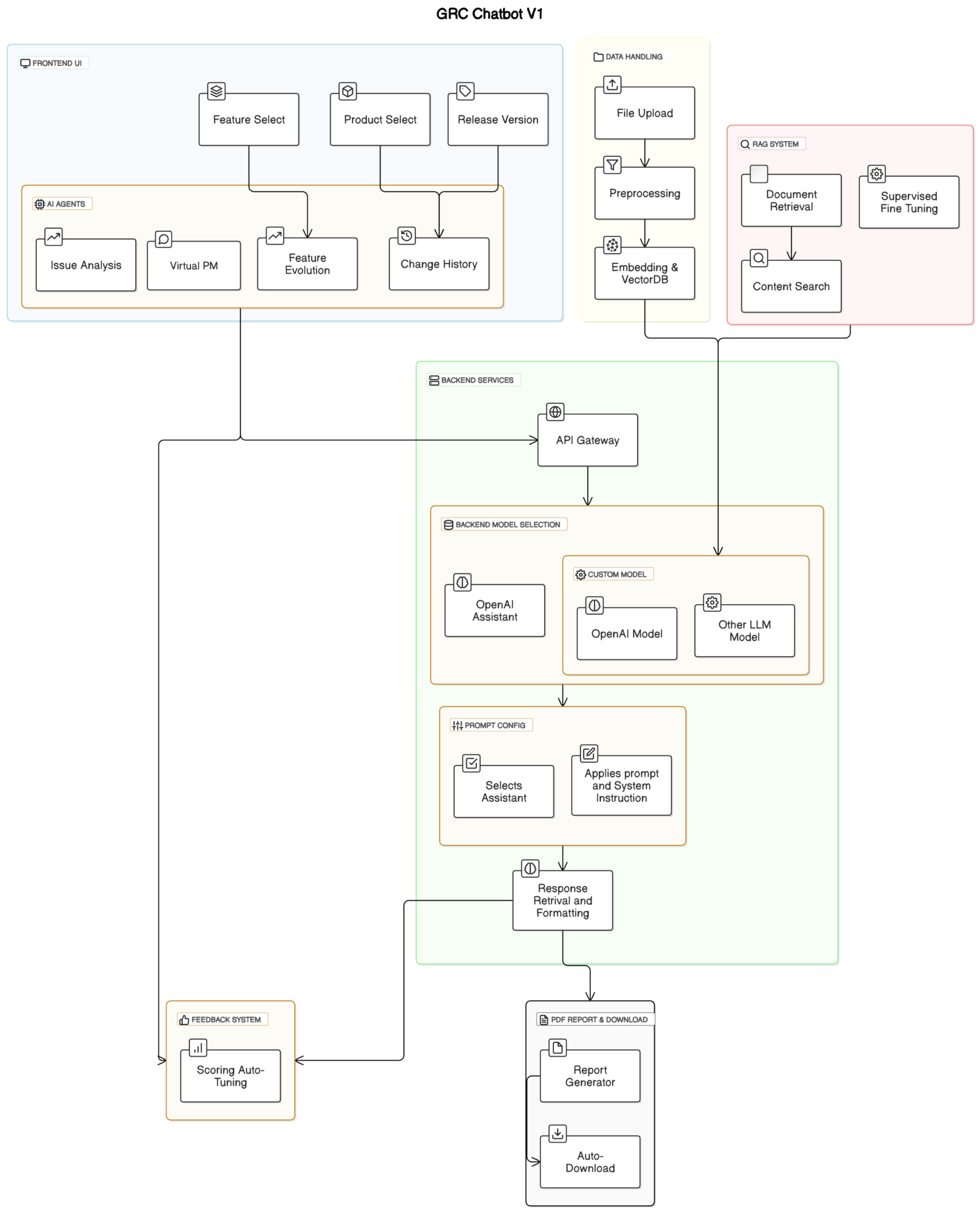
High-Level Design (HLD) for Doc and Support Agent V1

16falsenonelisttrue



1. System Overview

The Doc and Support Agent system provides an AI-driven, document-embedded virtual assistant and report generation platform. It consists of two primary components:

1. **Doc Agent** - Focused on document analysis, report generation, and virtual PM functionality

2. **Support Agent** - Focused on case management, defect analysis, and automated fixes

The system integrates with Confluence, SharePoint, and JIRA via an MCP server, supports RAG-based querying, and enables cross-product communication via Agent-to-Agent (A2A) interactions.

2. Doc Agent Architecture

2.1 Core Capabilities

- **Change History Report Generation**: Automatically generates reports summarizing product changes across release versions
- **Feature Evolution Report Generation**: Tracks and documents the evolution of specific features across releases
- **Virtual PM (Chatbot)**: Context-aware virtual assistant that provides Q&A functionality based on embedded documents

2.2 User Interface Components

- **Doc Agent UX Portal**: Dedicated user interface for Doc Agent functionality
  - Feature/Product Selection Panel
  - Report Type Selection (Change History, Feature Evolution)
  - Virtual PM Chat Interface
  - Document Upload Interface

2.3 Data Ingestion Flow

- **Manual File Upload**:
  - Web interface for uploading documents
  - Support for multiple file formats (PDF, DOCX, TXT, etc.)
  - Direct API calls to Vector DB upload endpoints
- **Confluence Integration**:
  - MCP server-based connector
  - Auto-download of content with configurable schedules
  - Delta-sync capability to fetch only updated documents
  - Automatic chunking and embedding
- **SharePoint Integration**:
  - MCP server-based connector
  - Auto-download with delta updates
  - Permission-based access control
  - Document versioning support

2.4 Document Processing Pipeline

1. **Preprocessing**:
    - Format normalization
    - Content extraction
    - Metadata tagging (source, date, product, version)
    - Chunking with configurable size and overlap
2. **Embedding & Storage**:
    - Vector embedding generation
    - Metadata association
    - Storage in Vector DB with configurable backend (OpenAI, Pinecone, etc.)
3. **Indexing**:
    - Creation of efficient search indices
    - Cross-reference mapping between documents
    - Version tracking metadata

2.5 Agent Model Architecture

- **Product-Specific Sub-Agents**:
    - Each product has its own dedicated OpenAI Assistant
    - Independent knowledge bases per product
    - Specialized prompt and System instruction templates for each product's domain
    - **Note**: Current initial application with Platform + 5 Products. Other products dedicated OpenAI Assistant needs to be built.
- **Agent-to-Agent Communication (A2A)**:
    - Communication protocol between product sub-agents
    - Query routing based on product expertise
    - Response aggregation and deduplication
    - Conflict resolution for contradictory information
- **Model Selection Framework**:
    - Current: OpenAI Assistant-based implementation
    - Future: Migration path to Custom LLM model (e.g. Llama4 LLM)
    - Support for Specialized Language Models (SLMs) for specific tasks
    - Dynamic model selection based on query type and complexity

2.6 RAG (Retrieval-Augmented Generation) System

- **Document Retrieval**:
  - Semantic search over embedded knowledge
  - Multi-vector search with reranking
  - Relevance scoring and thresholding
- **Context Assembly**:
  - Smart document chunking and stitching
  - Context window optimization
  - Version-aware document prioritization
- **Supervised Fine-Tuning**:
  - Enhance retrieval precision via feedback loops
  - Continuous model improvement based on user interactions

3. Support Agent Architecture

3.1 Core Capabilities (Future Development)

- **JIRA Integration**: Fetches issue metadata, RCA, tickets for vector enrichment
- **Log Analyzer Sub-agent**: Analyzes system logs to identify patterns and issues
- **Defect Analysis**: Comprehensive analysis using JIRA data, logs, and code repositories
- **Automated Fix Planning**: Generates implementation plans with impact analysis
- **Code Fix Implementation**: Performs fixes based on manual confirmation

3.2 JIRA Integration

- **MCP Server Connection**:
  - Secure API-based integration with JIRA
  - Real-time and scheduled sync options
  - Bidirectional data flow (read issues, create/update tickets)
- **Issue Metadata Processing**:
  - Extraction of issue details, severity, impact
  - Association with relevant documentation
  - Historical pattern analysis

3.3 Log Analysis Framework

- **Log Collector**:

  - Multi-source log aggregation
  - Format normalization
  - Time-series alignment
- **Pattern Recognition Engine**:
  - ML-based anomaly detection
  - Error pattern classification
  - Root cause correlation
- **Analysis Output**:
  - Structured issue summaries
  - Visual pattern representations
  - Temporal analysis of recurring issues

3.4 Defect Analysis Pipeline

1. **Data Collection**:
   - Pull relevant JIRA issues
   - Gather associated logs
   - Access code repositories
2. **Comprehensive Analysis**:
   - Cross-reference issues with logs and code
   - Identify potential failure points
   - Pattern matching with historical issues
3. **Root Cause Analysis**:
   - Generate potential root causes
   - Confidence scoring for each hypothesis
   - Evidence collection for verification

3.5 Fix Implementation Framework

- **Impact Analysis**:
  - Dependency mapping
  - Risk assessment
  - Performance impact evaluation
- **Fix Plan Generation**:
  - Code change recommendations

    - Testing strategy
    - Rollback contingencies
- **Automated Implementation**:
    - Code changes with manual approval gates
    - Automated test execution
    - Documentation of changes

4. Shared System Components

4.1 MCP Server

- **Central Integration Hub**:
    - Common platform for all external system connections
    - Authentication and access control
    - Rate limiting and quota management
- **Sync Management**:
    - Delta detection and efficient updates
    - Change tracking and notification
    - Conflict resolution

4.2 API Gateway

- **Request Routing**:
    - Service discovery
    - Load balancing
    - Request/response logging
- **Security**:
    - Authentication
    - Authorization
    - Input validation

4.3 Backend Services

- **Model Selection**:
    - Dynamic routing between OpenAI Assistants, Llama4, and SLMs
    - Cost optimization algorithms
    - Performance monitoring

- **Prompt Configuration**:
    - Task-specific prompt libraries
    - Dynamic prompt assembly
    - Context window optimization
- **Response Processing**:
    - Output formatting and structuring
    - Citation and source tracking
    - Quality assurance checks

4.4 Report Generation System

- **Template Engine**:
    - Customizable report layouts
    - Dynamic content assembly
    - Brand compliance
- **PDF Generation**:
    - High-quality document rendering
    - Interactive elements support
    - Accessibility compliance
- **Auto-Download**:
    - Triggered download on completion
    - Email delivery options
    - Secure storage with expiration

4.5 Feedback System

- **User Feedback Collection**:
    - Rating interface
    - Structured feedback forms
    - Implicit satisfaction tracking
- **RLHF (Reinforcement Learning from Human Feedback)**:
    - Feedback data collection pipeline
    - Model tuning framework
    - Performance monitoring

5. Communication Flows

5.1 Doc Agent Primary Flow

1. User selects product and feature in Doc Agent UX
2. Request routed through API Gateway to Doc Agent backend
3. Doc Agent determines request type:
    - Change History Report
    - Feature Evolution Report
    - Virtual PM query
4. For reports:
    - Relevant documents retrieved via RAG
    - Report template selected
    - Content assembled and formatted
    - PDF generated and downloaded
5. For Virtual PM:
    - Query processed and enhanced
    - Product-specific sub-agent activated
    - If needed, A2A communication initiated
    - Response generated and returned to UI

5.2 Data Upload Flow

1. User selects upload method:
    - Manual file upload
    - Confluence connection
    - SharePoint connection
2. For manual upload:
    - Files are processed, chunked, and embedded
    - Vectors and metadata stored in Vector DB
3. For Confluence/SharePoint:
    - MCP server establishes connection
    - Delta sync identifies new/changed content
    - Documents are processed, chunked, and embedded
    - Vectors and metadata stored in Vector DB

5.3 Support Agent Flow (Future)

1. User submits issue or selects existing JIRA ticket
2. Support Agent retrieves relevant context:
   - JIRA issue details
   - System logs
   - Related documentation
   - Code repositories
3. Log Analyzer sub-agent processes relevant logs
4. Defect analysis combines all sources to identify root cause
5. Fix planning generates implementation options with impact analysis
6. Upon approval, automated fix implementation with verification

5.4 A2A Communication Flow

1. Primary agent receives query that spans multiple products
2. Primary agent:
   - Decomposes query into product-specific sub-queries
   - Identifies target sub-agents
3. Sub-queries routed to relevant product sub-agents
4. Each sub-agent processes its query and returns results
5. Primary agent:
   - Aggregates responses
   - Resolves conflicts
   - Synthesizes comprehensive answer
6. Final response returned to user

6. Technology Stack

6.1 Frontend

- React for web interfaces
- Material UI for components
- WebSocket for real-time chat capabilities

6.2 Backend

- FastAPI for API services (planned migration)

- Python for main processing logic
- Node.js for MCP server

6.3 AI Models

- OpenAI Assistant API (current)
- Llama4 LLM (planned)
- Specialized Language Models for specific tasks

6.4 Data Storage

- Vector Database (configurable: OpenAI, Oracle23AI, Pinecone)
- Oracle23AI for metadata and operational data
- Redis for caching

6.5 Integration

- REST APIs for external system connections
- OAuth/API Keys for authentication
- Webhooks for event-driven updates

7. Deployment Architecture

7.1 Infrastructure

- Containerized services with Docker
- Kubernetes for orchestration
- Cloud-native deployment (AWS/Azure/GCP)

7.2 Scalability

- Horizontal scaling for API services
- Worker pools for document processing
- Auto-scaling based on usage patterns

7.3 Resilience

- Circuit breakers for external dependencies
- Retry mechanisms with exponential backoff
- Fallback strategies for AI model failures

A4. Security

- End-to-end encryption for sensitive data
- Role-based access control
- API key rotation and management

8. Future Enhancements

8.1 Doc Agent Enhancements

- Multi-modal document understanding (images, diagrams)
- Interactive report customization
- Collaborative editing capabilities

8.2 Support Agent Expansion

- Predictive issue detection
- Automated regression test generation
- Self-healing system recommendations

8.3 A2A Communication Improvements

- Enhanced reasoning capabilities between agents
- Learning from cross-agent interactions
- Dynamic agent specialization

8.4 Infrastructure Evolution

- Edge deployment for latency-sensitive operations
- Hybrid on-prem/cloud deployment options
- Green computing optimizations

9. System Architecture Diagram

[See attached system diagram in original document]

10. Implementation Roadmap

Phase 1: Doc Agent Core

- Implement basic Doc Agent with OpenAI Assistant
- Develop manual upload and RAG system
- Create Change History and Feature Evolution reports

Phase 2: Doc Agent Enhancements

- Add Confluence and SharePoint integration via MCP
- Implement A2A communication between product sub-agents
- Develop feedback system for continuous improvement

Phase 3: System Evolution

- Migrate to Llama4 LLM and SLMs
- Enhance A2A with advanced reasoning
- Implement full RLHF pipeline

Phase 4: Support Agent Foundation

- Establish JIRA integration through MCP
- Develop Log Analyzer sub-agent
- Create basic defect analysis capabilities

Phase 5: Support Agent Advanced Features

- Implement fix planning with impact analysis
- Develop automated fix implementation
- Create comprehensive testing framework