date().toISOString() }
  });
});
```

7. Real-Time Communication Architecture (WebSocket)

7.1 Socket.IO Server Setup

```
// src/websocket/socket-server.ts
const io = require('socket.io')(server, {
  cors: { origin: ['http://slackteam.lab.home.lucasacchi.net:8282'], methods: ['GET', 'POST'] },
  transports: ['websocket', 'polling'],  // Fallback if firewall blocks WS
  path: '/socket.io/',
  serveClient: false,  // Don't serve Socket.IO client (use npm package)
  upgradeTimeout: 10000,

  // Memory adapter (single server)
  // For multi-server: use @socket.io/redis-adapter
});

// Middleware: Authentication before connection
io.use((socket, next) =&gt; {
  const token = socket.handshake.auth.token;

  try {
    const decoded = jwt.verify(token, process.env.JWT_SECRET);
    socket.userId = decoded.user_id;
    socket.workspaceIds = decoded.workspace_ids;
    next();
  } catch (err) {
    next(new Error('Authentication error'));
  }
});

// Connection handler
io.on('connection', (socket) =&gt; {
  const userId = socket.userId;

  logger.info(`User ${userId} connected`, { socketId: socket.id });

  // User joins their personal room (for targeted messages)
  socket.join(`user:${userId}`);

  // User joins all their workspaces
  socket.workspaceIds.forEach(wsId =&gt; {
    socket.join(`workspace:${wsId}`);
  });

  // ========================================================================
  // MESSAGE EVENTS
  // ========================================================================

  socket.on('message:send', async (payload) =&gt; {
    const { channelId, content, threadId } = payload;

    try {
      // Validate &amp; create message
      const message = await messageService.sendMessage(userId, channelId, content, threadId);

      // Broadcast to channel
      io.to(`channel:${channelId}`).emit('message:received', {
        ...message,
        sender: { id: userId, name: getUsername(userId) }
      });

    } catch (error) {
      socket.emit('error', { message: error.message });
    }
  });

  socket.on('message:edit', async (payload) =&gt; {
    const { messageId, newContent } = payload;

    try {
      const updated = await messageService.editMessage(messageId, userId, newContent);

      // Get channel from message
      const channelId = updated.channel_id;
      io.to(`channel:${channelId}`).emit('message:edited', updated);

    } catch (error) {
      socket.emit('error', { message: error.message });
    }
  });

  socket.on('message:delete', async (payload) =&gt; {
    const { messageId } = payload;

    try {
      const deleted = await messageService.deleteMessage(messageId, userId);
      const channelId = deleted.channel_id;

      io.to(`channel:${channelId}`).emit('message:deleted', { messageId });

    } catch (error) {
      socket.emit('error', { message: error.message });
    }
  });

  // ============================================================================
  // CHANNEL EVENTS
  // ============================================================================

  socket.on('channel:join', async (payload) =&gt; {
    const { channelId } = payload;

    // Verify permission (user is member)
    const isMember = await channelService.isMember(channelId, userId);
    if (!isMember) {
      return socket.emit('error', { message: 'Not a member of this channel' });
    }

    socket.join(`channel:${channelId}`);
    socket.emit('channel:joined', { channelId });
  });

  socket.on('channel:leave', async (payload) =&gt; {
    const { channelId } = payload;
    socket.leave(`channel:${channelId}`);
  });

  // ============================================================================
  // TYPING INDICATORS
  // ============================================================================

  socket.on('typing:start', (payload) =&gt; {
    const { channelId } = payload;

    // Broadcast to channel (exclude sender)
    socket.to(`channel:${channelId}`).emit('typing:notification', {
      userId,
      userName: getUsername(userId)
    });
  });

  socket.on('typing:stop', (payload) =&gt; {
    const { channelId } = payload;

    socket.to(`channel:${channelId}`).emit('typing:stopped', { userId });
  });

  // ============================================================================
  // PRESENCE
  // ============================================================================

  socket.on('presence:update', async (payload) =&gt; {
    const { status } = payload;  // online, away, offline

    // Update in Redis
    await redis.set(`user:${userId}:status`, status, { EX: 3600 });

    // Broadcast to all workspaces
    socket.workspaceIds.forEach(wsId =&gt; {
      io.to(`workspace:${wsId}`).emit('presence:changed', {
        userId,
        status,
        timestamp: Date.now()
      });
    });
  });

  // ========================================================================
  // REACTIONS
  // ========================================================================

  socket.on('reaction:add', async (payload) =&gt; {
    const { messageId, emoji } = payload;

    try {
      await reactionService.addReaction(messageId, userId, emoji);

      const message = await messageService.getMessage(messageId);
      io.to(`channel:${message.channel_id}`).emit('reaction:added', {
        messageId,
        emoji,
        userId,
        count: await reactionService.getReactionCount(messageId, emoji)
      });

    } catch (error) {
      socket.emit('error', { message: error.message });
    }
  });

  socket.on('reaction:remove', async (payload) =&gt; {
    const { messageId, emoji } = payload;

    try {
      await reactionService.removeReaction(messageId, userId, emoji);

      const message = await messageService.getMessage(messageId);
      io.to(`channel:${message.channel_id}`).emit('reaction:removed', {
        messageId,
        emoji,
        userId
      });

    } catch (error) {
      socket.emit('error', { message: error.message });
    }
  });

  // ========================================================================
  // DISCONNECT
  // ========================================================================

  socket.on('disconnect', () =&gt; {
    logger.info(`User ${userId} disconnected`, { socketId: socket.id });

    // Update presence to 'offline' after 30s grace period
    setTimeout(async () =&gt; {
      const isStillConnected = io.sockets.sockets.get(socket.id);
      if (!isStillConnected) {
        await redis.set(`user:${userId}:status`, 'offline');
        socket.workspaceIds.forEach(wsId =&gt; {
          io.to(`workspace:${wsId}`).emit('presence:changed', {
            userId,
            status: 'offline',
            timestamp: Date.now()
          });
        });
      }
    }, 30000);  // 30 second grace period for reconnection
  });

  socket.on('error', (error) =&gt; {
    logger.error('Socket error', { userId, error: error.message });
  });
});
```

7.2 Real-Time Event Flow

```
┌─────────────────────────────────────────────────────────────────────────────┐
│                      CLIENT (React + Socket.IO)                        │
│                                                                         │
│  1. User types message: "Hello team!"                                  │
│  2. Client emits: socket.emit('message:send', {                        │
│        channelId: 'ch-123',                                            │
│        content: 'Hello team!'                                          │
│      })                                                                │
│  3. UI shows message immediately (optimistic render)                   │
│  4. Waits for 'message:received' confirmation                          │
│  5. Updates message ID when confirmed                                  │
└──────────────────────────┬──────────────────────────────────────────────────┘
                       │ WebSocket (bidirectional)
                       ↓
┌─────────────────────────────────────────────────────────────────────────────┐
│                    SERVER (Express + Socket.IO)                        │
│                                                                         │
│  1. Receive 'message:send' event from client                           │
│  2. Authenticate token                                                 │
│  3. Validate permission (user in channel)                              │
│  4. Validate content (not empty, &lt;4KB)                                  │
│  5. Sanitize content (escape HTML)                                     │
│  6. Create transaction:                                                │
│     - INSERT into messages table                                       │
│     - INSERT into notifications (if @mentions)                         │
│     - UPDATE channel.message_count++                                   │
│     - INSERT into audit_logs                                           │
│  7. Async (non-blocking):                                              │
│     - Index message in search (Elasticsearch)                          │
│     - Send push notifications                                          │
│  8. Broadcast to channel via Socket.IO:                                │
│     io.to(`channel:${channelId}`).emit('message:received', ...) │
│                                                                         │
│     Room: channel:ch-123 includes all members                          │
│     Event carries: {id, user_id, content, created_at, ...}             │
└──────────────────────────┬──────────────────────────────────────────────────┘
                       │ WebSocket push
                       ↓ (to all in room)
┌─────────────────────────────────────────────────────────────────────────────┐
│              OTHER CLIENTS (React + Socket.IO)                         │
│                                                                         │
│  1. Receive 'message:received' event                                   │
│  2. Update Zustand store with new message                              │
│  3. React component re-renders message                                 │
│  4. Message visible to all users in &lt;500ms (typical)                    │
│     P50: 100ms, P95: 300ms, P99: 500ms                                 │
└─────────────────────────────────────────────────────────────────────────────┘

Performance Timeline:
  0ms:    User clicks Send
  50ms:   Message persisted to DB
  100ms:  Sent to other users (P50)
  300ms:  Sent to other users (P95)
  500ms:  Sent to all users (P99)
```

8. Security Architecture & Implementation

8.1 Authentication Flow

```
┌────────────────────┐
│   User Browser  │
└─────────┬──────────┘
         │
         ├─ 1. POST /api/auth/signup {email, password, name}
         │
         ↓
┌────────────────────────────────────────────────┐
│  Express.js Server                          │
│  ├─ Validate inputs                          │
│  ├─ Check duplicate email (case-i)         │
│  ├─ Hash password: bcrypt(pass, 12)        │  ← 250ms
│  ├─ INSERT user (email_verified=false)  │
│  ├─ Generate JWT (24h TTL)                 │
│  ├─ Send verification email                │
│  └─ Return: 201 Created                    │
└──────────┬─────────────────────────────────────┘
         │
         ├─ 2. Check email inbox
         ├─ Click verification link: /verify?token=JWT
         │
         ↓
┌────────────────────────────────────────────────┐
│  Express.js Server                          │
│  ├─ Decode JWT verification token          │
│  ├─ Update user (email_verified=true)     │
│  └─ Redirect to login                      │
└──────────┬─────────────────────────────────────┘
         │
         ├─ 3. POST /api/auth/login {email, password}
         │
         ↓
┌────────────────────────────────────────────────┐
│  Express.js Server                          │
│  ├─ Rate limit check (5 attempts)          │
│  ├─ Fetch user (indexed on email)          │
│  ├─ Verify password: bcrypt.compare()     │  ← 250ms
│  ├─ Generate access token (24h)            │
│  ├─ Generate refresh token (30d)           │
│  ├─ Store session in Redis                 │
│  ├─ Update last_login                      │
│  └─ Return: {accessToken, refreshToken}│
└──────────┬─────────────────────────────────────┘
         │
         ├─ Browser stores tokens:
         │  ├─ accessToken: localStorage (accessible by JS)
         │  └─ refreshToken: httpOnly cookie (NOT accessible by JS)
         │
         ↓
┌────────────────────────────────────────────────┐
│  Subsequent API Calls                       │
│  ├─ Authorization: Bearer accessToken     │
│  ├─ Middleware validates JWT               │
│  ├─ Extracts user_id, workspace_ids        │
│  └─ Passes to route handlers               │
└────────────────────────────────────────────────┘
```

8.2 JWT Token Implementation

```
// src/utils/jwt.ts
const JWT_SECRET = process.env.JWT_SECRET;
const JWT_REFRESH_SECRET = process.env.JWT_REFRESH_SECRET;

// Issue access token (24 hours)
function generateAccessToken(userId, email, workspaceIds) {
  return jwt.sign(
    {
      user_id: userId,
      email,
      workspace_ids: workspaceIds,
      type: 'access_token',
      iat: Math.floor(Date.now() / 1000),
      exp: Math.floor(Date.now() / 1000) + 24 * 3600
    },
    JWT_SECRET,
    {
      algorithm: 'HS256',
      issuer: 'chatflow',
      subject: userId
    }
  );
}

// Issue refresh token (30 days)
function generateRefreshToken(userId) {
  return jwt.sign(
    {
      user_id: userId,
      type: 'refresh_token',
      iat: Math.floor(Date.now() / 1000),
      exp: Math.floor(Date.now() / 1000) + 30 * 24 * 3600
    },
    JWT_REFRESH_SECRET,
    {
      algorithm: 'HS256',
      issuer: 'chatflow'
    }
  );
}

// Verify &amp; decode token
function verifyToken(token, isRefresh = false) {
  try {
    const secret = isRefresh ? JWT_REFRESH_SECRET : JWT_SECRET;
    return jwt.verify(token, secret, {
      algorithms: ['HS256'],
      issuer: 'chatflow'
    });
  } catch (error) {
    if (error.name === 'TokenExpiredError') {
      throw new Error('TOKEN_EXPIRED');
    } else if (error.name === 'JsonWebTokenError') {
      throw new Error('TOKEN_INVALID');
    }
    throw error;
  }
}

// Express middleware
function authenticateToken(req, res, next) {
  const authHeader = req.headers['authorization'];
  const token = authHeader &amp;&amp; authHeader.split(' ')[^1];  // Bearer &lt;token&gt;

  if (!token) {
    return res.status(401).json({
      success: false,
      error: { code: 'UNAUTHORIZED', message: 'Missing authorization token' }
    });
  }

  try {
    const decoded = verifyToken(token);
    req.user = decoded;
    next();
  } catch (error) {
    return res.status(403).json({
      success: false,
      error: { code: 'FORBIDDEN', message: error.message }
    });
  }
}

module.exports = {
  generateAccessToken,
  generateRefreshToken,
  verifyToken,
  authenticateToken
};
```

8.3 Password Security

```
// src/utils/password.ts
const bcrypt = require('bcryptjs');

// Hash password (cost factor 12 = ~250ms per hash)
async function hashPassword(plaintext) {
  return bcrypt.hash(plaintext, 12);
}

// Verify password (constant time comparison)
async function verifyPassword(plaintext, hash) {
  return bcrypt.compare(plaintext, hash);  // ~250ms
}

// Validate password strength
function validatePassword(password) {
  const errors = [];

  if (password.length &lt; 8) {
    errors.push('At least 8 characters');
  }
  if (!password.match(/[A-Z]/)) {
    errors.push('At least 1 uppercase letter');
  }
  if (!password.match(/[0-9]/)) {
    errors.push('At least 1 number');
  }
  if (!password.match(/[!@#$%^&amp;*()\-_=+{};:'",.&lt;&gt;?\/]/)) {
    errors.push('At least 1 special character');
  }

  return errors;
}

module.exports = {
  hashPassword,
  verifyPassword,
  validatePassword
};
```

8.4 Role-Based Access Control (RBAC)

```
// src/middleware/authorization.ts

// Permission matrix
const PERMISSIONS = {
  owner: {
    workspace: ['*'],  // All workspace operations
    channels: ['*'],   // All channel operations
    members: ['invite', 'remove', 'promote', 'demote'],
    messages: ['delete_any']
  },
  admin: {
    workspace: ['view', 'update_settings'],
    channels: ['create', 'delete'],
    members: ['invite', 'remove'],
    messages: ['delete_any']
  },
  moderator: {
    workspace: ['view'],
    channels: ['create'],
    members: [],
    messages: ['delete_any']
  },
  member: {
    workspace: ['view'],
    channels: ['view'],
    members: [],
    messages: ['delete_own']
  }
};

// Middleware: Check permission
function authorize(resource, action) {
  return async (req, res, next) =&gt; {
    const userId = req.user.user_id;
    const workspaceId = req.params.workspaceId || req.body.workspaceId;

    try {
      // Get user's role in workspace
      const membership = await db.query(
        `SELECT role FROM user_workspace_members
         WHERE user_id = $1 AND workspace_id = $2`,
        [userId, workspaceId]
      );

      if (!membership) {
        return res.status(403).json({
          error: { code: 'FORBIDDEN', message: 'Not a member of this workspace' }
        });
      }

      const role = membership[^0].role;
      const permissions = PERMISSIONS[role][resource] || [];

      if (!permissions.includes('*') &amp;&amp; !permissions.includes(action)) {
        return res.status(403).json({
          error: { code: 'FORBIDDEN', message: `Insufficient permissions for ${action}` }
        });
      }

      req.userRole = role;
      next();
    } catch (error) {
      res.status(500).json({ error: { code: 'INTERNAL_ERROR' } });
    }
  };
}

// Example usage in routes
app.delete('/channels/:id', authenticate Token, authorize('channels', 'delete'), (req, res) =&gt; {
  // Delete channel (user has permission)
});
```

8.5 Input Validation & Sanitization

```
// src/middleware/validation.ts
const joi = require('joi');
const sanitizeHtml = require('sanitize-html');

// Schemas
const schemas = {
  signup: joi.object({
    email: joi.string().email().required(),
    password: joi.string().min(8).required(),
    display_name: joi.string().min(2).max(100).required()
  }),

  sendMessage: joi.object({
    content: joi.string().min(1).max(4000).required(),
    thread_id: joi.string().uuid().optional()
  }),

  editMessage: joi.object({
    content: joi.string().min(1).max(4000).required()
  })
};

// Validation middleware
function validate(schemaKey) {
  return (req, res, next) =&gt; {
    const schema = schemas[schemaKey];
    if (!schema) {
      return next();  // No schema defined
    }

    const { error, value } = schema.validate(req.body, {
      abortEarly: false,
      stripUnknown: true
    });

    if (error) {
      return res.status(400).json({
        success: false,
        error: {
          code: 'VALIDATION_ERROR',
          details: error.details.map(d =&gt; ({
            field: d.path.join('.'),
            message: d.message
          }))
        }
      });
    }

    req.validatedData = value;
    next();
  };
}

// Content sanitization
function sanitizeMessage(content) {
  return sanitizeHtml(content, {
    allowedTags: ['b', 'i', 'em', 'strong', 'code', 'pre'],
    allowedAttributes: {},
    allowedIframeHostnames: []
  });
}

module.exports = { validate, sanitizeMessage };
```

9. Performance & Scalability Design

9.1 Performance Targets (MVP Phase)

```
Message Latency (WebSocket):
  P50:  &lt;100ms    # 50% of messages delivered
  P95:  &lt;300ms    # 95% of messages delivered
  P99:  &lt;500ms    # 99% of messages delivered
  Target: 100+ msg/sec at 50 concurrent users

API Response Time (HTTP):
  GET /messages:    P95 &lt;100ms   # 50 messages paginated
  POST /messages:   P95 &lt;150ms   # Create new message
  GET /search:      P95 &lt;2s      # Full-text search
  PUT /messages:    P95 &lt;200ms   # Edit message
  GET /auth/me:     P95 &lt;50ms    # User profile fetch
  Other endpoints:  P95 &lt;100ms

Database Queries:
  Simple SELECT:    &lt;10ms        # Indexed queries
  Paginated query:  &lt;50ms (p95)  # 50 messages
  INSERT message:   &lt;100ms (p95) # With indexes
  Complex JOIN:     &lt;200ms (p95)

Connection Pool (Single VM):
  Min connections: 5
  Max connections: 20
  Idle timeout: 30s
  Acquisition timeout: 5s

Cache Performance (Redis):
  Cache hit rate:   &gt;90%
  Response time:    &lt;5ms per hit
  Eviction policy:  LRU (Least Recently Used)

Database Size:
  1M messages:      ~1GB
  100K users:       ~50MB
  10K channels:     ~10MB
  Total ~10GB:      Should fit on VM (100GB disk available)
```

9.2 Caching Strategy

```
// src/cache/strategy.ts

// Cache keys and TTLs
const CACHE_KEYS = {
  USER_PROFILE: `user:{userId}`,           // TTL: 1 hour
  CHANNEL_INFO: `channel:{channelId}`,     // TTL: 1 hour
  CHANNEL_MEMBERS: `channel:{channelId}:members`, // TTL: 30 min
  USER_WORKSPACES: `user:{userId}:workspaces`,    // TTL: 1 hour
  USER_PRESENCE: `user:{userId}:presence`,        // TTL: Session
  SESSION: `session:{userId}`,              // TTL: 24 hours
  RATE_LIMIT: `ratelimit:{key}`,            // TTL: 15 min / 1 hour
  MESSAGE_CACHE: `msg:{messageId}`,         // TTL: 48 hours (MVP)
  SEARCH_INDEX: `search:index:updated`,     // TTL: 5 min
};

class CacheManager {
  // Get with fallback to database
  async getOrFetch(key, fetchFn, ttl = 3600) {
    // 1. Try cache
    const cached = await redis.get(key);
    if (cached) {
      return JSON.parse(cached);
    }

    // 2. Fetch from database
    const fresh = await fetchFn();

    // 3. Cache result
    if (fresh) {
      await redis.setex(key, ttl, JSON.stringify(fresh));
    }

    return fresh;
  }

  // Cache invalidation
  async invalidate(key) {
    await redis.del(key);
  }

  // Bulk invalidation (e.g., when channel updated)
  async invalidatePattern(pattern) {
    const keys = await redis.keys(pattern);
    if (keys.length &gt; 0) {
      await redis.del(...keys);
    }
  }

  // Warm cache (on startup)
  async warmCache() {
    // Pre-load frequently accessed data
    // 1. Load top 100 channels
    // 2. Load active user profiles
    // 3. Load workspace info
  }
}

// Usage: Cache user profile
async function getUserProfile(userId) {
  return cache.getOrFetch(
    `user:${userId}`,
    () =&gt; db.query(`SELECT * FROM users WHERE id = $1`, [userId]),
    3600  // 1 hour TTL
  );
}
```

9.3 Database Query Optimization

```
// src/db/optimization.ts

// Connection pooling configuration
const pool = new Pool({
  host: process.env.DB_HOST,
  port: 5432,
  user: process.env.DB_USER,
  password: process.env.DB_PASSWORD,
  database: process.env.DB_NAME,
  max: 20,              // Max pool size
  min: 5,               // Min pool size
  idleTimeoutMillis: 30000,
  connectionTimeoutMillis: 5000,
  allowExitOnIdle: true
});

// Query batching for bulk operations
async function batchInsertMessages(messages) {
  const client = await pool.connect();

  try {
    await client.query('BEGIN');

    for (const msg of messages) {
      await client.query(
        `INSERT INTO messages (id, channel_id, user_id, content, created_at)
         VALUES ($1, $2, $3, $4, $5)`,
        [msg.id, msg.channel_id, msg.user_id, msg.content, msg.created_at]
      );
    }

    await client.query('COMMIT');
  } catch (error) {
    await client.query('ROLLBACK');
    throw error;
  } finally {
    client.release();
  }
}

// Prepared statements (prevent SQL injection)
const stmts = {
  getUser: `SELECT * FROM users WHERE id = $1`,
  getMessages: `SELECT * FROM messages WHERE channel_id = $1
                ORDER BY created_at DESC LIMIT $2 OFFSET $3`,
  createMessage: `INSERT INTO messages (...) VALUES (...) RETURNING *`
};

// EXPLAIN ANALYZE to find slow queries
async function analyzeQuery(query, params) {
  const result = await db.query(`EXPLAIN ANALYZE ${query}`, params);
  console.log(result.rows);
}
```

10. Deployment Architecture (Lab VM)

10.1 Infrastructure Setup

VM Specs (Lab Target):

- Host: slackteam.lab.home.lucasacchi.net
- OS: Ubuntu 22.04 LTS
- CPU: TBD (verify with `nproc`)
- RAM: TBD (verify with `free -h`)
- Disk: 100GB (verify with `df -h`)
- Network: Local LAN

Software Stack:

```
Layer 1 - OS &amp; Runtime:
  Linux Kernel: 5.15+
  Node.js: 24.11.1 (official binary or nvm)
  Python: 3.11.2

Layer 2 - Web Server:
  Nginx: 1.25+ (reverse proxy, static files)

Layer 3 - Application:
  Express.js: 4.x (Node.js)
  Socket.IO: 4.x (real-time)

Layer 4 - Data:
  PostgreSQL: 15.x (primary database)
  Redis: 7.x (cache &amp; session store)

Layer 5 - Monitoring:
  PM2: 5.x (process manager)
  Prometheus: 2.x (metrics)
  Grafana: 10.x (visualization)
```

10.2 Nginx Configuration

```
# /etc/nginx/sites-available/chatflow<a></a>
upstream app_backend {
    server 127.0.0.1:4000 max_fails=3 fail_timeout=30s;
}

server {
    listen 8282 default_server;
    listen [::]:8282 default_server;

    server_name slackteam.lab.home.lucasacchi.net;

    # SSL (optional for lab: self-signed)
    # ssl_certificate /etc/ssl/certs/chatflow.crt;
    # ssl_certificate_key /etc/ssl/private/chatflow.key;
    # ssl_protocols TLSv1.3 TLSv1.2;
    # ssl_ciphers HIGH:!aNULL:!MD5;

    # Gzip compression
    gzip on;
    gzip_types text/plain text/css text/javascript application/json;
    gzip_min_length 1024;
    gzip_comp_level 6;
    # Client limits
    client_max_body_size 50M;
    client_body_timeout 60s;

    # ======================================================================
    # Frontend - Static SPA (React)
    # ======================================================================
    location / {
        root /var/www/chatflow;
        index index.html;

        # SPA routing: any request not matching file → index.html
        try_files $uri $uri/ /index.html;

        # Cache control
        expires 1h;
        add_header Cache-Control "public, max-age=3600";

        # Security headers
        add_header X-Content-Type-Options "nosniff";
        add_header X-Frame-Options "SAMEORIGIN";
        add_header X-XSS-Protection "1; mode=block";
    }

    # ======================================================================
    # Static Assets (JS, CSS, images) - Long cache
    # ======================================================================
    location ~* \.(js|css|png|jpg|jpeg|gif|ico|svg|woff|woff2|ttf|eot)$ {
        root /var/www/chatflow;

        expires 1y;
        add_header Cache-Control "public, immutable";
        access_log off;
    }

    # ======================================================================
    # API Routes - Proxy to Express
    # ======================================================================
    location /api/ {
        proxy_pass http://app_backend;
        proxy_http_version 1.1;

        # Headers
        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;
        proxy_set_header X-Request-ID $request_id;

        # Timeouts
        proxy_connect_timeout 60s;
        proxy_send_timeout 60s;
        proxy_read_timeout 60s;

        # Buffering
        proxy_buffering on;
        proxy_buffer_size 4k;
        proxy_buffers 8 4k;

        # Error handling
        proxy_redirect off;
    }

    # ======================================================================
    # WebSocket Routes - Socket.IO
    # ======================================================================
    location /socket.io {
        proxy_pass http://app_backend;
        proxy_http_version 1.1;

        # WebSocket upgrade
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection "Upgrade";

        # Headers
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;

        # Timeouts (longer for long-lived WS)
        proxy_read_timeout 86400s;
        proxy_send_timeout 86400s;

        # Disable buffering for WebSocket
        proxy_buffering off;
        proxy_cache_bypass $http_upgrade;
    }

    # ====================================================================
    # Health Check
    # ====================================================================
    location /health {
        proxy_pass http://app_backend;
        access_log off;
    }

    # ====================================================================
    # Monitoring
    # ====================================================================
    location /metrics {
        proxy_pass http://app_backend;
    }
}
```

10.3 Automated Deployment Script

```
#!/bin/bash
# deploy.sh - Automated ChatFlow deployment to lab<a></a>

set -e  # Exit on error

# Configuration<a></a>
HOST="slackteam.lab.home.lucasacchi.net"
USER="slackteam"
APP_DIR="/home/slackteam/chatflow"
BACKUP_DIR="/home/slackteam/backups"
TIMESTAMP=$(date +%Y%m%d_%H%M%S)
LOG_FILE="/var/log/chatflow-deploy-$TIMESTAMP.log"

# Colors<a></a>
RED='\033[0;31m'
GREEN='\033[0;32m'
YELLOW='\033[1;33m'
NC='\033[0m' # No Color

log() {
  echo -e "${GREEN}[$(date +'%Y-%m-%d %H:%M:%S')]${NC} $1" | tee -a "$LOG_FILE"
}

error() {
  echo -e "${RED}[ERROR]${NC} $1" | tee -a "$LOG_FILE"
  exit 1
}

warn() {
  echo -e "${YELLOW}[WARN]${NC} $1" | tee -a "$LOG_FILE"
}

# Pre-flight checks<a></a>
log "Running pre-flight checks..."
ssh -o ConnectTimeout=5 $USER@$HOST "echo 'SSH OK'" || error "Cannot SSH to $HOST"
ssh $USER@$HOST "command -v node" || error "Node.js not installed"
ssh $USER@$HOST "command -v npm" || error "npm not installed"
ssh $USER@$HOST "command -v psql" || error "PostgreSQL not installed"

# Create backup<a></a>
log "Creating backup..."
ssh $USER@$HOST "mkdir -p $BACKUP_DIR &amp;&amp; cp -r $APP_DIR $BACKUP_DIR/chatflow_$TIMESTAMP 2&gt;/dev/n
log "Backup created: $BACKUP_DIR/chatflow_$TIMESTAMP"

# Stop services<a></a>
log "Stopping services..."
ssh $USER@$HOST "pm2 stop chatflow || true"
sleep 3

# Pull latest code<a></a>
log "Pulling latest code from Git..."
ssh $USER@$HOST "cd $APP_DIR &amp;&amp; git pull origin main &amp;&amp; git log -1 --oneline"

# Install dependencies<a></a>
log "Installing dependencies..."
ssh $USER@$HOST "cd $APP_DIR &amp;&amp; npm install --production --legacy-peer-deps"

# Database migrations<a></a>
log "Running database migrations..."
ssh $USER@$HOST "cd $APP_DIR &amp;&amp; npm run migrate:up 2&gt;&amp;1 | tee -a $LOG_FILE"

# Build frontend<a></a>
log "Building frontend..."
ssh $USER@$HOST "cd $APP_DIR/frontend &amp;&amp; npm install --legacy-peer-deps &amp;&amp; npm run build"

# Copy frontend to Nginx<a></a>
log "Deploying frontend..."
ssh $USER@$HOST "sudo cp -r $APP_DIR/frontend/dist/* /var/www/chatflow/ 2&gt;/dev/null || true"

# Start services<a></a>
log "Starting services..."
ssh $USER@$HOST "pm2 start npm --name chatflow -- start || pm2 restart chatflow"
sleep 5

# Health checks<a></a>
log "Running health checks..."

# Frontend check<a></a>
curl -s -f "http://$HOST:8282/" &gt; /dev/null &amp;&amp; log "✓ Frontend responding" || warn "Frontend no

# API health check<a></a>
curl -s -f "http://$HOST:8282/api/health" &gt; /dev/null &amp;&amp; log "✓ API health check passed" || war

# WebSocket check (optional)<a></a>
# timeout 5 curl -i -N -H "Connection: Upgrade" -H "Upgrade: websocket" http://$HOST:8282/socket.io || warn

# Logs<a></a>
log "Recent logs:"
ssh $USER@$HOST "pm2 logs chatflow --lines 20 --nostream" | tail -n 20

log "${GREEN}=== DEPLOYMENT SUCCESSFUL ===${NC}"
log "ChatFlow deployed to $HOST"
log "Frontend: http://$HOST:8282"
log "API: http://$HOST:8282/api"
log "Logs: $LOG_FILE"

# Rollback command<a></a>
log "${YELLOW}Rollback command (if needed):${NC}"
log "ssh $USER@$HOST 'cp -r $BACKUP_DIR/chatflow_$TIMESTAMP $APP_DIR &amp;&amp; pm2 restart chatflow'"
```

11. DevOps & Infrastructure Design

11.1 Process Management (PM2)

```
// ecosystem.config.js - PM2 configuration
module.exports = {
  apps: [
    {
      name: 'chatflow',
      script: './src/index.js',
      instances: 'max',  // Use all CPU cores
      exec_mode: 'cluster',
      env: {
        NODE_ENV: 'development',
        PORT: 4000
      },
      env_production: {
        NODE_ENV: 'production',
        PORT: 4000
      },
      // Logging
      out_file: '/var/log/chatflow/out.log',
      error_file: '/var/log/chatflow/error.log',
      log_file: '/var/log/chatflow/combined.log',
      time: true,
      // Restart policies
      autorestart: true,
      watch: false,  // Don't watch in production
      max_memory_restart: '500M',
      // Graceful shutdown
      kill_timeout: 5000,
      wait_ready: true,
      listen_timeout: 3000,
      // Monitoring
      monitor_interval: 5000
    }
  ],
  // Cluster mode
  exec_mode: 'cluster',
  // Deployment
  deploy: {
    production: {
      user: 'slackteam',
      host: 'slackteam.lab.home.lucasacchi.net',
      ref: 'origin/main',
      repo: 'https://github.com/your-repo/chatflow.git',
      path: '/home/slackteam/chatflow',
      'post-deploy': 'npm install &amp;&amp; npm run build &amp;&amp; pm2 reload ecosystem.config.js --env
    }
  }
};

// Usage:
// pm2 start ecosystem.config.js
// pm2 start ecosystem.config.js --env production
// pm2 monit
// pm2 logs
// pm2 reload ecosystem.config.js
```

12. Monitoring, Observability & Health Checks

12.1 Health Check Endpoint

```
// src/routes/health.ts
app.get('/health', async (req, res) =&gt; {
  const status = {
    status: 'healthy',
    timestamp: new Date().toISOString(),
    uptime: process.uptime(),
    services: {
      api: { status: 'ok' },
      database: { status: 'checking' },
      cache: { status: 'checking' },
      memory: { status: 'ok' }
    }
  };

  // Check database
  try {
    await db.query('SELECT 1');
    status.services.database = { status: 'ok' };
  } catch (error) {
    status.services.database = { status: 'error', error: error.message };
    status.status = 'degraded';
  }

  // Check Redis
  try {
    await redis.ping();
    status.services.cache = { status: 'ok' };
  } catch (error) {
    status.services.cache = { status: 'error', error: error.message };
    status.status = 'degraded';
  }

  // Check memory
  const used = process.memoryUsage();
  const heapUsedPercent = (used.heapUsed / used.heapTotal) * 100;
  if (heapUsedPercent &gt; 90) {
    status.services.memory = { status: 'warning', heapUsedPercent };
    status.status = 'degraded';
  } else if (heapUsedPercent &gt; 95) {
    status.services.memory = { status: 'critical', heapUsedPercent };
    status.status = 'unhealthy';
  }

  const statusCode = status.status === 'healthy' ? 200 : status.status === 'degraded' ? 503 : 500;
  res.status(statusCode).json(status);
});
```

12.2 Structured Logging

```
// src/utils/logger.ts
const pino = require('pino');

const logger = pino({
  level: process.env.LOG_LEVEL || 'info',
  transport: {
    target: 'pino-pretty',
    options: {
      colorize: true,
      translateTime: 'SYS:standard',
      singleLine: false,
      ignore: 'pid,hostname'
    }
  }
});

// Usage
logger.info({ userId: 'user-123' }, 'User logged in');
logger.error({ error: err, userId: 'user-123' }, 'Database error');
logger.warn({ action: 'rate_limit' }, 'Rate limit exceeded');

// Structured logging in middleware
app.use((req, res, next) =&gt; {
  const start = Date.now();

  res.on('finish', () =&gt; {
    const duration = Date.now() - start;
    logger.info({
      method: req.method,
      path: req.path,
      status: res.statusCode,
      duration,
      userId: req.user?.id
    }, `${req.method} ${req.path} ${res.statusCode}`);
  });

  next();
});
```

13. Testing Strategy & Coverage

13.1 Unit Tests (Jest)

```
// __tests__/auth.test.js
const { signup, login } = require('../src/services/AuthService');

describe('AuthService', () =&gt; {
  describe('signup', () =&gt; {
    test('Valid signup creates account', async () =&gt; {
      const user = await signup({
        email: 'new@example.com',
        password: 'ValidPass123!',
        displayName: 'John Doe'
      });

      expect(user.id).toBeDefined();
      expect(user.email_verified).toBe(false);
    });

    test('Duplicate email rejected', async () =&gt; {
      try {
        await signup({
          email: 'existing@example.com',
          password: 'ValidPass123!',
          displayName: 'Jane Doe'
        });
        fail('Should have thrown');
      } catch (error) {
        expect(error.message).toContain('Email already registered');
      }
    });
  });
});

// Coverage targets:
// ├─ Statements: &gt;80%
// ├─ Branches: &gt;75%
// ├─ Functions: &gt;80%
// └─ Lines: &gt;80%
```

13.2 Integration Tests

```
// __tests__/integration/messaging.test.js
const request = require('supertest');
const app = require('../src/app');

describe('Message Integration', () =&gt; {
  let token;
  let channelId;

  beforeAll(async () =&gt; {
    // Setup: Create user, workspace, channel
    const signupRes = await request(app)
      .post('/api/auth/login')
      .send({ email: 'test@example.com', password: 'TestPass123!' });
    token = signupRes.body.data.access_token;
  });

  test('Send message in channel', async () =&gt; {
    const res = await request(app)
      .post(`/api/channels/${channelId}/messages`)
      .set('Authorization', `Bearer ${token}`)
      .send({ content: 'Hello team!' });

    expect(res.status).toBe(201);
    expect(res.body.data.id).toBeDefined();
  });
});
```

14. Error Handling & Resilience

14.1 Retry Logic

```
// src/utils/retry.ts
async function withRetry(fn, options = {}) {
  const {
    maxAttempts = 3,
    delay = 1000,
    backoff = 2,  // Exponential backoff multiplier
    onRetry = () =&gt; {}
  } = options;

  let lastError;

  for (let attempt = 1; attempt &lt;= maxAttempts; attempt++) {
    try {
      return await fn();
    } catch (error) {
      lastError = error;

      if (attempt === maxAttempts) {
        throw error;
      }

      const waitTime = delay * Math.pow(backoff, attempt - 1);
      onRetry({ attempt, waitTime, error });

      await new Promise(resolve =&gt; setTimeout(resolve, waitTime));
    }
  }

  throw lastError;
}

// Usage
async function queryWithRetry() {
  return withRetry(
    () =&gt; db.query('SELECT * FROM users'),
    {
      maxAttempts: 3,
      delay: 500,
      backoff: 2
    }
  );
}
```

14.2 Circuit Breaker Pattern

```
// src/utils/circuit-breaker.ts
class CircuitBreaker {
  constructor(options = {}) {
    this.failureThreshold = options.failureThreshold || 5;
    this.resetTimeout = options.resetTimeout || 60000;
    this.state = 'closed';  // closed, open, half-open
    this.failures = 0;
    this.lastFailure = null;
  }

  async execute(fn) {
    if (this.state === 'open') {
      if (Date.now() - this.lastFailure &gt; this.resetTimeout) {
        this.state = 'half-open';
      } else {
        throw new Error('Circuit breaker is open');
      }
    }

    try {
      const result = await fn();
      this.onSuccess();
      return result;
    } catch (error) {
      this.onFailure();
      throw error;
    }
  }

  onSuccess() {
    this.failures = 0;
    this.state = 'closed';
  }

  onFailure() {
    this.failures++;
    this.lastFailure = Date.now();

    if (this.failures &gt;= this.failureThreshold) {
      this.state = 'open';
    }
  }
}

// Usage
const emailCircuit = new CircuitBreaker({ failureThreshold: 3, resetTimeout: 60000 });

async function sendEmail(to, subject, body) {
  return emailCircuit.execute(() =&gt; emailService.send(to, subject, body));
}
```

15. Implementation Sequence & Dependencies

15.1 Phase 1: Backend Foundation (Week 1)

```
Day 1-2: Database &amp; Schema
├─ Setup PostgreSQL 15
├─ Create database chatflow_dev
├─ Run schema migrations
├─ Create indexes
└─ Seed test data

Day 3-4: Authentication Service
├─ User signup (email verification)
├─ User login (JWT)
├─ Logout &amp; token refresh
├─ Rate limiting
└─ Unit tests

Day 5: API Gateway &amp; Deployment
├─ Express.js setup
├─ Nginx reverse proxy
├─ CORS &amp; security headers
├─ Health check endpoint
└─ PM2 process management
```

15.2 Phase 2: Core Messaging (Week 2)

```
Day 1-2: Message CRUD &amp; WebSocket
├─ Message table &amp; indexes
├─ Socket.IO server setup
├─ message:send event handler
├─ Real-time broadcast
└─ Optimistic UI

Day 3-4: Channels &amp; Membership
├─ Channel management API
├─ Channel member permissions
├─ Join/leave channel
└─ RBAC enforcement

Day 5: Search &amp; Notifications
├─ Full-text search (PostgreSQL FTS)
├─ Notification system (@mentions)
├─ User presence broadcast
└─ Integration tests
```

15.3 Phase 3: Frontend UI (Week 3)

```
Day 1-2: Auth &amp; Workspace UI
├─ Login/signup forms
├─ Workspace selector
├─ User profile page
└─ Auth state management (Zustand)

Day 3-4: Messaging UI
├─ Channel list
├─ Message view
├─ Message input + send
├─ Real-time message display
└─ Typing indicators

Day 5: Search &amp; Features
├─ Search UI with filters
├─ Message reactions (emoji picker)
├─ Direct messages
└─ Notifications
```

15.4 Phase 4: Polish & Deploy (Week 4)

```
Day 1-2: Load Testing &amp; Optimization
├─ K6 load test (100 concurrent users)
├─ Database query optimization
├─ Cache warming
├─ WebSocket tuning
└─ Frontend bundle optimization

Day 3: Security Audit
├─ Penetration testing
├─ SQL injection checks
├─ XSS prevention verification
├─ CSRF token validation
└─ Rate limit testing

Day 4: Monitoring &amp; Alerting
├─ Prometheus metrics setup
├─ Grafana dashboards
├─ Alert rules
├─ Backup strategy
└─ Disaster recovery test

Day 5: Production Deployment
├─ Final testing
├─ Deploy to lab
├─ Smoke tests
├─ Performance validation
└─ Documentation complete
```

16. Risk Mitigation & Technical Debt

Risk	Severity	Probability	Mitigation
WebSocket latency >1s	High	Medium	Load testing (Week 4), connection pooling tuning, Redis optimization
Database scalability	High	Low	Proper indexing (critical path), query optimization, connection pool sizing
Message loss	Critical	Low	Transaction guarantees, acknowledgments, backup strategy
Security vulnerability	Critical	Medium	Code review, security audit, OWASP checklist, penetration testing
Token expiry UX	Medium	High	Auto-refresh token logic, graceful error handling
File upload abuse	Medium	Medium	File size limits, virus scanning, rate limiting per user
Cache staleness	Low	Low	TTL strategy, invalidation on updates, cache warming
Deployment failure	Medium	Low	Automated rollback, backup strategy, blue-green deployment

17. Design Decisions & Trade-Offs

17.1 Monolithic vs Microservices

Decision: Monolithic (Single VM)

Rationale:

- MVP: 50-100 concurrent users, <10GB data
- Simpler debugging & deployment

- Faster development (no inter-service latency)
- Lower operational complexity

Trade-off:

- ✖ Harder to scale individual services (future: microservices in v1.1)
- ✅ Better for MVP timeline (4 weeks)

Upgrade Path (v1.1+):

```
Monolithic (MVP)
  ↓ (50→500 users)
  ├─ Split Auth service (independent scaling)
  ├─ Split Message service (high load)
  ├─ Split Search service (Elasticsearch)
  └─ Add message queue (RabbitMQ/Kafka)
```

17.2 SQL vs NoSQL

Decision: PostgreSQL (SQL)

Rationale:

- ACID compliance (message integrity critical)
- Complex queries (search, filters, joins)
- Strong schema validation
- Proven at scale

Trade-off:

- ✖ Less flexible schema (upfront design required)
- ✅ Data consistency guaranteed

17.3 JWT vs Sessions

Decision: JWT (Stateless)

Rationale:

- Scales without server-side session store
- Better for distributed systems (future)
- Mobile-friendly

Trade-off:

- ✖ Cannot revoke token immediately (TTL: 24h)
- ✅ Simpler architecture for MVP

17.4 Single Server vs Cluster

Decision: Single Node (MVP)

Rationale:

- Simpler deployment & debugging
- Sufficient for 50-100 users
- PM2 cluster mode ready for future

Trade-off:

- ✖ No redundancy (planned for v1.1)
- ✅ Cost-effective, faster development

18. Appendices

Appendix A: Dependencies & Versions

Backend:

```
{
  "express": "^4.18.2",
  "socket.io": "^4.5.4",
  "pg": "^8.10.0",
  "redis": "^4.6.5",
  "jsonwebtoken": "^9.0.0",
  "bcryptjs": "^2.4.3",
  "joi": "^17.10.0",
  "pino": "^8.14.1"
}
```

Frontend:

```
{
  "react": "^19.0.0",
  "react-router-dom": "^6.x",
  "zustand": "^4.4.x",
  "socket.io-client": "^4.5.x",
  "tailwindcss": "^3.3.x",
  "typescript": "^5.x"
}
```

Appendix B: Performance Monitoring Checklist

```
Metrics to Track:
  ✓ Message latency (p50, p95, p99)
  ✓ API response time (by endpoint)
  ✓ Database query time (by query type)
  ✓ WebSocket connection count
  ✓ Active users (concurrent)
  ✓ Cache hit rate
  ✓ Error rate (by type)
  ✓ CPU/Memory usage
  ✓ Disk I/O
  ✓ Network throughput

Thresholds for Alerts:
  ⚠ Warning: Message latency p95 &gt; 300ms
  ▯ Critical: Message latency p99 &gt; 1s
  ⚠ Warning: Error rate &gt; 0.5%
  ▯ Critical: Error rate &gt; 5%
  ⚠ Warning: Memory usage &gt; 80%
  ▯ Critical: Memory usage &gt; 95%
```

Appendix C: Security Checklist

```
Authentication:
  ☑ Password hashing (bcrypt, cost 12)
  ☑ JWT token signing &amp; validation
  ☑ Rate limiting (5 failures → 15min lockout)
  ☑ Email verification

Authorization:
  ☑ RBAC (owner, admin, moderator, member)
  ☑ Channel ACL enforcement
  ☑ Workspace isolation

Transport Security:
  ☑ HTTPS/TLS (self-signed for lab)
  ☑ HSTS header
  ☑ Secure cookies (httpOnly, SameSite)

Data Protection:
  ☑ Password never in logs
  ☑ Sensitive data encrypted at rest (future)
  ☑ Input validation &amp; sanitization
  ☑ XSS prevention (escape HTML)
  ☑ SQL injection prevention (parameterized)
  ☑ CSRF token (state-changing ops)

Compliance:
  ☑ Audit logs (all actions)
  ☑ Data retention (48h MVP)
  ☑ User data export (GDPR)
```

Document Status: Final for Development
Version: 1.0 (Lab-Optimized)
Last Updated: November 19, 2025
Next Review: Upon completion of Milestone 1 (Week 1)

Approved By:

- [ ] Senior Architect
- [ ] Engineering Lead
- [ ] DevOps Engineer
- [ ] QA Lead

[1]

⁂

1. ChatFlow_FAD_v2_Lab-Optimized.pdf