# **Definition for TDD**

A Technical Design Document (TDD) is a detailed, structured document that outlines the technical aspects of a software project. It acts as a blueprint for the development team, providing a clear guide on how a system should be built, configured, and maintained. This document typically includes requirements, architecture, technical specifications, and design details, ensuring that every stakeholder and team member understands the system’s intended structure and functionality.

## **Key Elements of a Technical Design Document**

**Overview and Purpose:** Brief description of the project’s goals, objectives, and key functionalities. This section sets the context for the technical details to follow.

**System Architecture:** A high-level overview of the system architecture, including diagrams showing how different components interact. This may involve diagrams for both front-end and back-end components, database design, and how these pieces connect.

**Technical Requirements:** Lists the software and hardware requirements, including programming languages, frameworks, libraries, databases, third-party integrations, and APIs.

## **Detailed Design Specifications:**

**Component Design:** Descriptions of individual modules or components, outlining their purpose, responsibilities, and dependencies.

**Data Models**: Database schema or data structures, including tables, fields, and relationships for database-driven applications.

**APIs:** Specifications for API endpoints, request and response structures, authentication, and error handling.

**User Interface (UI) Components:** Details on UI structure and behavior if applicable.

**Workflow and Process Diagrams:** Sequence diagrams, flowcharts, or activity diagrams that visually represent how data flows and processes execute across the system.

**Coding Standards and Guidelines:** Defines coding practices, naming conventions, and formatting standards to maintain code consistency and readability.

**Error Handling & Logging:** Describes the error handling and logging mechanisms to ensure that the system captures and manages errors effectively.

**Security Considerations**: Explains the security measures in place, such as data encryption, access control, authentication, and authorization mechanisms.

**Testing and Validation:** Outlines the testing strategy, including unit tests, integration tests, and performance tests, to validate that the system meets functional and non-functional requirements.

**Deployment and Maintenance:** Details deployment procedures, configuration management, monitoring, and any maintenance tasks needed to keep the system running smoothly.

## **Benefits of a Technical Design Document**

**Clear Communication**: Ensures all team members and stakeholders have a common understanding of the technical approach.

**Reduced Misinterpretation:** Provides a point of reference to clarify design intentions and avoid misunderstandings.

**Facilitates Development and Maintenance:** Speeds up development by offering a clear roadmap and makes future maintenance or expansion easier due to documented details.

A well-crafted TDD helps to ensure alignment between development, design, and business goals, streamlining the overall project execution.

# **Technical Design Document (TDD) for Stomble Project**

## **1. Document Overview**

* **Purpose**: This TDD serves to outline the technical architecture, design specifications, and development requirements for the Stomble project, focusing on a scalable, high-performance system.
* **Scope**: This document covers the technical design for front-end, back-end, API, identity management, and data integration, as well as considerations for future AI/LLM integration, DevOps, and CI/CD pipelines.
* **Intended Audience**: Project Managers, Development Team, Stakeholders, and any other relevant team members.

## **2. Project Overview**

* **Product Name**: Stomble
* **Goal**: Develop an Angular-based web application with integrated SDKs, identity management, backend APIs, and an RAG (Retrieval-Augmented Generation) LLM service.
* **Key Components**:
  + Front-end (Angular application)
  + Identity and Access Management (IAM)
  + Stomble API using Java 21 Micronaut framework
  + RDBMS for structured data
  + AI Service and Time Series Forecasting Model
  + Cloud Storage for Data Handling
* **Common Languages**: JavaScript, Python, Java

## **3. System Architecture**

* **Front-End**: Angular application using SDK integrations (JavaScript, .NET) for client interactions.
* **Backend**:
  + **Identity IAM**: Java Spring authentication server for secure access control.
  + **API Layer**: Stomble API built with Java Micronaut for handling business logic.
  + **Database**: Relational Database Management System (RDBMS) with structured data storage.
* **AI Services**:
  + **LLM Integration (RAG)**: Langchain for implementing retrieval-augmented generation.
  + **Data Modeling and AI Services**: Supporting AI model and time series forecasting.
  + **Cloud Storage**: Centralized storage solution for data and model assets.
* **Client Database Compatibility**: Integration with MySQL, MongoDB, and PostgreSQL for various clients.

## **4. Technical Requirements**

* **Front-End (Angular App)**:
  + **Tech Stack**: Angular 14+, TypeScript, HTML, CSS.
  + **SDK Support**: Support for JavaScript and .NET SDKs.
  + **Components**: Authentication UI, Dashboard, Data Visualization, Form Components.
* **Identity IAM**:
  + **Tech Stack**: Java 21, Spring Boot, OAuth2.
  + **Features**: User authentication, access management, token management.
* **Stomble API**:
  + **Tech Stack**: Java 21, Micronaut, REST APIs.
  + **Endpoints**:
    - CRUD operations for user management and data storage.
    - AI model and forecasting service integrations.
* **RDBMS**:
  + **Database Engine**: PostgreSQL (Primary), with compatibility for MySQL and MongoDB clients.
  + **Schema Design**: Includes tables for user data, transaction logs, and historical forecasting data.
* **LLM Integration (RAG)**:
  + **Framework**: Langchain for document retrieval and generation.
  + **Data Sources**: Connect to cloud storage and RDBMS for data retrieval.
* **Cloud Storage**:
  + **Service Provider**: AWS S3 or equivalent.
  + **Data Types**: Model files, AI-generated insights, logs.

## **5. Data Flow and Communication**

* **Data Flow**:
  + Front-end calls the Stomble API for data and operations.
  + Stomble API interacts with IAM for authentication and authorization.
  + API calls to RDBMS for structured data storage and retrieval.
  + RAG-LLM service fetches data from cloud storage and interacts with Stomble Datamodel for AI-enhanced responses.
* **Communication Protocols**:
  + **REST API** for Front-end and API communication.
  + **HTTPS** for secure data transmission.
  + **OAuth2** for secure authentication and access management.

## **6. Use Cases and User Stories**

* **User Authentication**:
  + As a user, I want to securely log in to access the dashboard and personal data.
* **Data Retrieval**:
  + As a client, I want to retrieve my historical data in real-time for analysis.
* **AI Forecasting**:
  + As a user, I want to access AI-powered forecasts to make informed business decisions.
* **Multi-Client Database Integration**:
  + As an admin, I want the system to support different databases for different clients.

## **7. Performance and Scalability**

* **API Rate Limiting**: Set limits for API requests per minute to prevent overloading.
* **Database Optimization**: Indexing frequently queried fields, partitioning large tables, caching.
* **Load Balancing**: Use load balancers to distribute traffic between API instances.
* **Horizontal Scaling**: Enable scaling for both front-end and backend servers.

## **8. Security Considerations**

* **Authentication**: Use OAuth2.0 standards for secure token-based authentication.
* **Data Encryption**: Encrypt sensitive data at rest and in transit.
* **Access Control**: Implement role-based access control (RBAC) within IAM.
* **API Security**: Use HTTPS for secure API calls and validate input data.

## **9. DevOps and CI/CD Pipeline**

* **Source Control**: GitHub for version control.
* **CI/CD Tools**: Jenkins or GitHub Actions for automated builds, tests, and deployments.
* **Deployment**: Docker for containerization, Kubernetes for orchestration.
* **Monitoring**: Implement monitoring using Prometheus and Grafana.
* **Logging**: Use ELK stack (Elasticsearch, Logstash, Kibana) for logging and analytics.

## **10. Testing Strategy**

* **Unit Testing**: For individual modules (e.g., front-end components, API endpoints).
* **Integration Testing**: Between front-end and backend, IAM and API, RAG-LLM, and API.
* **End-to-End Testing**: For core user flows using Cypress or Selenium.
* **Performance Testing**: Load testing for API endpoints using JMeter.
* **Security Testing**: Regular vulnerability scanning and penetration testing.

## **11. Milestones and Timeline**

* **Phase 1: Initial Setup and Front-End Development**
  + Duration: 2 weeks
  + Tasks: Set up Angular, initial SDK integrations, basic IAM.
* **Phase 2: Backend and API Development**
  + Duration: 4 weeks
  + Tasks: Implement IAM, Stomble API with RDBMS integration, CI/CD setup.
* **Phase 3: AI Integration and Data Handling**
  + Duration: 3 weeks
  + Tasks: Set up RAG-LLM, connect cloud storage, deploy AI service.
* **Phase 4: Testing and Security Enhancements**
  + Duration: 2 weeks
  + Tasks: Complete testing, security audits, optimization.
* **Phase 5: Deployment and Monitoring**
  + Duration: 1 week
  + Tasks: Final deployment, monitor performance, address post-launch issues.