Data Model Suggestions

Generated by Requirements Gathering Agent v2.1.2
Category: technical-analysis
Generated: 2025-06-10T08:17:45.144Z
Description: Database architecture and data model recommendations

---

Data Model for Requirements Gathering Agent

This data model focuses on storing project metadata, discovered documents, AI interaction results, and generated PMBOK documents. Given the nature of the application, a relational database (SQL) is recommended for its data integrity features and mature tooling. A hybrid approach could be considered in the future if specific document content requires NoSQL capabilities (e.g., storing large unstructured text efficiently).

Database Technology Recommendation: PostgreSQL (for its scalability, extensibility, and JSON support) or MySQL (for simpler setup and wide community support).

Normalization Level: 3NF (Third Normal Form) will be targeted for most entities to minimize redundancy and improve data integrity.

Core Entities and Relationships:

```
Project (1:1)-----> ProjectMetadata
Project (1:N)-----> Document
Project (1:N)-----> AIProviderConfiguration
Document (1:N)-----> DocumentContent
Document (1:N)-----> DocumentAnalysis
Document (1:N)-----> DocumentValidationResult
AIProviderConfiguration (1:N)-----> AIProviderCredentials
```

Entity Details:

- **Project:**
  - `projectId` (INT, PRIMARY KEY, AUTO_INCREMENT)
  - `projectName` (VARCHAR(255))
  - `projectPath` (VARCHAR(255)) // Local path to project directory
  - `createdAt` (TIMESTAMP)
  - `updatedAt` (TIMESTAMP)
- **ProjectMetadata:**
  - `projectId` (INT, PRIMARY KEY, FOREIGN KEY referencing Project.projectId)
  - `readmeContent` (TEXT) // Store the README.md content
  - `packageJson` (JSONB) // Store parsed package.json data
  - `otherMetadata` (JSONB) // Other relevant project metadata

- **Document:**
  - `documentId` (INT, PRIMARY KEY, AUTO_INCREMENT)
  - `projectId` (INT, FOREIGN KEY referencing Project.projectId)
  - `documentType` (VARCHAR(255), e.g., 'ProjectCharter', 'StakeholderRegister')
  - `documentName` (VARCHAR(255))
  - `filePath` (VARCHAR(255)) // Path within the generated documents directory
  - `relevanceScore` (INT) // Score from the relevance engine (0-100)
  - `documentCategory` (VARCHAR(255), e.g., 'Planning', 'Development')
  - `createdAt` (TIMESTAMP)
  - `updatedAt` (TIMESTAMP)
- **DocumentContent:**
  - `contentId` (INT, PRIMARY KEY, AUTO_INCREMENT)
  - `documentId` (INT, FOREIGN KEY referencing Document.documentId)
  - `content` (TEXT) // Stores the actual document content
  - `version` (INT) // Version control for content changes
- **DocumentAnalysis:**
  - `analysisId` (INT, PRIMARY KEY, AUTO_INCREMENT)
  - `documentId` (INT, FOREIGN KEY referencing Document.documentId)
  - `analysisResult` (JSONB) // Results of content analysis (e.g., keywords, sentiment)
  - `analysisTimestamp` (TIMESTAMP)
- **DocumentValidationResult:**
  - `validationId` (INT, PRIMARY KEY, AUTO_INCREMENT)
  - `documentId` (INT, FOREIGN KEY referencing Document.documentId)
  - `validationStatus` (VARCHAR(255), e.g., 'Passed', 'Failed')
  - `validationReport` (JSONB) // Detailed validation report with recommendations
  - `validationTimestamp` (TIMESTAMP)
- **AIProviderConfiguration:**
  - `configId` (INT, PRIMARY KEY, AUTO_INCREMENT)
  - `projectId` (INT, FOREIGN KEY referencing Project.projectId)
  - `providerName` (VARCHAR(255), e.g., 'AzureOpenAI', 'GoogleAI')
  - `model` (VARCHAR(255))
  - `createdAt` (TIMESTAMP)
  - `updatedAt` (TIMESTAMP)
- **AIProviderCredentials:**
  - `credentialId` (INT, PRIMARY KEY, AUTO_INCREMENT)
  - `configId` (INT, FOREIGN KEY referencing AIProviderConfiguration.configId)
  - `credentialType` (VARCHAR(255), e.g., 'APIKey', 'AccessToken')
  - `credentialValue` (TEXT) // Securely stored credential value (consider encryption)

Indexing Recommendations:

- Index `projectId` in all related tables (Document, AIProviderConfiguration, etc.) for efficient querying.
- Index `documentType` in the `Document` table for filtering documents based on type.
- Index relevant fields in `DocumentAnalysis` and `DocumentValidationResult` for efficient retrieval of analysis and validation data.
- Consider a full-text index on `content` in `DocumentContent` for searching within document text (depending on database size and performance requirements).

Data Integrity Constraints:

- Ensure referential integrity between tables using foreign keys.
- Enforce data type constraints to maintain data quality.
- Use `NOT NULL` constraints where appropriate to prevent missing data.
- Implement unique constraints where necessary (e.g., unique project names).

Scalability and Performance Considerations:

- Use appropriate database indexing strategies.
- Optimize database queries for performance.
- Consider database sharding or replication for very large datasets.
- Employ connection pooling to improve efficiency.
- Regularly monitor database performance and tune as needed.

Data Security and Privacy Recommendations:

- Store sensitive data (API keys, access tokens) securely using encryption at rest and in transit.
- Implement access control mechanisms to restrict access to sensitive data.
- Regularly audit database security and update security measures as needed.
- Comply with relevant data privacy regulations (GDPR, CCPA, etc.).

This detailed data model provides a solid foundation for the Requirements Gathering Agent. Remember to adapt and extend it as the project evolves and new requirements emerge. Regular performance monitoring and database tuning will be crucial for maintaining optimal performance as the application scales.