**Business Requirement Document: DokyDoc Validation Service**

**Version:** 1.0 **Date:** 2025-07-17 **Author:** Gemini AI (Generated based on existing source code)

**1. Introduction**

**1.1. Project Overview**

The DokyDoc platform is designed to maintain a clear and reliable link between technical documentation and its corresponding source code. A critical challenge in software development is ensuring that documentation remains synchronized with the code as it evolves. The **Validation Service** is a core backend component of the DokyDoc system, created to automate the detection of inconsistencies between linked documents and code components. Its primary function is to identify and log these discrepancies, enabling development teams to maintain high-quality, accurate documentation.

**1.2. Business Objectives**

The primary business goals of the Validation Service are:

* **Improve Documentation Accuracy:** Automatically detect and flag outdated or inconsistent documentation, reducing the risk of errors caused by misinformation.
* **Increase Developer Productivity:** Minimize the time developers spend verifying documentation against the codebase.
* **Enhance System Reliability:** Ensure that technical specifications and operational guides accurately reflect the current state of the software.
* **Streamline Maintenance:** Provide a clear, actionable list of documentation issues that need to be addressed.

**1.3. Scope**

**In-Scope:**

* Automated detection of version mismatches between linked documents and code components.
* Logging all identified mismatches into a dedicated mismatches database table.
* Providing an API endpoint to trigger the validation scan on-demand.
* Providing an API endpoint to retrieve the list of all logged mismatches.

**Out-of-Scope:**

* Automated correction of mismatches.
* Analysis of code quality, style, or syntax.
* Validation of the content or prose within the documentation itself.
* Real-time (instantaneous) validation upon code or document changes.

**1.4. Target Audience**

* **Developers:** To verify that the documentation for the code they are working on is current.
* **Technical Writers:** To identify which documents require updates after a new software release.
* **Quality Assurance (QA) Engineers:** To ensure testing procedures and documentation are aligned with the software's actual behavior.
* **System Administrators / DevOps:** To confirm that deployment and operational guides are accurate.

**2. Functional and Non-Functional Requirements**

**2.1. Functional Requirements**

|  |  |  |
| --- | --- | --- |
| **ID** | **Requirement Name** | **Description** |
| **FR-001** | Version Mismatch Detection | The system must be able to compare the version attribute of a document with the version attribute of its linked code component. |
| **FR-002** | Mismatch Logging | If the versions from FR-001 do not match, the system must create a new record in the mismatches database table. |
| **FR-003** | Mismatch Data Persistence | Each mismatch record must store the ID of the document, the ID of the code component, the type of mismatch (e.g., "Version Mismatch"), and descriptive details. |
| **FR-004** | On-Demand Scan Execution | The system must expose an API endpoint that allows an authorized user to initiate a full validation scan across all linked documents and code components. |
| **FR-005** | Mismatch Data Retrieval | The system must expose an API endpoint that allows an authorized user to retrieve a list of all currently logged mismatches from the database. |

**2.2. Non-Functional Requirements**

|  |  |  |
| --- | --- | --- |
| **ID** | **Requirement Name** | **Description** |
| **NFR-001** | Performance | A validation scan of 1,000 linked items should complete in a reasonable timeframe (e.g., under 5 minutes). |
| **NFR-002** | Data Integrity | All mismatch records must accurately reflect the state of the system at the time of the scan. No false positives. |
| **NFR-003** | Scalability | The service must be able to handle a growing number of documents and code components without significant degradation. |
| **NFR-004** | Usability | The API endpoints for triggering scans and retrieving results must be well-documented, secure, and easy to use. |

**3. System Features & Functionality**

**3.1. Core Validation Logic (ValidationService)**

The service contains a primary method, run\_version\_mismatch\_check, which performs the core business logic. The process is as follows:

1. The service queries the database for all document\_code\_links.
2. It iterates through each link.
3. For each link, it compares the version field of the associated Document with the version field of the associated CodeComponent.
4. If the versions are different, it calls the create\_mismatch CRUD function to log the discrepancy.

**3.2. Mismatch Data Model (mismatch.py)**

When a mismatch is identified, a record is created with the following schema:

* id (Integer, Primary Key)
* document\_id (Integer, Foreign Key to documents table)
* code\_component\_id (Integer, Foreign Key to code\_components table)
* mismatch\_type (String): Describes the category of the error (e.g., "Version Mismatch").
* details (JSON/Text): A field for storing additional information about the mismatch.
* status (String): The current state of the mismatch (e.g., "new", "acknowledged", "resolved").
* created\_at (DateTime): Timestamp of when the mismatch was first detected.

**3.3. API Endpoints**

The Validation Service is controlled and accessed via the following RESTful API endpoints:

* **Trigger Validation Scan:**
  + **Endpoint:** POST /api/v1/validation/run
  + **Description:** Initiates a complete validation scan.
  + **Response:** 202 Accepted with a confirmation message.
* **Retrieve Mismatch Results:**
  + **Endpoint:** GET /api/v1/mismatches/
  + **Description:** Retrieves a list of all identified mismatches.
  + **Response:** 200 OK with a JSON array of mismatch objects.

**4. Assumptions and Dependencies**

* **Assumption:** Both Document and CodeComponent models have a version attribute that is consistently used and updated.
* **Assumption:** The relationship between documents and code is accurately maintained in the document\_code\_links table.
* **Dependency:** The service relies on the application's primary PostgreSQL database being available and the schema being up-to-date via Alembic migrations.
* **Dependency:** The service requires access to the document, code\_component, and document\_code\_link CRUD modules to fetch data for comparison.