**Business Requirements Document**

**Gemini AI Analysis Service**

**Document Information**

* **Document Title**: Business Requirements Document - Gemini AI Analysis Service
* **Version**: 1.0
* **Date**: July 24, 2025
* **Author**: Technical Team
* **File**: backend/app/services/ai/gemini.py

**1. Executive Summary**

**1.1 Purpose**

This document outlines the business requirements for the Gemini AI Analysis Service, which provides automated code analysis and repository evaluation capabilities using Google's Gemini AI model. The service enables intelligent code review, architectural insights, and project assessment to support development teams and technical decision-making.

**1.2 Scope**

The Gemini AI Analysis Service encompasses two primary functionalities:

* **Code Analysis**: Deep analysis of individual Python files for quality assessment, architectural patterns, and improvement recommendations
* **Repository Analysis**: Comprehensive evaluation of GitHub repositories for technical stack assessment, project insights, and development recommendations

**2. Business Objectives**

**2.1 Primary Objectives**

* **Automated Code Review**: Reduce manual code review time by 60-70% through AI-powered analysis
* **Quality Assurance**: Improve code quality by identifying patterns, issues, and architectural decisions
* **Technical Decision Support**: Provide data-driven insights for technology stack and architectural choices
* **Developer Productivity**: Enable faster onboarding and understanding of codebases

**2.2 Success Metrics**

* Response time < 30 seconds for code analysis
* Analysis accuracy rate > 85%
* User satisfaction score > 4.0/5.0
* Reduction in code review cycles by 40%

**3. Stakeholder Analysis**

**3.1 Primary Stakeholders**

* **Development Teams**: Primary users requiring code analysis and insights
* **Technical Leads**: Users needing repository and architecture assessment
* **Project Managers**: Consumers of project complexity and development stage insights
* **DevOps Teams**: Users requiring technology stack and scalability assessments

**3.2 Secondary Stakeholders**

* **Quality Assurance Teams**: Beneficiaries of code quality insights
* **Product Owners**: Consumers of project development stage information
* **Engineering Management**: Users of team productivity metrics

**4. Functional Requirements**

**4.1 Code Analysis Service (FR-001)**

**4.1.1 Input Requirements**

* **Code Content**: Accept Python code as string input
* **File Context**: Support analysis of individual file content
* **Format Support**: Handle various Python code structures and patterns

**4.1.2 Analysis Capabilities**

* **Summary Generation**: Provide 2-3 sentence file purpose description
* **Function Analysis**: Extract function names, purposes, parameters, and return types
* **Class Analysis**: Identify classes, purposes, methods, and inheritance relationships
* **Import Categorization**: Classify imports as standard library, third-party, or local
* **Quality Assessment**: Identify design patterns, potential issues, and architectural decisions

**4.1.3 Output Requirements**

* **Structured JSON Response**: Return standardized JSON format
* **Comprehensive Coverage**: Include all analysis categories
* **Error Handling**: Provide meaningful error messages for failed analyses

**4.2 Repository Analysis Service (FR-002)**

**4.2.1 Input Requirements**

* **Repository Metadata**: Accept GitHub repository metadata as JSON
* **Multi-language Support**: Handle repositories with multiple programming languages
* **Metadata Completeness**: Work with varying levels of available metadata

**4.2.2 Analysis Capabilities**

* **Project Summary**: Generate purpose and value proposition description
* **Architecture Assessment**: Infer architecture patterns and tech stack decisions
* **Technology Evaluation**: Assess language choices and dependencies
* **Project Insights**: Evaluate complexity, target users, and development stage

**4.2.3 Output Requirements**

* **Structured Analysis**: Return comprehensive JSON-formatted insights
* **Scalability Assessment**: Include scalability considerations
* **Maintenance Insights**: Provide maintenance and development stage information

**5. Non-Functional Requirements**

**5.1 Performance Requirements (NFR-001)**

* **Response Time**: API calls must complete within 30 seconds
* **Throughput**: Support minimum 100 concurrent analysis requests
* **Availability**: Maintain 99.5% uptime during business hours

**5.2 Reliability Requirements (NFR-002)**

* **Error Handling**: Graceful degradation for API failures
* **Retry Logic**: Implement exponential backoff for failed requests
* **Fallback Responses**: Provide meaningful fallback when AI analysis fails

**5.3 Security Requirements (NFR-003)**

* **API Key Management**: Secure storage and handling of Gemini API keys
* **Input Validation**: Sanitize and validate all input content
* **Logging Security**: Avoid logging sensitive code content

**5.4 Scalability Requirements (NFR-004)**

* **Horizontal Scaling**: Support scaling across multiple instances
* **Resource Management**: Efficient memory and CPU utilization
* **Rate Limiting**: Implement appropriate rate limiting for API calls

**6. Technical Requirements**

**6.1 Integration Requirements**

* **Google Gemini API**: Integration with gemini-1.5-flash-latest model
* **Configuration Management**: Support for environment-based configuration
* **Async Processing**: Asynchronous processing capabilities

**6.2 Data Requirements**

* **Input Format**: Support for string-based code content and JSON metadata
* **Output Format**: Standardized JSON response schema
* **Error Reporting**: Comprehensive error logging and reporting

**6.3 Dependencies**

* **Core Libraries**: google.generativeai, httpx, json, logging
* **Configuration**: Integration with application configuration system
* **Python Version**: Compatible with Python 3.8+

**7. Business Rules**

**7.1 Analysis Rules (BR-001)**

* All code analysis must follow the standardized JSON schema
* Repository analysis must infer insights from available metadata only
* Failed analyses must return structured error responses

**7.2 Quality Rules (BR-002)**

* Analysis responses must be professional and technical in nature
* Recommendations must be actionable and specific
* Error messages must be user-friendly and informative

**7.3 Security Rules (BR-003)**

* API keys must be configured through environment variables
* Code content must not be logged or persisted
* All API communications must use secure protocols

**8. Constraints and Assumptions**

**8.1 Technical Constraints**

* **API Dependency**: Service functionality depends on Google Gemini API availability
* **Language Limitation**: Code analysis optimized for Python language
* **Model Limitation**: Analysis quality limited by Gemini model capabilities

**8.2 Business Constraints**

* **Cost Considerations**: API usage costs must be monitored and controlled
* **Rate Limits**: Subject to Google Gemini API rate limiting
* **Model Updates**: Service behavior may change with model updates

**8.3 Assumptions**

* Users have appropriate access permissions for analyzed repositories
* Input code is syntactically valid Python code
* Repository metadata is accurate and up-to-date

**9. Risk Assessment**

**9.1 Technical Risks**

* **API Unavailability**: Gemini API service interruptions
* **Rate Limiting**: Exceeding API rate limits during peak usage
* **Model Changes**: Updates to Gemini model affecting analysis quality

**9.2 Business Risks**

* **Cost Overruns**: Unexpected API usage costs
* **Analysis Accuracy**: Incorrect or misleading analysis results
* **User Adoption**: Low adoption due to analysis quality concerns

**9.3 Mitigation Strategies**

* Implement robust error handling and fallback mechanisms
* Monitor API usage and implement cost controls
* Establish feedback loops for analysis quality improvement
* Provide comprehensive documentation and training

**10. Success Criteria**

**10.1 Acceptance Criteria**

* All functional requirements implemented and tested
* Performance requirements met under expected load
* Error handling provides meaningful feedback
* JSON response schemas validated and documented

**10.2 User Acceptance**

* Development teams report improved code review efficiency
* Analysis results deemed accurate and helpful by technical leads
* Integration successful with existing development workflows

**11. Implementation Timeline**

**11.1 Phase 1: Core Development (Complete)**

* ✅ Basic Gemini API integration
* ✅ Code analysis functionality
* ✅ Repository analysis functionality
* ✅ Error handling implementation

**11.2 Phase 2: Enhancement (Future)**

* Performance optimization
* Extended language support
* Advanced analysis features
* User interface integration

**12. Appendices**

**12.1 JSON Schema Definitions**

Detailed JSON schemas for both code analysis and repository analysis responses are defined in the implementation and should be documented separately for API consumers.

**12.2 API Documentation**

Comprehensive API documentation should be maintained separately, including request/response examples and error code definitions.

**Document Control**

* **Review Required**: Technical Lead, Product Owner
* **Approval Required**: Engineering Manager
* **Next Review Date**: August 24, 2025