Product Requirements Document (PRD)

DocAgent: Multi-Agent AI-Powered Code Documentation Generation Platform

SaaS MERN Stack Web Application

Document Version: 2.0 (MERN Stack Edition)
Creation Date: October 11, 2025
Product Owner: [To be assigned]
Engineering Lead: [To be assigned]
Target Release Date: Q2 2026
Status: Planning Phase
Technology Stack: MERN (MongoDB, Express.js, React.js, Node.js) + LLM Integration

1. Executive Summary

1.1 Product Overview

DocAgent is a cloud-based SaaS platform built on the MERN stack that automatically generates comprehensive, context-aware documentation from source code using a multi-agent AI system. The platform coordinates specialized Large Language Models through a Node.js-based orchestration framework, delivering high-quality documentation including docstrings, architectural diagrams, usage examples, and cross-referenced API documentation through an intuitive React-based web interface.[3][4][1]

1.2 Elevator Pitch

DocAgent is a subscription-based SaaS platform that eliminates the 60% of software maintenance costs attributed to inadequate documentation by automatically generating, verifying, and maintaining comprehensive documentation synchronized with code changes, accessible through a modern web dashboard that integrates seamlessly with existing development workflows.[1][3]

1.3 Problem Statement

Current software documentation practices face critical challenges:[3]

- Manual documentation creates unsustainable overhead for development teams
- Documentation becomes outdated within weeks of code changes
- Existing automated tools generate only basic API references without architectural context
- Teams lack centralized platforms for managing documentation across projects
- No real-time collaboration or visibility into documentation generation status

1.4 SaaS Value Proposition

Why SaaS Model:[2][1]

- Zero installation or infrastructure management for customers
- Automatic updates and feature releases without customer intervention
- Scalable pricing aligned with usage (pay-as-you-grow)
- Centralized documentation accessible from anywhere
- Built-in collaboration and team management features

2. Strategic Alignment

2.1 Business Objectives

Primary Objectives:[1][3]

- Achieve $1M ARR (Annual Recurring Revenue) by Month 18
- Reduce customer documentation maintenance costs by 70%
- Onboard 500+ paying organizations by Month 12
- Maintain <5% monthly churn rate

Secondary Objectives:

- Establish market leadership in AI-powered documentation tools
- Create viral growth through developer community advocacy
- Build scalable multi-tenant platform supporting 10,000+ organizations

2.2 Product Goals

Short-term (Months 1-6):

- Launch MVP with Python and JavaScript support
- Onboard 100 beta customers (20% conversion to paid)
- Achieve 80% automation rate for documentation tasks
- Process 50,000 components monthly

Long-term (Months 7-18):

- Support 8+ programming languages
- Process 500,000+ components monthly
- Expand to enterprise tier with dedicated resources
- International expansion (APAC, Europe)

2.3 Revenue Model

Pricing Tiers:[5][1]

Tier	Price/Month	Components/Month	Repositories	Team Size	Key Features
Free	$0	100	1	1 user	Basic docstrings, CLI only
Starter	$29	1,000	5	5 users	Web UI, diagrams, GitHub integration
Professional	$99	10,000	20	20 users	Advanced diagrams, priority support, API access
Team	$299	50,000	Unlimited	50 users	Custom templates, SSO, analytics dashboard
Enterprise	Custom	Unlimited	Unlimited	Unlimited	On-premises option, SLA, dedicated support

Revenue Projections:

- Month 6: $15,000 MRR (100 Starter, 30 Professional, 5 Team)
- Month 12: $60,000 MRR (300 Starter, 100 Professional, 30 Team, 3 Enterprise)

- Month 18: $120,000 MRR (500 Starter, 200 Professional, 60 Team, 10 Enterprise)

2.4 Success Metrics

Metric Category	KPI	Target	Measurement Method
Revenue	Monthly Recurring Revenue (MRR)	$60K by Month 12	Billing system
Growth	Month-over-month growth rate	15%	User analytics
Activation	Time to first documentation	<10 minutes	Product analytics
Engagement	Weekly active users	70% of subscribers	Usage tracking
Quality	Documentation accuracy	>90%	Expert validation
Efficiency	Automation rate	85%	System telemetry
Satisfaction	Net Promoter Score (NPS)	>50	User surveys
Retention	Churn rate	<5%/month	Billing system

3. Target Users & Market

3.1 Primary Market Segments

Segment 1: Startups & Scale-ups (5-50 developers)[1]

- Market size: 200,000+ companies globally
- Pain: Fast-moving codebases, limited documentation resources
- Budget: $100-500/month for developer tools
- Target tier: Professional to Team

Segment 2: Mid-Market Companies (50-500 developers)[2]

- Market size: 50,000+ companies globally
- Pain: Knowledge silos, onboarding challenges, compliance requirements
- Budget: $500-5,000/month for developer tools
- Target tier: Team to Enterprise

Segment 3: Open Source Projects[3]

- Market size: 100,000+ active projects
- Pain: Volunteer maintainer time constraints, contributor friction
- Budget: $0-100/month (community-driven)
- Target tier: Free to Starter

3.2 User Personas

Persona 1: Senior Software Engineer (Sarah)[3]

- Role: Individual contributor on distributed systems team
- Company size: 50-200 employees
- Tech stack: Python microservices, React frontend
- Goals: Maintain documentation without overtime, improve code quality scores
- Frustrations: Manual documentation outdated immediately, lack of time
- Success criteria: Save 10+ hours/week on documentation tasks

Persona 2: Engineering Manager (Michael)

- Role: Manages team of 15 developers
- Company size: 200-1,000 employees
- Tech stack: Multi-language (Java, JavaScript, Python)
- Goals: Standardize documentation, reduce onboarding time, improve team velocity
- Frustrations: Inconsistent docs across teams, knowledge loss when developers leave
- Success criteria: Cut onboarding time from 4 weeks to 2 weeks

Persona 3: CTO/VP Engineering (Jennifer)

- Role: Technical leadership
- Company size: 100-500 employees
- Tech stack: Enterprise architecture
- Goals: Scale engineering org, reduce technical debt, ensure compliance
- Frustrations: Documentation debt blocking new hires, audit requirements

- Success criteria: 90%+ documentation coverage, audit-ready codebase

Persona 4: Open Source Maintainer (Alex)

- Role: Project maintainer
- Company size: Individual or small team
- Tech stack: Various open source projects
- Goals: Grow contributor community, reduce support burden
- Frustrations: Community complaints about docs, limited maintenance time
- Success criteria: Increase contributor retention, reduce duplicate issues

4. Core Features & Requirements

4.1 MERN Stack Architecture

4.1.1 Frontend Layer (React.js)

Technology Stack:[6][1]

- React 18+ with TypeScript
- State Management: Redux Toolkit + React Query
- UI Framework: Material-UI (MUI) or Ant Design
- Styling: TailwindCSS
- Code Editor: Monaco Editor (VSCode engine)
- Diagram Rendering: Mermaid.js, React-Flow
- Charts: Recharts, D3.js
- Authentication: JWT with secure HTTP-only cookies

Functional Requirements:

- FR-FE-001: Responsive web dashboard optimized for desktop and tablet
- FR-FE-002: Repository management interface with OAuth connection to GitHub/GitLab
- FR-FE-003: Real-time documentation generation progress tracking

- FR-FE-004: Interactive code and documentation side-by-side viewer
- FR-FE-005: Diagram editor with live preview and customization
- FR-FE-006: Team management and user role administration
- FR-FE-007: Analytics dashboard showing documentation coverage and quality metrics
- FR-FE-008: Settings panel for configuring templates, styles, and preferences
- FR-FE-009: Notification center for generation status and errors
- FR-FE-010: Search functionality across all documentation

Technical Requirements:

- TR-FE-001: Initial page load <2 seconds on 3G connection
- TR-FE-002: Real-time updates using WebSockets or Server-Sent Events
- TR-FE-003: Lazy loading for documentation and diagrams
- TR-FE-004: Offline capability for viewing previously loaded documentation
- TR-FE-005: Browser support: Chrome, Firefox, Safari, Edge (latest 2 versions)
- TR-FE-006: Accessibility compliant with WCAG 2.1 Level AA

Acceptance Criteria:

- React components achieve >90% test coverage
- Lighthouse performance score >90
- Zero critical accessibility violations
- Works seamlessly on screens 1024px width and above

4.1.2 Backend Layer (Node.js + Express.js)

Technology Stack:[2][1]

- Node.js 20+ LTS
- Express.js 4.x
- TypeScript for type safety
- Authentication: Passport.js with JWT strategy
- Validation: Joi or Zod schemas

- Job Queue: Bull (Redis-backed)
- WebSocket: Socket.io
- Payment Processing: Stripe SDK
- Rate Limiting: express-rate-limit
- Security: helmet, cors, express-mongo-sanitize

Functional Requirements:

- FR-BE-001: RESTful API endpoints for all CRUD operations
- FR-BE-002: Multi-tenant request handling with tenant isolation
- FR-BE-003: Job queue management for asynchronous documentation generation
- FR-BE-004: WebSocket server for real-time progress updates
- FR-BE-005: Subscription management and billing integration
- FR-BE-006: OAuth integration for GitHub, GitLab, Bitbucket
- FR-BE-007: Webhook endpoints for repository events
- FR-BE-008: LLM API orchestration and request management
- FR-BE-009: File upload and processing for direct code submission
- FR-BE-010: Rate limiting and quota enforcement per subscription tier

Technical Requirements:

- TR-BE-001: API response time <500ms for 95% of requests (excluding LLM calls)
- TR-BE-002: Support 1,000 concurrent API requests
- TR-BE-003: Horizontal scaling across multiple Node.js instances
- TR-BE-004: Graceful handling of LLM API failures with retries
- TR-BE-005: Comprehensive logging with Winston or Pino
- TR-BE-006: Input validation on all endpoints
- TR-BE-007: Rate limiting: 100 requests/minute per user (Free), 1,000/min (Paid)

Acceptance Criteria:

- API test coverage >85%

- Zero critical security vulnerabilities (OWASP Top 10)
- Successfully processes 100 concurrent documentation jobs
- Handles LLM API failures without data loss

4.1.3 Database Layer (MongoDB)

Database Architecture:[5][2]

Multi-Tenancy Model: Shared database, shared schema with tenant isolation via `tenantId` field[2]

Collections Design:

```
// Users Collection
{
  _id: ObjectId,
  email: String (indexed, unique),
  passwordHash: String,
  tenantId: ObjectId (indexed),
  role: String, // 'owner', 'admin', 'member', 'viewer'
  createdAt: Date,
  lastLogin: Date,
  settings: Object
}

// Tenants Collection (Organizations)
{
  _id: ObjectId,
  name: String,
  subdomain: String (indexed, unique),
  subscription: {
    tier: String, // 'free', 'starter', 'professional', 'team', 'enterprise'
    status: String, // 'active', 'trial', 'cancelled', 'suspended'
    currentPeriodEnd: Date,
    stripeCustomerId: String,
    stripeSubscriptionId: String
  },
  quotas: {
    componentsPerMonth: Number,
    maxRepositories: Number,
    maxTeamMembers: Number
```

```
  },
  usage: {
    componentsThisMonth: Number,
    lastResetDate: Date
  },
  createdAt: Date
}

// Repositories Collection
{
  _id: ObjectId,
  tenantId: ObjectId (indexed),
  name: String,
  provider: String, // 'github', 'gitlab', 'bitbucket'
  repoUrl: String,
  branch: String,
  language: String,
  lastSyncedAt: Date,
  webhookId: String,
  settings: Object,
  createdAt: Date
}

// Documentation Collection
{
  _id: ObjectId,
  tenantId: ObjectId (indexed),
  repositoryId: ObjectId (indexed),
  componentPath: String (indexed),
  componentType: String, // 'function', 'class', 'module'
  language: String,
  generatedContent: {
    docstring: String,
    markdown: String,
    metadata: Object
  },
  diagrams: [{
    type: String,
    mermaidCode: String,
    imageUrl: String,
```

```
    generatedAt: Date
  }],
  quality: {
    score: Number,
    verifierFeedback: String,
    iterationsCount: Number
  },
  version: Number,
  generatedAt: Date,
  lastUpdated: Date
}

// Jobs Collection
{
  _id: ObjectId,
  tenantId: ObjectId (indexed),
  repositoryId: ObjectId,
  type: String, // 'generate', 'regenerate', 'verify'
  status: String, // 'queued', 'processing', 'completed', 'failed'
  progress: Number,
  componentPath: String,
  result: Object,
  error: Object,
  createdAt: Date,
  completedAt: Date
}

// Diagrams Collection
{
  _id: ObjectId,
  tenantId: ObjectId (indexed),
  documentationId: ObjectId,
  diagramType: String,
  mermaidCode: String,
  jsonSchema: Object,
  svgUrl: String,
  pngUrl: String,
  validationStatus: String,
  retryCount: Number,
  reviewRequired: Boolean,
```

```
  createdAt: Date
}
```

Functional Requirements:

- FR-DB-001: Support multi-tenant data isolation with tenantId scoping
- FR-DB-002: Store user accounts, organizations, repositories, documentation, and diagrams
- FR-DB-003: Maintain subscription and billing information
- FR-DB-004: Track usage quotas and enforce limits
- FR-DB-005: Version documentation with change history
- FR-DB-006: Store job queue state and results

Technical Requirements:

- TR-DB-001: MongoDB Atlas for managed cloud database (or self-hosted for enterprise)
- TR-DB-002: Mongoose ODM for schema validation and queries
- TR-DB-003: Compound indexes on tenantId + other fields for query performance
- TR-DB-004: Database response times <100ms for 95% of queries
- TR-DB-005: Automated daily backups with 30-day retention
- TR-DB-006: Implement data retention policies (delete unused data after 90 days on Free tier)

Acceptance Criteria:

- Query performance meets SLA requirements
- Data isolation validated across 100+ test tenants
- Zero cross-tenant data leakage in security testing
- Backup restoration tested and completes within 2 hours

4.1.4 Middleware & Services Layer

Authentication & Authorization:[2]

- FR-AUTH-001: JWT-based authentication with refresh token mechanism
- FR-AUTH-002: Role-based access control (Owner, Admin, Member, Viewer)

- FR-AUTH-003: OAuth integration with GitHub, GitLab, Bitbucket
- FR-AUTH-004: SSO support for enterprise tier (SAML 2.0)
- FR-AUTH-005: API key management for REST API access

Tenant Management:[5][2]

- FR-TEN-001: Tenant identification via subdomain or custom domain
- FR-TEN-002: Automatic tenant context injection in all requests
- FR-TEN-003: Tenant-scoped database queries preventing cross-tenant access
- FR-TEN-004: Organization creation and settings management
- FR-TEN-005: Team member invitation and role assignment

Subscription & Billing:[1]

- FR-SUB-001: Stripe integration for payment processing
- FR-SUB-002: Automated subscription lifecycle management
- FR-SUB-003: Usage-based quota enforcement
- FR-SUB-004: Automatic upgrade/downgrade handling
- FR-SUB-005: Invoice generation and email delivery
- FR-SUB-006: Trial period management (14-day free trial)

5. Multi-Agent System Architecture (MERN Implementation)

5.1 Node.js-Based Agent Orchestration

Architecture Pattern: Microservices-based agent implementation using Node.js workers[7][6]

Orchestrator Service (Node.js + GPT-4o mini):

```
// Orchestrator Service Architecture
class OrchestatorService {
  constructor() {
    this.stateManager = new StateManager(redisClient);
    this.agentRouter = new AgentRouter();
    this.jobQueue = new Bull('documentation-jobs', redisConfig);
```

```
  }

  async processComponent(tenantId, repositoryId, componentPath) {
    // Initialize workflow
    const workflowId = await this.stateManager.createWorkflow();

    // Iterative agent coordination
    for (let iteration = 0; iteration < MAX_ITERATIONS; iteration++) {
      const readerResult = await this.agentRouter.invokeReader(componentPath);
      const searcherResult = await
this.agentRouter.invokeSearcher(readerResult.requests);
      const writerResult = await this.agentRouter.invokeWriter(searcherResult.context);
      const verifierResult = await
this.agentRouter.invokeVerifier(writerResult.documentation);

      if (verifierResult.approved) {
        return await this.finalizeDocumentation(writerResult);
      }

      // Update state for retry
      await this.stateManager.updateWorkflow(workflowId, verifierResult.feedback);
    }
  }
}
```

Functional Requirements:

- FR-ORCH-001: Process documentation jobs from Bull queue asynchronously
- FR-ORCH-002: Coordinate Reader → Searcher → Writer → Verifier workflow
- FR-ORCH-003: Manage workflow state in Redis with fallback to MongoDB
- FR-ORCH-004: Emit real-time progress updates via Socket.io
- FR-ORCH-005: Implement circuit breakers for LLM API calls
- FR-ORCH-006: Support parallel processing of multiple components
- FR-ORCH-007: Graceful shutdown and job recovery

Technical Requirements:

- TR-ORCH-001: Node.js worker processes (cluster mode for multi-core)
- TR-ORCH-002: Bull queue for job management with priorities
- TR-ORCH-003: Redis for state management and caching
- TR-ORCH-004: GPT-4o mini API for coordination decisions
- TR-ORCH-005: Exponential backoff retry strategy (max 3 attempts)

5.2 Reader Agent Service

Implementation: Node.js microservice with CodeT5+ integration via HuggingFace Inference API

Functional Requirements:

- FR-READ-001: Analyze code components and determine documentation needs
- FR-READ-002: Evaluate complexity using cyclomatic and cognitive complexity
- FR-READ-003: Identify internal dependencies via AST parsing
- FR-READ-004: Map architectural relationships between components
- FR-READ-005: Generate structured requests for additional context (JSON format)
- FR-READ-006: Support Python, JavaScript, TypeScript, Java, C++, Go, Rust, C#

Technical Requirements:

- TR-READ-001: Express.js microservice exposing REST endpoints
- TR-READ-002: Integration with CodeT5+ via HuggingFace API or local deployment
- TR-READ-003: Parser libraries: @babel/parser (JS/TS), tree-sitter (multi-language)
- TR-READ-004: Process components within 30 seconds
- TR-READ-005: Cache analysis results in Redis (TTL: 1 hour)

API Endpoint Design:

```
POST /api/agents/reader/analyze
Request: {
  tenantId: string,
  componentPath: string,
  sourceCode: string,
  language: string,
  existingDocs: string?
```

```
}

Response: {
  complexity: { cyclomatic: number, cognitive: number },
  dependencies: { internal: string[], external: string[] },
  architectureMap: object,
  informationRequests: {
    internal: [{ type: string, target: string, reason: string }],
    external: [{ concept: string, query: string }]
  }
}
```

5.3 Searcher Agent Service

Implementation: Dual-mode Node.js service with Gemini and Perplexity API integrations[8][9]

Functional Requirements:

- FR-SEARCH-001: Internal code navigation using Gemini 2.5 Pro
- FR-SEARCH-002: External knowledge retrieval using Perplexity Sonar API
- FR-SEARCH-003: Consolidate retrieved information into structured context
- FR-SEARCH-004: Rank information by relevance to focal component
- FR-SEARCH-005: Cache frequently accessed code components
- FR-SEARCH-006: Trace call sites and dependency relationships

Technical Requirements:

- TR-SEARCH-001: Express.js service with parallel request handling
- TR-SEARCH-002: Gemini 2.5 Pro API for internal analysis (1M+ token context)
- TR-SEARCH-003: Perplexity Sonar API for web-grounded search
- TR-SEARCH-004: Static analysis using tree-sitter or @babel/traverse
- TR-SEARCH-005: Redis caching with 4-hour TTL for external knowledge
- TR-SEARCH-006: Complete searches within 15 seconds

API Endpoint Design:

```
POST /api/agents/searcher/retrieve
Request: {
  tenantId: string,
  repositoryId: string,
  internalRequests: array,
  externalRequests: array
}

Response: {
  internalContext: {
    codeComponents: array,
    callSites: array,
    dependencies: object
  },
  externalKnowledge: {
    concepts: array,
    references: array,
    examples: array
  },
  confidence: number
}
```

5.4 Writer Agent Service

Implementation: Node.js service with GPT-4o integration for fast generation[10]

Functional Requirements:

- FR-WRITE-001: Generate docstrings following language-specific conventions
- FR-WRITE-002: Create comprehensive documentation with multiple sections
- FR-WRITE-003: Generate realistic usage examples
- FR-WRITE-004: Apply templates based on component type
- FR-WRITE-005: Compose Markdown with proper formatting
- FR-WRITE-006: Support custom templates per organization

Technical Requirements:

- TR-WRITE-001: Express.js service with GPT-4o API integration

- TR-WRITE-002: Template engine using Handlebars or EJS
- TR-WRITE-003: Markdown generation with Marked.js
- TR-WRITE-004: Streaming responses for real-time preview
- TR-WRITE-005: Generate documentation within 45 seconds
- TR-WRITE-006: Support concurrent requests (20+ simultaneous)

API Endpoint Design:

```
POST /api/agents/writer/generate
Request: {
  tenantId: string,
  componentInfo: object,
  context: object,
  templateId: string?,
  style: string // 'google', 'numpy', 'jsdoc', etc.
}

Response: {
  docstring: string,
  markdown: string,
  examples: array,
  crossReferences: array,
  metadata: object
}
```

5.5 Verifier Agent Service

Implementation: Node.js service with GPT-4 Turbo for quality evaluation[11]

Functional Requirements:

- FR-VERIFY-001: Evaluate documentation against quality criteria
- FR-VERIFY-002: Check factual accuracy versus source code
- FR-VERIFY-003: Validate completeness of required sections
- FR-VERIFY-004: Generate structured feedback with improvement suggestions
- FR-VERIFY-005: Assign quality scores and confidence levels

- FR-VERIFY-006: Approve or reject with detailed reasoning

Technical Requirements:

- TR-VERIFY-001: Express.js service with GPT-4 Turbo API
- TR-VERIFY-002: Semantic comparison using embeddings
- TR-VERIFY-003: Complete evaluation within 25 seconds
- TR-VERIFY-004: Configurable quality thresholds per organization

API Endpoint Design:

```
POST /api/agents/verifier/evaluate
Request: {
  tenantId: string,
  sourceCode: string,
  generatedDocs: object,
  diagrams: array?,
  qualityThresholds: object?
}

Response: {
  approved: boolean,
  qualityScore: number, // 0-100
  evaluation: {
    informationValue: number,
    detailLevel: number,
    completeness: number,
    accuracy: number
  },
  feedback: array, // specific improvement suggestions
  requiresHumanReview: boolean
}
```

5.6 Diagram Generator Service

Implementation: Node.js service with GPT-4o and Mermaid.js[12][1]

Functional Requirements:

- FR-DIAG-001: Classify appropriate diagram type for component

- FR-DIAG-002: Generate structured JSON schema for diagram
- FR-DIAG-003: Convert JSON to valid Mermaid.js syntax
- FR-DIAG-004: Validate and render diagrams using Mermaid CLI
- FR-DIAG-005: Implement retry logic with error feedback
- FR-DIAG-006: Export to SVG and PNG formats
- FR-DIAG-007: Support all major Mermaid diagram types

Technical Requirements:

- TR-DIAG-001: Express.js service with GPT-4o API
- TR-DIAG-002: Mermaid CLI integration via child_process
- TR-DIAG-003: Image storage in AWS S3 or compatible object storage
- TR-DIAG-004: Generate and render within 20 seconds
- TR-DIAG-005: Support diagrams with up to 100 nodes

API Endpoint Design:

```
POST /api/agents/diagram/generate
Request: {
  tenantId: string,
  componentData: object,
  diagramType: string?, // auto-detect if not specified
  style: object?
}

Response: {
  diagramType: string,
  mermaidCode: string,
  jsonSchema: object,
  svgUrl: string,
  pngUrl: string,
  validationStatus: string,
  requiresReview: boolean
}
```

6. User Interface Design

6.1 Dashboard Views

6.1.1 Home Dashboard

Components:

- Overview cards: Total repositories, components documented, coverage percentage
- Recent activity feed: Latest documentation generations
- Quick actions: Connect repository, generate docs, view analytics
- Notification bell: Status updates and errors
- Repository list with search and filters

User Flow:

1. User logs in → Redirected to home dashboard
2. Sees overview metrics and repository status
3. Can trigger documentation generation with single click
4. Receives real-time updates via WebSocket

6.1.2 Repository Management View

Components:

- Repository list with provider badges (GitHub/GitLab/Bitbucket)
- Connect repository wizard with OAuth flow
- Repository settings: branch selection, language configuration, webhook status
- Documentation coverage visualization (tree map or progress bars)
- Sync button for manual triggering

Features:

- Drag-and-drop to upload code directly
- Filter by language, coverage, last updated
- Bulk actions for multiple repositories

6.1.3 Documentation Viewer

Components:

- Split-pane layout: source code (left) | documentation (right)
- Monaco editor for code display with syntax highlighting
- Rendered Markdown documentation with collapsible sections
- Embedded interactive Mermaid diagrams
- Version history sidebar
- Comment/feedback system
- Export options (PDF, HTML, Markdown)

Features:

- Zoom and pan for diagrams
- Copy code snippets with single click
- Search within documentation
- Cross-reference navigation between components

6.1.4 Analytics Dashboard

Metrics & Visualizations:

- Documentation coverage trend (line chart)
- Components documented by language (pie chart)
- Quality score distribution (histogram)
- Generation time trends (line chart)
- Most documented vs. least documented modules
- Team activity feed

Filters:

- Date range selector
- Repository filter
- Language filter

- Quality threshold filter

6.1.5 Settings Panel

Tabs:

- **Profile**: User profile, password change, API keys
- **Organization**: Team members, roles, subdomain
- **Repositories**: Repository connections, webhooks
- **Templates**: Documentation style preferences, custom templates
- **Integrations**: GitHub/GitLab OAuth, Slack notifications
- **Billing**: Subscription plan, payment method, usage tracking, invoices

6.1.6 Admin Panel (Organization Owners)

Features:

- User management: Invite, remove, change roles
- Usage analytics: Quota consumption, cost projections
- Audit logs: User actions, API calls, data access
- Billing management: Plan upgrades, payment history
- Organization settings: Branding, custom domain

6.2 Component Library

Reusable React Components:[6]

- RepositoryCard: Display repository info with status indicators
- DocumentationCard: Preview documentation with quality badges
- DiagramViewer: Interactive Mermaid diagram renderer
- CodeEditor: Monaco-based code viewer with syntax highlighting
- ProgressTracker: Real-time job progress with WebSocket updates
- QualityBadge: Visual indicator for documentation quality score
- UsageQuotaBar: Progress bar showing quota consumption

- NotificationToast: Toast notifications for async operations

Design System:

- Material-UI or Ant Design for consistent UI components
- TailwindCSS for custom styling
- Responsive grid system (mobile-first design)
- Dark mode support
- Accessibility features (keyboard navigation, screen reader support)

7. API Specification

7.1 REST API Endpoints

Authentication:

```
POST   /api/auth/register           # Register new user
POST   /api/auth/login              # Login and get JWT
POST   /api/auth/logout             # Logout and invalidate token
POST   /api/auth/refresh            # Refresh access token
POST   /api/auth/forgot-password    # Request password reset
POST   /api/auth/reset-password     # Reset password with token
GET    /api/auth/oauth/:provider    # OAuth login (GitHub/GitLab)
GET    /api/auth/oauth/callback     # OAuth callback handler
```

Organizations (Tenants):

```
POST   /api/organizations           # Create organization
GET    /api/organizations/:id       # Get organization details
PUT    /api/organizations/:id       # Update organization
DELETE /api/organizations/:id       # Delete organization
GET    /api/organizations/:id/members  # List team members
POST   /api/organizations/:id/members  # Invite team member
DELETE /api/organizations/:id/members/:userId  # Remove member
PUT    /api/organizations/:id/members/:userId/role  # Update role
```

Repositories:

```
GET    /api/repositories            # List all repositories
POST   /api/repositories            # Connect new repository
GET    /api/repositories/:id        # Get repository details
PUT    /api/repositories/:id        # Update repository settings
DELETE /api/repositories/:id        # Disconnect repository
POST   /api/repositories/:id/sync   # Trigger manual sync
GET    /api/repositories/:id/webhooks  # Get webhook configuration
POST   /api/repositories/:id/webhooks  # Create webhook
```

Documentation:

```
GET    /api/documentation            # List documentation (paginated)
POST   /api/documentation/generate   # Generate documentation
GET    /api/documentation/:id        # Get specific documentation
PUT    /api/documentation/:id        # Update documentation
DELETE /api/documentation/:id        # Delete documentation
POST   /api/documentation/:id/regenerate  # Regenerate documentation
GET    /api/documentation/:id/versions    # Get version history
GET    /api/documentation/search    # Search across documentation
```

Diagrams:

```
GET    /api/diagrams                # List diagrams
POST   /api/diagrams/generate       # Generate diagram
GET    /api/diagrams/:id            # Get diagram details
PUT    /api/diagrams/:id            # Update diagram
DELETE /api/diagrams/:id            # Delete diagram
GET    /api/diagrams/:id/render     # Render to SVG/PNG
POST   /api/diagrams/:id/regenerate # Regenerate diagram
```

Jobs:

```
GET    /api/jobs                    # List jobs (with filters)
GET    /api/jobs/:id                # Get job status
POST   /api/jobs/:id/cancel         # Cancel running job
GET    /api/jobs/:id/logs           # Get job execution logs
```

Analytics:

```
GET    /api/analytics/overview      # Dashboard overview metrics
GET    /api/analytics/coverage      # Documentation coverage stats
GET    /api/analytics/quality       # Quality score distribution
GET    /api/analytics/usage         # Usage and quota tracking
GET    /api/analytics/trends        # Historical trends
```

Subscriptions & Billing:

```
GET    /api/subscriptions           # Get current subscription
POST   /api/subscriptions/upgrade   # Upgrade plan
POST   /api/subscriptions/downgrade # Downgrade plan
POST   /api/subscriptions/cancel    # Cancel subscription
GET    /api/billing/invoices        # List invoices
GET    /api/billing/usage           # Current billing period usage
POST   /api/billing/payment-method  # Update payment method
```

Webhooks (External integrations):

```
POST   /api/webhooks/github         # GitHub webhook receiver
POST   /api/webhooks/gitlab         # GitLab webhook receiver
POST   /api/webhooks/bitbucket      # Bitbucket webhook receiver
```

7.2 WebSocket Events

Real-time Updates:[1]

```
// Client → Server
socket.emit('subscribe:job', { jobId })
socket.emit('subscribe:repository', { repositoryId })

// Server → Client
socket.on('job:progress', { jobId, progress, stage, message })
socket.on('job:completed', { jobId, result })
socket.on('job:failed', { jobId, error })
socket.on('documentation:updated', { documentationId, changes })
socket.on('repository:synced', { repositoryId, componentsCount })
```

8. Technical Architecture (MERN Stack)

8.1 Complete Technology Stack

Frontend (React.js):[6][1]

- React 18+ with TypeScript
- Redux Toolkit (state management)
- React Query (API data fetching & caching)
- React Router v6 (routing)
- Material-UI or Ant Design (UI components)
- TailwindCSS (utility-first styling)
- Monaco Editor (code display)
- Mermaid.js (diagram rendering)
- Recharts (analytics visualization)
- Socket.io-client (real-time updates)
- Axios (HTTP client)
- React Hook Form (form management)
- Yup or Zod (client-side validation)

Backend (Node.js + Express.js):[2][1]

- Node.js 20+ LTS
- Express.js 4.x with TypeScript
- Mongoose (MongoDB ODM)
- Passport.js (authentication)
- jsonwebtoken (JWT handling)
- bcryptjs (password hashing)
- Bull (job queue with Redis)
- Socket.io (WebSocket server)
- Stripe SDK (payment processing)

- Axios (external API calls)
- Joi or Zod (request validation)
- Winston (logging)
- helmet (security headers)
- cors (CORS management)
- express-rate-limit (rate limiting)
- multer (file upload handling)

Database & Caching:[2]

- MongoDB Atlas (primary database)
- Mongoose (ODM with schemas)
- Redis (caching & job queue)
- AWS S3 or Cloudflare R2 (object storage for diagrams/exports)

LLM Integration Layer:

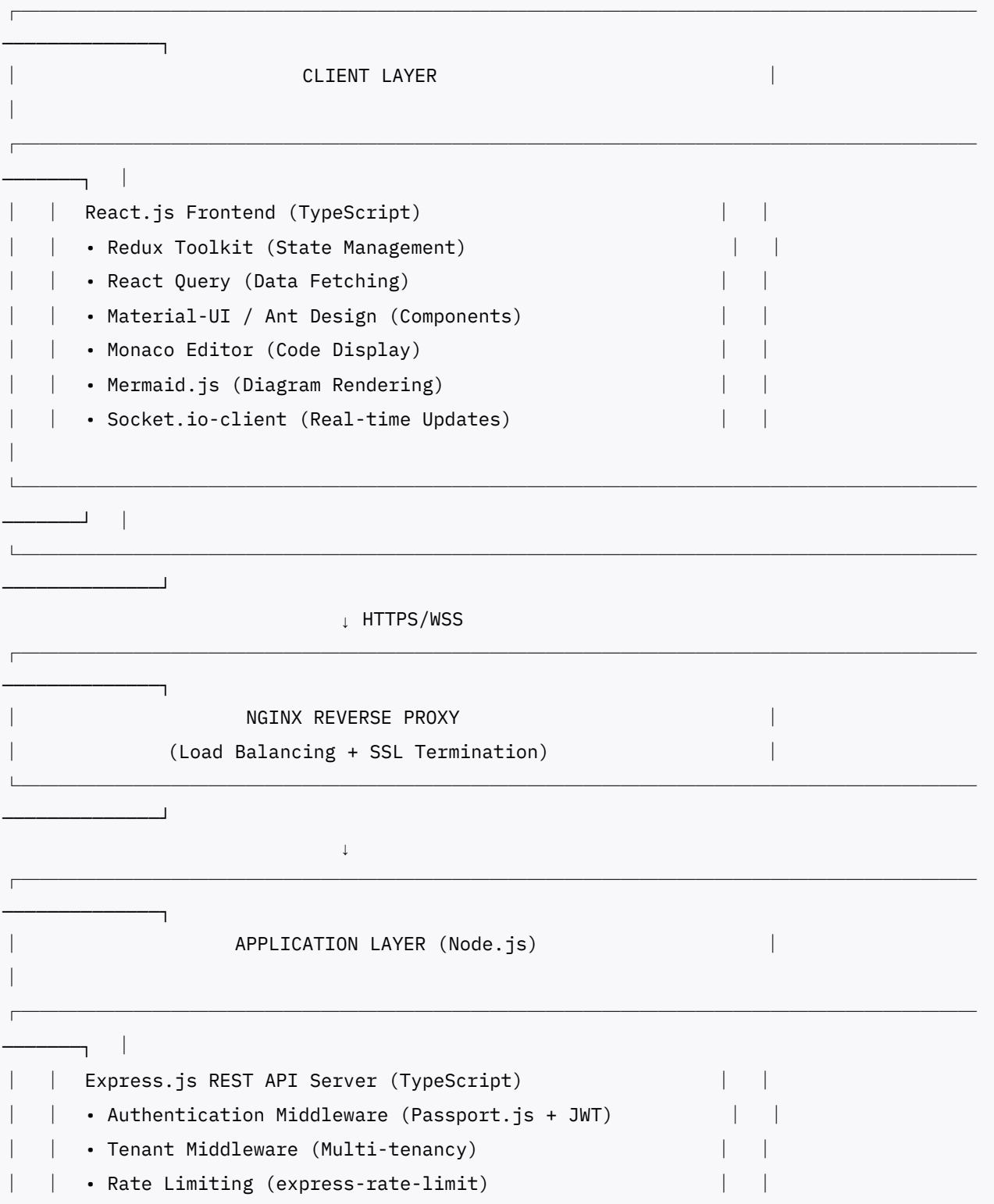
- OpenAI Node.js SDK (GPT-4o, GPT-4 Turbo, GPT-4o mini)
- Google AI Node.js SDK (Gemini 2.5 Pro)
- Perplexity API (Sonar search)
- HuggingFace Inference API (CodeT5+)

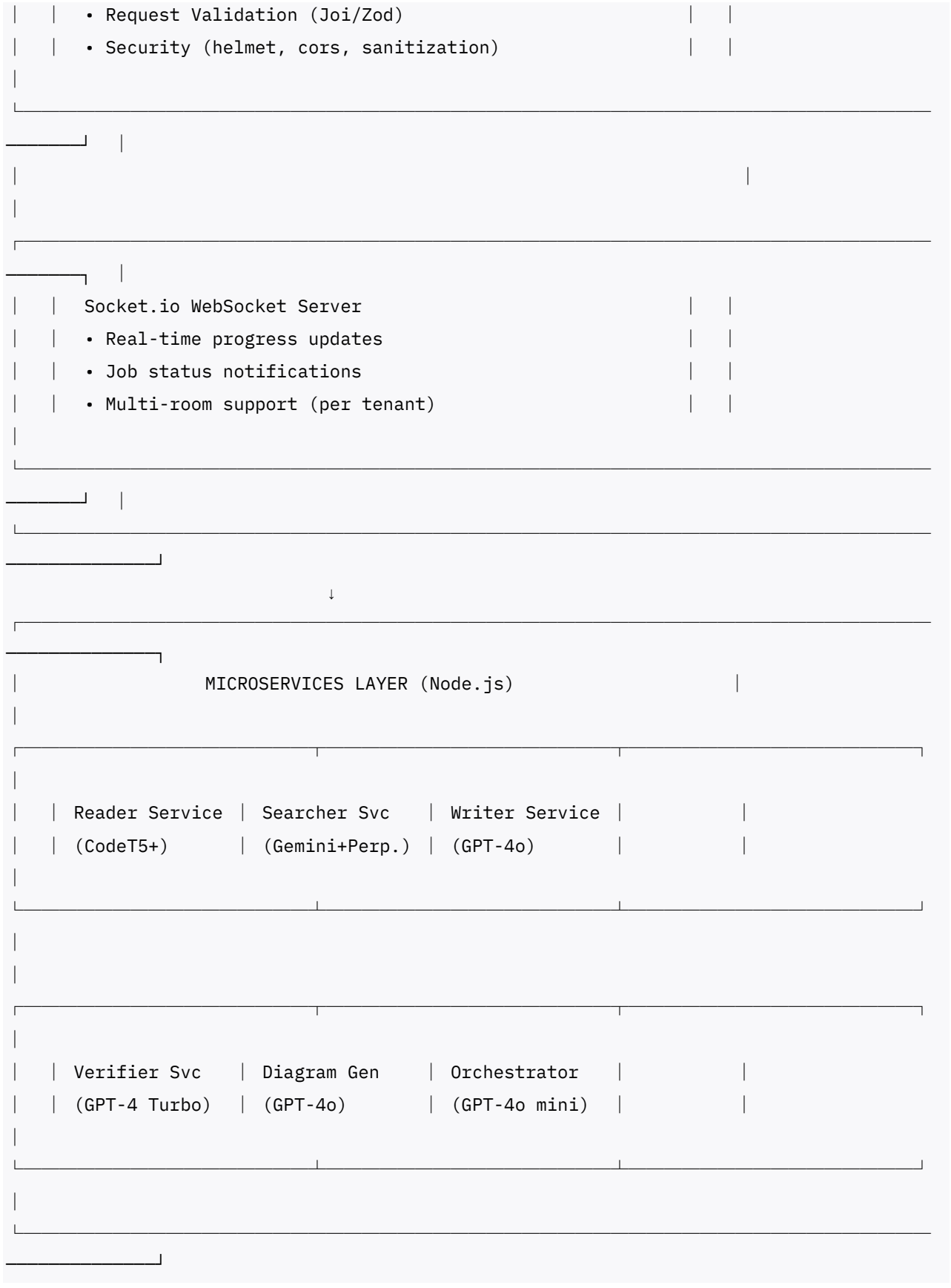
DevOps & Infrastructure:[6]

- Docker & Docker Compose (containerization)
- Kubernetes (orchestration) or AWS ECS
- Nginx (reverse proxy & load balancing)
- GitHub Actions (CI/CD)
- PM2 (Node.js process management)
- New Relic or DataDog (monitoring)
- Sentry (error tracking)
- Vercel or Netlify (frontend hosting)

- AWS, Google Cloud, or DigitalOcean (backend hosting)

8.2 System Architecture Diagram (MERN Stack)




• Request Validation (Joi/Zod)
• Security (helmet, cors, sanitization)
Socket.io WebSocket Server
• Real-time progress updates
• Job status notifications
• Multi-room support (per tenant)
MICROSERVICES LAYER (Node.js)
Reader Service
(CodeT5+)
Searcher Svc
(Gemini+Perp.)
Writer Service
(GPT-4o)
Verifier Svc
(GPT-4 Turbo)
Diagram Gen
(GPT-4o)
Orchestrator
(GPT-4o mini)


JOB QUEUE LAYER (Bull + Redis)
Bull Queue (Redis-backed)
• Documentation Generation Queue
• Diagram Generation Queue
• Repository Sync Queue
• Priority Queue for Premium Users
DATA LAYER
MongoDB Atlas
(Primary Database)
• Users
• Tenants
• Repositories
• Documentation
• Diagrams
• Jobs
Redis Cache
• Session Store
• Job Queue State
• Analysis Cache
• Rate Limit Counters
AWS S3 / Cloudflare R2
(Object Storage)
• Diagram Images (SVG/PNG)
• Exported Documentation
• User Uploads

8.3 Multi-Tenant Architecture

Tenant Isolation Strategy:[5][2]

Model: Shared database, shared schema with `tenantId` field (optimal for SaaS scalability)[2]

Middleware Implementation:

```
// Tenant middleware for all requests
const tenantMiddleware = async (req, res, next) => {
  // Extract tenant from subdomain or custom domain
  const hostname = req.hostname;
  const subdomain = hostname.split('.')[^0];

  // Lookup tenant in database
  const tenant = await Tenant.findOne({ subdomain });

  if (!tenant) {
    return res.status(404).json({ error: 'Organization not found' });
  }

  // Check subscription status
  if (tenant.subscription.status !== 'active') {
```

```
    return res.status(403).json({ error: 'Subscription inactive' });
  }

  // Inject tenant context
  req.tenant = tenant;
  req.tenantId = tenant._id;

  next();
};

// Mongoose plugin for automatic tenantId injection
const tenantPlugin = (schema) => {
  schema.add({ tenantId: { type: Schema.Types.ObjectId, required: true, index: true }
});

  schema.pre('save', function(next) {
    if (!this.tenantId && this.$__.context?.tenantId) {
      this.tenantId = this.$__.context.tenantId;
    }
    next();
  });

  schema.pre(/^find/, function(next) {
    if (this.options.tenantId) {
      this.where({ tenantId: this.options.tenantId });
    }
    next();
  });
};
```

Data Isolation:[2]

- All queries automatically scoped to tenantId
- Compound indexes: `{ tenantId: 1, <other_field>: 1 }`
- Pre-save hooks ensure tenantId is always set
- Query hooks automatically filter by tenantId
- Validation at API level prevents cross-tenant access

Performance Optimization:[5]

- Connection pooling for MongoDB (min: 10, max: 100 per instance)
- Redis caching for frequently accessed data (TTL: 5 minutes)
- Database indexes on tenantId + query fields
- Pagination for large result sets (default: 50 items/page)
- Query result caching with cache invalidation on updates

8.4 Scalability Architecture

Horizontal Scaling:[7][6]

- **API Servers**: 3+ Node.js instances behind load balancer
- **Worker Processes**: Auto-scale based on queue depth (2-20 workers)
- **Database**: MongoDB Atlas auto-scaling (M30 to M100 cluster)
- **Caching**: Redis Cluster for distributed caching

Load Balancing Strategy:[6]

- Round-robin for API requests
- Queue-based distribution for background jobs
- Sticky sessions for WebSocket connections

Resource Limits per Tenant:

```
const TIER_LIMITS = {
  free: {
    componentsPerMonth: 100,
    maxRepositories: 1,
    maxTeamMembers: 1,
    maxConcurrentJobs: 1,
    apiRateLimit: 10 // requests per minute
  },
  starter: {
    componentsPerMonth: 1000,
    maxRepositories: 5,
    maxTeamMembers: 5,
    maxConcurrentJobs: 3,
```

```
    apiRateLimit: 100
  },
  professional: {
    componentsPerMonth: 10000,
    maxRepositories: 20,
    maxTeamMembers: 20,
    maxConcurrentJobs: 10,
    apiRateLimit: 1000
  },
  team: {
    componentsPerMonth: 50000,
    maxRepositories: -1, // unlimited
    maxTeamMembers: 50,
    maxConcurrentJobs: 20,
    apiRateLimit: 5000
  },
  enterprise: {
    componentsPerMonth: -1, // unlimited
    maxRepositories: -1,
    maxTeamMembers: -1,
    maxConcurrentJobs: 50,
    apiRateLimit: 10000
  }
};
```

9. Security Requirements (SaaS-Specific)

9.1 Authentication & Authorization

Requirements:[2]

- NFR-SEC-001: JWT-based authentication with 15-minute access tokens
- NFR-SEC-002: Refresh tokens with 30-day expiration stored in HTTP-only cookies
- NFR-SEC-003: Password requirements: min 8 chars, 1 uppercase, 1 number, 1 special char
- NFR-SEC-004: bcrypt password hashing with salt rounds: 12

- NFR-SEC-005: OAuth 2.0 integration with GitHub, GitLab, Bitbucket
- NFR-SEC-006: SSO support via SAML 2.0 (Enterprise tier)
- NFR-SEC-007: Role-based access control with 4 roles: Owner, Admin, Member, Viewer
- NFR-SEC-008: API key management with rotation capabilities

9.2 Data Security

Requirements:[5][2]

- NFR-SEC-009: All communications over HTTPS/TLS 1.3
- NFR-SEC-010: Data encryption at rest (MongoDB encryption)
- NFR-SEC-011: Sensitive environment variables stored in secure vault (AWS Secrets Manager)
- NFR-SEC-012: Input sanitization to prevent NoSQL injection
- NFR-SEC-013: XSS protection via Content Security Policy headers
- NFR-SEC-014: CSRF protection for state-changing operations
- NFR-SEC-015: Secure session management with Redis
- NFR-SEC-016: Code repository credentials encrypted in database

9.3 Multi-Tenant Security

Requirements:[2]

- NFR-SEC-017: Mandatory tenantId validation on all database queries
- NFR-SEC-018: Middleware preventing cross-tenant data access
- NFR-SEC-019: Audit logging of all data access by user and tenant
- NFR-SEC-020: Regular security testing for tenant isolation
- NFR-SEC-021: Per-tenant data export and deletion capabilities (GDPR)

9.4 Compliance & Privacy

Requirements:[5]

- NFR-SEC-022: SOC 2 Type II compliance by Month 12
- NFR-SEC-023: GDPR compliance with data processing agreements

- NFR-SEC-024: Privacy policy and terms of service
- NFR-SEC-025: User consent for data processing
- NFR-SEC-026: Data retention policies configurable per tenant
- NFR-SEC-027: Right to deletion (GDPR Article 17)
- NFR-SEC-028: Data portability (export all user data)

10. Non-Functional Requirements (SaaS Platform)

10.1 Performance Requirements

- NFR-P-001: React app initial load <3 seconds (3G connection)
- NFR-P-002: API response time <500ms for 95% of requests (P95)
- NFR-P-003: WebSocket message latency <100ms
- NFR-P-004: Documentation generation: <5 minutes for 95% of components
- NFR-P-005: Support 1,000 concurrent users
- NFR-P-006: Database query response time <100ms (P95)
- NFR-P-007: Redis cache hit ratio >80%

10.2 Availability & Reliability

- NFR-A-001: System uptime 99.9% (excluding planned maintenance)
- NFR-A-002: Planned maintenance windows <4 hours/month
- NFR-A-003: Zero-downtime deployments using blue-green strategy
- NFR-A-004: Automatic failover for critical services
- NFR-A-005: Data backup every 6 hours with 30-day retention
- NFR-A-006: Disaster recovery RTO (Recovery Time Objective): <4 hours
- NFR-A-007: Disaster recovery RPO (Recovery Point Objective): <1 hour

10.3 Scalability Requirements

- NFR-S-001: Support 10,000+ organizations on platform

- NFR-S-002: Handle 1M+ documentation generations per month
- NFR-S-003: Auto-scale API servers based on CPU >70% threshold
- NFR-S-004: Auto-scale worker processes based on queue depth >100 jobs
- NFR-S-005: Database cluster auto-scaling to handle growth
- NFR-S-006: CDN distribution for static assets (global <200ms latency)

10.4 Monitoring & Observability

- NFR-M-001: Real-time application performance monitoring (APM)
- NFR-M-002: Error tracking with stack traces and context
- NFR-M-003: Custom metrics dashboard for operations team
- NFR-M-004: Alerting for critical errors, downtime, quota breaches
- NFR-M-005: Request tracing across microservices
- NFR-M-006: Database performance monitoring and slow query detection
- NFR-M-007: LLM API usage and cost tracking

10.5 Usability Requirements (SaaS)

- NFR-U-001: User onboarding completion within 10 minutes
- NFR-U-002: First documentation generated within 5 minutes of signup
- NFR-U-003: In-app tutorials and tooltips for key features
- NFR-U-004: Comprehensive help center and documentation
- NFR-U-005: Live chat support for paid tiers
- NFR-U-006: Mobile-responsive design (viewport 768px+)

11. Feature Requirements (MERN Stack SaaS)

11.1 User Authentication & Onboarding

Registration Flow:[1][2]

- FR-AUTH-001: Email/password registration with email verification

- FR-AUTH-002: OAuth registration via GitHub, GitLab, Google
- FR-AUTH-003: Organization creation during signup (subdomain selection)
- FR-AUTH-004: 14-day free trial (no credit card required for Starter tier)
- FR-AUTH-005: Interactive onboarding wizard (3-step: profile → connect repo → generate first doc)

Login & Session Management:

- FR-AUTH-006: Email/password login with JWT tokens
- FR-AUTH-007: "Remember me" option with extended refresh token
- FR-AUTH-008: Password reset via email link
- FR-AUTH-009: Two-factor authentication (2FA) via TOTP (Professional tier+)
- FR-AUTH-010: Session management with automatic timeout after 30 days inactivity

Technical Implementation:

```
// Frontend: React login component
const LoginPage = () => {
  const [credentials, setCredentials] = useState({ email: '', password: '' });
  const { mutate: login, isLoading } = useMutation(authApi.login, {
    onSuccess: (data) => {
      localStorage.setItem('accessToken', data.accessToken);
      // Refresh token stored in HTTP-only cookie
      navigate('/dashboard');
    }
  });

  return <LoginForm onSubmit={login} isLoading={isLoading} />;
};

// Backend: Express auth route
router.post('/login', async (req, res) => {
  const { email, password } = req.body;

  const user = await User.findOne({ email });
  if (!user || !await bcrypt.compare(password, user.passwordHash)) {
    return res.status(401).json({ error: 'Invalid credentials' });
  }
```

```
  const accessToken = jwt.sign({ userId: user._id, tenantId: user.tenantId },
    process.env.JWT_SECRET, { expiresIn: '15m' });
  const refreshToken = jwt.sign({ userId: user._id },
    process.env.JWT_REFRESH_SECRET, { expiresIn: '30d' });

  res.cookie('refreshToken', refreshToken, { httpOnly: true, secure: true });
  res.json({ accessToken, user: { id: user._id, email: user.email, role: user.role } });
});
```

11.2 Repository Management

Repository Connection:[1]

- FR-REPO-001: OAuth flow to connect GitHub repositories
- FR-REPO-002: GitLab and Bitbucket integration
- FR-REPO-003: Repository selection with multi-select support
- FR-REPO-004: Branch selection (default: main/master)
- FR-REPO-005: Language detection with manual override
- FR-REPO-006: Webhook automatic installation for push events
- FR-REPO-007: Manual code upload via ZIP file or drag-and-drop

Repository Settings:

- FR-REPO-008: Configure documentation style per repository
- FR-REPO-009: Select specific directories to document (include/exclude patterns)
- FR-REPO-010: Schedule automatic regeneration (daily, weekly, on-push)
- FR-REPO-011: Notification preferences (email, Slack, in-app)

Frontend Component:

```
// Repository connection wizard
const ConnectRepositoryWizard = () => {
  const [step, setStep] = useState(1);
  const [provider, setProvider] = useState('github');
  const [selectedRepos, setSelectedRepos] = useState([]);

  const { data: repos, isLoading } = useQuery(
```

```
    ['repositories', provider],
    () => fetchUserRepositories(provider)
  );

  const connectRepoMutation = useMutation(connectRepository, {
    onSuccess: () => {
      showNotification('Repository connected successfully');
      navigate('/dashboard/repositories');
    }
  });

  return (
    <Wizard currentStep={step}>
      <Step1 provider={provider} setProvider={setProvider} />
      <Step2 repos={repos} selectedRepos={selectedRepos}
setSelectedRepos={setSelectedRepos} />
      <Step3 onSubmit={() => connectRepoMutation.mutate(selectedRepos)} />
    </Wizard>
  );
};
```

11.3 Documentation Generation Workflow

Manual Generation:

- FR-DOC-001: Select repository and components to document
- FR-DOC-002: Preview code and existing documentation
- FR-DOC-003: Trigger generation with single click
- FR-DOC-004: Real-time progress tracking with WebSocket updates
- FR-DOC-005: View results immediately upon completion
- FR-DOC-006: Edit generated documentation inline
- FR-DOC-007: Approve or request regeneration

Automatic Generation:

- FR-DOC-008: Trigger on Git push via webhook
- FR-DOC-009: Detect changed files and affected components

- FR-DOC-010: Auto-queue documentation jobs for changed components
- FR-DOC-011: Send notifications on completion
- FR-DOC-012: Auto-commit to documentation branch or create PR

Bulk Operations:

- FR-DOC-013: Generate documentation for entire repository
- FR-DOC-014: Regenerate all documentation (version updates)
- FR-DOC-015: Batch export to PDF, HTML, or Markdown archive

Real-time Updates UI:

```
// WebSocket integration for live updates
const DocumentationGenerationTracker = ({ jobId }) => {
  const [progress, setProgress] = useState(0);
  const [stage, setStage] = useState('Initializing');

  useEffect(() => {
    socket.emit('subscribe:job', { jobId });

    socket.on('job:progress', (data) => {
      setProgress(data.progress);
      setStage(data.stage);
    });

    socket.on('job:completed', (data) => {
      setProgress(100);
      showNotification('Documentation generated successfully');
      refetchDocumentation();
    });

    return () => socket.emit('unsubscribe:job', { jobId });
  }, [jobId]);

  return (
    <ProgressCard>
      <ProgressBar value={progress} />
      <StatusText>{stage}</StatusText>
    </ProgressCard>
```

```
  );
};
```

11.4 Diagram Generation & Management

Diagram Creation:

- FR-DIAG-001: Automatic diagram type detection (flowchart, sequence, class, etc.)
- FR-DIAG-002: Generate diagrams during documentation creation
- FR-DIAG-003: Standalone diagram generation from code selection
- FR-DIAG-004: Support all major Mermaid.js diagram types
- FR-DIAG-005: Custom diagram styling and themes

Diagram Editor:

- FR-DIAG-006: Live preview of Mermaid code
- FR-DIAG-007: Syntax highlighting for Mermaid code
- FR-DIAG-008: Drag-and-drop node positioning (for supported types)
- FR-DIAG-009: Export to SVG, PNG, PDF
- FR-DIAG-010: Share diagrams via public link

Validation & Quality:

- FR-DIAG-011: Automated syntax validation before saving
- FR-DIAG-012: Retry logic with error feedback (max 3 attempts)
- FR-DIAG-013: Fallback to template library on failures
- FR-DIAG-014: Human review workflow for complex diagrams
- FR-DIAG-015: Version history for diagrams

React Component:

```
// Interactive diagram viewer
const DiagramViewer = ({ diagramId }) => {
  const { data: diagram } = useQuery(['diagram', diagramId], fetchDiagram);
  const [zoom, setZoom] = useState(1);
```

```
  return (
    <DiagramContainer>
      <DiagramToolbar>
        <ZoomControls zoom={zoom} setZoom={setZoom} />
        <ExportButton formats={['SVG', 'PNG', 'PDF']} />
        <EditButton onClick={() => navigate(`/diagrams/${diagramId}/edit`)} />
      </DiagramToolbar>
      <MermaidRenderer
        code={diagram.mermaidCode}
        zoom={zoom}
        theme={diagram.style}
      />
    </DiagramContainer>
  );
};
```

11.5 Team Collaboration Features

Team Management:

- FR-TEAM-001: Invite team members via email
- FR-TEAM-002: Assign roles with granular permissions
- FR-TEAM-003: View team activity feed
- FR-TEAM-004: Team member list with status indicators

Collaboration:

- FR-COLLAB-001: Comment on documentation sections
- FR-COLLAB-002: Suggest improvements to generated docs
- FR-COLLAB-003: Share documentation via public links
- FR-COLLAB-004: Real-time collaboration indicators (who's viewing)
- FR-COLLAB-005: Notification system (email + in-app)

Permissions Matrix:

Action	Owner	Admin	Member	Viewer

View documentation	✓	✓	✓	✓
Generate documentation	✓	✓	✓	✗
Edit documentation	✓	✓	✓	✗
Connect repositories	✓	✓	✗	✗
Manage team members	✓	✓	✗	✗
Manage billing	✓	✗	✗	✗
Delete organization	✓	✗	✗	✗

11.6 Analytics & Insights

Dashboard Metrics:

- FR-ANALYTICS-001: Documentation coverage percentage by repository
- FR-ANALYTICS-002: Components documented over time (trend chart)
- FR-ANALYTICS-003: Average quality score distribution
- FR-ANALYTICS-004: Most/least documented modules
- FR-ANALYTICS-005: Generation time trends
- FR-ANALYTICS-