Product Requirements Document (PRD)

Sistema RAG para Documentação Técnica

---

1. Executive Summary

1.1 Product Overview

O **DocRag** é um sistema de Retrieval-Augmented Generation (RAG) que permite desenvolvedores consultar múltiplas documentações técnicas através de linguagem natural e receber respostas contextualizadas com geração de código.

1.2 Problem Statement

- **Desenvolvedores gastam 35% do tempo** procurando informações em documentações dispersas
- **Context switching** entre múltiplas fontes reduz produtividade
- **Documentações técnicas** são extensas e difíceis de navegar
- **Informações desatualizadas** causam bugs e retrabalho

1.3 Solution Overview

Sistema RAG que indexa documentações oficiais (React, Python, AWS) e permite consultas inteligentes com:

- Respostas contextualizadas de múltiplas fontes
- Geração de código funcional
- Comparação entre diferentes abordagens
- Rastreamento de versões da documentação

1.4 Success Metrics

- **Time-to-Answer**: <10 segundos para 90% das consultas
- **Accuracy**: >85% de respostas consideradas úteis pelos usuários
- **Code Quality**: 80% do código gerado executa sem erros
- **User Satisfaction**: 4.0+ stars em feedback

---

2. Product Goals & Success Criteria

2.1 Primary Goals

1. **Reduzir tempo de busca** em documentações de 15min para 30s
2. **Aumentar precisão** das informações encontradas
3. **Gerar código funcional** baseado em documentação
4. **Comparar abordagens** de diferentes tecnologias

2.2 Success Criteria

- [ ] Processar 3+ documentações técnicas principais
- [ ] Responder queries complexas em <10s
- [ ] Gerar código executável em 80% dos casos
- [ ] Interface web funcional e responsiva
- [ ] Deploy em produção com 99% uptime

2.3 Out of Scope (V1)

- Integração com IDEs
- Autenticação de usuários
- Personalização de documentações
- API pública
- Multi-linguagem (apenas inglês)

---

3. User Personas & Use Cases

3.1 Primary Persona: **Alex - Full-Stack Developer**

Demographics: 3-5 anos experiência, trabalha com múltiplas tecnologias **Pain Points**:

- Perde tempo alternando entre docs do React, Python, FastAPI
- Dificuldade em encontrar exemplos práticos
- Informações contraditórias entre fontes

Use Cases:

- "Como fazer autenticação JWT em FastAPI?"
- "Diferenças entre React hooks e class components?"
- "FastAPI vs Django para APIs REST?"

3.2 Secondary Persona: **Jordan - Junior Developer**

Demographics: 0-2 anos experiência, aprendendo tecnologias **Pain Points**:

- Documentações muito técnicas
- Não sabe fazer as perguntas certas
- Precisa de exemplos step-by-step

Use Cases:

- "Como criar meu primeiro component React?"
- "Exemplo básico de API com Python?"
- "Explicar conceitos como se eu fosse iniciante"

---

4. Functional Requirements

4.1 Core Features

4.1.1 Document Ingestion & Processing

Priority: P0 (Must Have)

- [ ] **FR-001**: Ingerir documentação oficial do React (docs.reactjs.org)
- [ ] **FR-002**: Ingerir documentação oficial do Python (docs.python.org)
- [ ] **FR-003**: Ingerir documentação oficial do FastAPI (fastapi.tiangolo.com)
- [ ] **FR-004**: Processar markdown, HTML, e texto estruturado
- [ ] **FR-005**: Chunking inteligente preservando contexto de código
- [ ] **FR-006**: Detecção automática de versão da documentação

4.1.2 Query Processing & Retrieval

Priority: P0 (Must Have)

- [ ] **FR-007**: Interface de chat para queries em linguagem natural
- [ ] **FR-008**: Busca semântica usando embeddings
- [ ] **FR-009**: Ranking de relevância de chunks recuperados
- [ ] **FR-010**: Suporte a queries multi-turn (conversação)
- [ ] **FR-011**: Histórico de conversas na sessão

4.1.3 Response Generation

Priority: P0 (Must Have)

- [ ] **FR-012**: Geração de resposta contextualizada via LLM
- [ ] **FR-013**: Citação de fontes com links diretos
- [ ] **FR-014**: Geração de código funcional quando aplicável

- [ ] **FR-015**: Explicação step-by-step para implementações
- [ ] **FR-016**: Sugestões de queries relacionadas

4.1.4 Multi-Source Comparison

Priority: P1 (Should Have)

- [ ] **FR-017**: Comparar abordagens entre tecnologias
- [ ] **FR-018**: Mostrar pros/cons de diferentes soluções
- [ ] **FR-019**: Timeline/versioning de features

4.2 Advanced Features

4.2.1 Code Generation

Priority: P1 (Should Have)

- [ ] **FR-020**: Gerar código React funcional
- [ ] **FR-021**: Gerar código Python executável
- [ ] **FR-022**: Gerar código FastAPI para APIs REST
- [ ] **FR-023**: Syntax highlighting no código gerado
- [ ] **FR-024**: Copy-to-clipboard para código

4.2.2 Quality & Validation

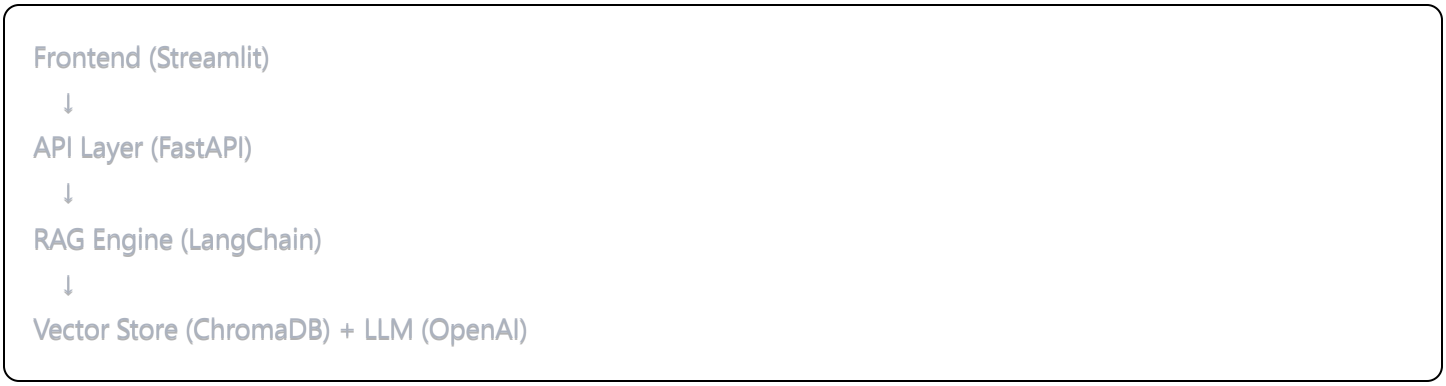
Priority: P2 (Could Have)

- [ ] **FR-025**: Validação de sintaxe do código gerado
- [ ] **FR-026**: Links para documentação original
- [ ] **FR-027**: Feedback thumbs up/down para respostas
- [ ] **FR-028**: Métricas de usage e performance

---

5. Technical Architecture

5.1 System Architecture



5.2 Tech Stack

5.2.1 Backend

- **Language**: Python 3.10+
- **Framework**: FastAPI 0.104+
- **RAG**: LangChain 0.1.0+
- **Vector DB**: ChromaDB 0.4.0+
- **LLM**: OpenAI GPT-4 Turbo

5.2.2 Frontend

- **Framework**: Streamlit 1.28+
- **UI Components**: Native Streamlit widgets
- **Styling**: Custom CSS for branding

5.2.3 Infrastructure

- **Containerization**: Docker + docker-compose
- **Database**: SQLite (development), PostgreSQL (production free tier)
- **Deployment**: Streamlit Cloud (frontend), Railway/Render (API)
- **Monitoring**: Python logging + basic health checks

5.3 Data Flow

5.3.1 Document Processing Pipeline

1. **Scrape/Download** → Raw documentation files
2. **Parse & Clean** → Structured text + metadata
3. **Chunk & Embed** → Vector representations
4. **Index & Store** → ChromaDB + metadata in PostgreSQL

5.3.2 Query Processing Pipeline

1. **User Query** → Natural language input
2. **Embedding** → Vector representation
3. **Retrieval** → Top-K similar chunks
4. **Augmentation** → Context + query to LLM
5. **Generation** → Structured response

6. **Post-processing** → Citations + formatting

---

6. User Experience & Interface Design

6.1 Core User Flow

1. **Landing Page** → Brief explanation + "Try Demo"
2. **Chat Interface** → Text input + message history
3. **Response Display** → Answer + citations + code blocks
4. **Follow-up** → Related questions + new query

6.2 Key UI Components

6.2.1 Chat Interface

6.2.2 Response Format

Sources

- React Docs - Components and Props

- Python Docs - Functions
- AWS Docs - Lambda Functions

🔍 Related Questions

- How to test this function?
- Performance best practices?

6.3 Responsive Design
- **Desktop**: Full-width chat interface
- **Mobile**: Collapsed sidebar, vertical layout
- **Accessibility**: Keyboard navigation, screen reader support

---

7. Technical Specifications

7.1 Performance Requirements
- **Response Time**: <10s for 95% of queries
- **Concurrent Users**: Support 50+ simultaneous users
- **Throughput**: 100+ queries per minute
- **Availability**: 99.5% uptime (MVP)

7.2 Data Requirements
- **Document Storage**: ~500MB of processed documentation
- **Vector Storage**: ~1GB for embeddings
- **Query History**: 1000 queries retained per session
- **Metadata**: Version tracking, timestamps, sources

7.3 Security & Privacy
- **API Keys**: Environment variables, not in code
- **Data Encryption**: HTTPS for all communications
- **Rate Limiting**: 10 queries per minute per IP
- **Privacy**: No persistent user data storage

7.4 Scalability Considerations
- **Horizontal Scaling**: Stateless API design
- **Caching**: Redis for frequent queries (future)
- **Load Balancing**: Multiple FastAPI instances
- **Database**: Connection pooling, query optimization

---

8. Implementation Plan

8.1 Development Phases

Phase 1: MVP Core (4 weeks)
Week 1-2: Foundation
- [ ] Project setup + environment

- [ ] Basic FastAPI + Streamlit structure
- [ ] Document ingestion pipeline
- [ ] ChromaDB setup + embedding

Week 3-4: RAG Implementation
- [ ] LangChain integration
- [ ] Query processing logic
- [ ] Response generation
- [ ] Basic UI implementation

Phase 2: Enhancement (2 weeks)
Week 5: Advanced Features
- [ ] Multi-source comparison
- [ ] Code generation improvement
- [ ] Citation system
- [ ] Error handling

Week 6: Polish & Deploy
- [ ] UI/UX improvements
- [ ] Performance optimization
- [ ] Docker containerization
- [ ] Production deployment

8.2 Development Milestones

Milestone 1: Document Processing ✅
- [ ] Successfully ingest React, Python, AWS docs
- [ ] Generate embeddings for all chunks
- [ ] Basic retrieval working

Milestone 2: RAG Pipeline ✅
- [ ] End-to-end query processing
- [ ] LLM integration functional
- [ ] Contextual responses generated

Milestone 3: Web Interface ✅
- [ ] Streamlit app deployed
- [ ] Chat interface working
- [ ] Basic styling applied

Milestone 4: Production Ready ✅
- [ ] Docker deployment
- [ ] Error handling robust
- [ ] Performance acceptable

- [ ] Documentation complete

---

9. Testing Strategy

9.1 Testing Types

9.1.1 Unit Tests
- **Document Processing**: Parsing, chunking, embedding
- **RAG Components**: Retrieval accuracy, generation quality
- **API Endpoints**: Request/response validation
- **Utilities**: Helper functions, data processing

9.1.2 Integration Tests
- **End-to-End Flow**: Query → Response pipeline
- **Database Operations**: CRUD operations
- **External APIs**: OpenAI API integration
- **Vector Search**: ChromaDB queries

9.1.3 Performance Tests
- **Load Testing**: 50+ concurrent users
- **Response Time**: <10s requirement validation
- **Memory Usage**: Acceptable resource consumption
- **Scalability**: Performance under increased load

9.2 Test Data & Scenarios

9.2.1 Query Test Cases
```python
test_queries = [
    "How to create a React component?",
    "Python list comprehension examples",
    "AWS Lambda deployment steps",
    "Compare React vs Vue components",
    "FastAPI authentication tutorial",
    "Error handling in Python functions"
]
```

9.2.2 Expected Outcomes

- **Accuracy**: >85% relevant responses
- **Code Quality**: Syntactically correct

- **Citations**: Proper source attribution
- **Performance**: Within time limits

---

10. Success Metrics & KPIs

10.1 Product Metrics

- **User Engagement**: Average session duration >5 minutes
- **Query Success Rate**: >85% queries receive useful responses
- **Code Generation Success**: >80% generated code is syntactically correct
- **Response Accuracy**: >90% responses include relevant citations

10.2 Technical Metrics

- **System Performance**: <10s response time for 95% queries
- **Uptime**: >99.5% availability
- **Error Rate**: <2% of queries result in errors
- **Resource Usage**: <2GB RAM under normal load

10.3 Business Metrics

- **Portfolio Impact**: Feature in top 3 GitHub repositories
- **Demo Engagement**: >100 unique users try the system
- **Interview Mentions**: Used in 80% of job interviews
- **GitHub Stars**: >50 stars on repository

---

11. Risks & Mitigation

11.1 Technical Risks

Risk 1: OpenAI API Costs

Impact: High usage could exceed budget **Mitigation**:

- Implement rate limiting
- Cache frequent queries
- Monitor usage dashboards
- Set spending limits

Risk 2: Vector DB Performance

Impact: Slow retrieval affects user experience **Mitigation**:

- Optimize chunk size and overlap
- Implement query result caching
- Monitor retrieval latency
- Consider alternative vector DBs

Risk 3: Documentation Changes

Impact: Outdated information in responses **Mitigation**:

- Version detection in pipeline
- Automated doc refresh workflow
- Timestamp tracking
- Update notifications

11.2 Product Risks

Risk 1: Poor Response Quality

Impact: Users abandon the system **Mitigation**:

- Extensive testing with real queries
- Feedback collection system
- Prompt engineering optimization
- Human evaluation of responses

Risk 2: Limited Documentation Coverage

Impact: Can't answer many user queries **Mitigation**:

- Start with most popular sections
- Gradual expansion of coverage
- Clear scope communication
- Suggestion system for missing topics

---

12. Go-to-Market Strategy

12.1 Target Audience

- **Primary**: Full-stack developers (3-7 years experience)
- **Secondary**: Junior developers learning new technologies
- **Tertiary**: Technical recruiters evaluating candidates

12.2 Distribution Channels

- **GitHub Repository**: Open source with detailed README
- **Live Demo**: Deployed on Railway/Streamlit Cloud
- **LinkedIn Posts**: Development progress updates
- **Dev.to Articles**: Technical implementation deep-dive
- **Portfolio Website**: Featured as primary project

12.3 Messaging

- **Value Prop**: "Stop context switching between docs. Get answers with code examples in seconds."
- **Differentiator**: "Multi-source comparison with version awareness"
- **Demo Hook**: "Ask me: 'How to build a React component that calls a Python API on AWS?'"

---

13. Post-Launch Plan

13.1 Immediate Post-Launch (Week 1-2)

- [ ] Monitor system performance and user feedback
- [ ] Fix critical bugs and performance issues
- [ ] Collect analytics on query patterns
- [ ] Create demo video for portfolio

13.2 Short-term Improvements (Month 1-2)

- [ ] Add more documentation sources (Vue, Django, etc.)
- [ ] Implement query suggestions and autocomplete
- [ ] Improve code generation accuracy
- [ ] Add export functionality (PDF, markdown)

13.3 Long-term Vision (Month 3+)

- [ ] IDE plugin development
- [ ] Enterprise version with private docs
- [ ] API monetization strategy

- [ ] Team collaboration features

---

14. Appendices

14.1 API Documentation Structure

```
GET  /health            - System health check
POST /query             - Process user query
GET  /sources           - List available documentation sources
GET  /history           - Get query history (session-based)
POST /feedback          - Submit response feedback
```

14.2 Environment Variables

```
OPENAI_API_KEY=sk-...
DATABASE_URL=postgresql://...
CHROMA_DB_PATH=./chroma_db
LOG_LEVEL=INFO
PORT=8000
```

14.3 Deployment Configuration

dockerfile

```
FROM python:3.10-slim
WORKDIR /app
COPY requirements.txt .
RUN pip install -r requirements.txt
COPY . .
EXPOSE 8000
CMD ["uvicorn", "main:app", "--host", "0.0.0.0", "--port", "8000"]
```

Document Version: 1.0

Last Updated: Current Date

Next Review: After MVP completion

Owner: Technical Lead

Stakeholders: Self (Portfolio Development)

This PRD serves as the single source of truth for the DocuRAG MVP development and should be updated as requirements evolve during implementation.