Software System Design

[System Name]

Status: Draft

Date: 02 07 2025

Table of Contents

Introduction

This document outlines the system design for **[System Name]**. It details the architectural decisions, components, data models, and operational considerations for the project.

Overview

Provide a high-level summary of the system. What problem does it solve? Who are the intended users?

Goals and Objectives

List the primary goals of the system. These should be specific, measurable, achievable, relevant, and time-bound (SMART), if possible.

- **Goal 1:** To provide a scalable platform for...
- **Goal 2:** To achieve a 99.9% uptime...
- **Goal 3:** To reduce processing time by X%...

Non-Goals

Clearly state what is out of scope for this system or this version of the design. This helps manage expectations.

- Real-time collaboration features will not be included in v1.0.
- Support for third-party authentication providers other than Google is a non-goal.

Glossary

Define any terms, acronyms, or concepts that might be unfamiliar to the reader.

- **API:** Application Programming Interface
- **DB:** Database
- **SLA:** Service-Level Agreement

Requirements

This section details the functional and non-functional requirements that the system must satisfy.

Functional Requirements

Describe the specific behaviors and functions of the system. Use a numbered list for clarity.

1. **User Registration:** Users must be able to create an account using their email and password.
2. **Data Submission:** Authenticated users must be able to submit data through a web form.
3. **Admin Dashboard:** Administrators must have access to a dashboard to view all submitted data.

Non-Functional Requirements (NFRs)

Describe the quality attributes and constraints of the system.

Performance

- The system should respond to 95% of API requests within 200ms.
- The main dashboard should load in under 2 seconds.

Scalability

- The system must be able to handle 10,000 concurrent users.
- The storage solution should scale automatically to accommodate growing data volumes.

Availability

- The system must have an uptime of 99.9% (less than 8.76 hours of downtime per year).

- The system should be deployed across multiple availability zones to ensure redundancy.

Security
- All data in transit must be encrypted using TLS 1.2 or higher.
- User passwords must be hashed and salted using a strong algorithm (e.g., Argon2).
- The system must be protected against common web vulnerabilities (OWASP Top 10).

High-Level Architecture
This section provides a bird's-eye view of the system's architecture.

Architectural Diagram
Provide a diagram illustrating the main components and their interactions.

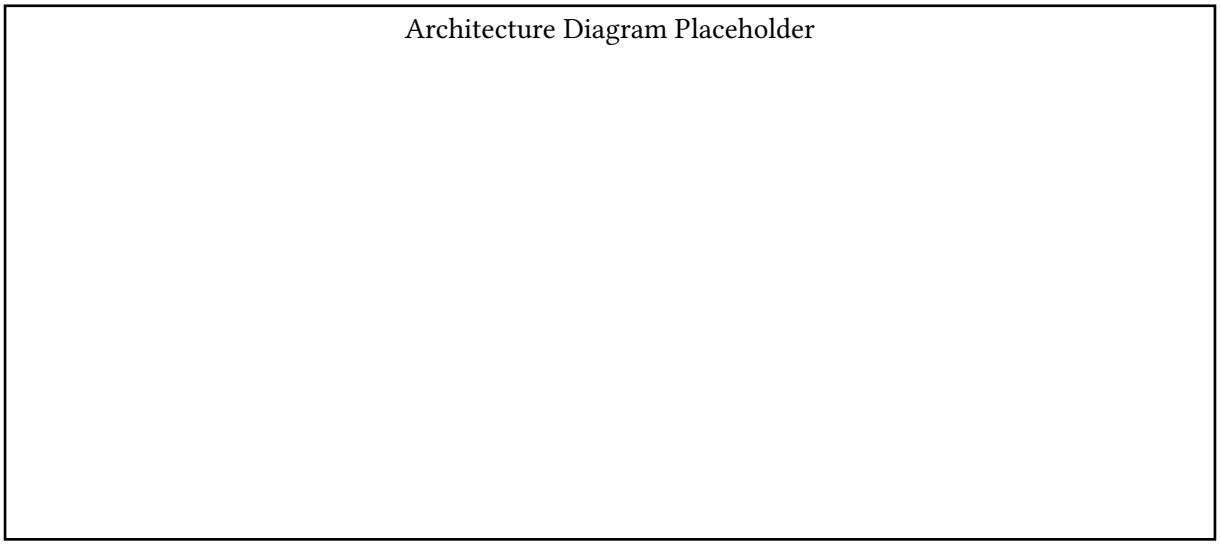

Figure 1: High-Level System Architecture.

System Components
Describe the responsibility of each major component shown in the diagram.

- **Web Client:** A single-page application (SPA) built with React that serves as the user interface.
- **API Gateway:** The single entry point for all client requests. Handles routing, authentication, and rate limiting.
- **Auth Service:** Manages user authentication and authorization.
- **Data Processing Service:** Handles the core business logic for processing submitted data.
- **Database:** A PostgreSQL database for persistent storage.

Technology Stack
List the key technologies, frameworks, and languages chosen for the project.

- **Frontend:** React, TypeScript
- **Backend:** Go, gRPC for inter-service communication
- **Database:** PostgreSQL
- **Infrastructure:** Docker, Kubernetes, AWS (EKS, S3, RDS)

Design Rationale & Trade-offs
Justify the architectural choices made. Why was a microservices architecture chosen over a monolith? Why PostgreSQL over a NoSQL database? Discuss the trade-offs.

- **Microservices vs. Monolith:** A microservices approach was chosen to allow for independent scaling and deployment of components, at the cost of increased operational complexity.

Data Model & Storage

This section describes how data is structured and stored.

Database Schema

Provide the database schema. You can use a code block for SQL DDL, a description of a NoSQL structure, or an Entity-Relationship Diagram (ERD).

```
-- Users Table
CREATE TABLE users (
    id UUID PRIMARY KEY,
    email VARCHAR(255) UNIQUE NOT NULL,
    password_hash VARCHAR(255) NOT NULL,
    created_at TIMESTAMP WITH TIME ZONE DEFAULT CURRENT_TIMESTAMP
);

-- Submissions Table
CREATE TABLE submissions (
    id UUID PRIMARY KEY,
    user_id UUID REFERENCES users(id),
    data JSONB NOT NULL,
    submitted_at TIMESTAMP WITH TIME ZONE DEFAULT CURRENT_TIMESTAMP
);
```

Data Flow

Describe how data moves through the system, from creation to storage and retrieval.

API Design

This section details the public-facing API endpoints.

REST API Endpoints

Describe the main API endpoints. A table is a good way to present this.

Method	Endpoint	Description
POST	/api/v1/users	Creates a new user account.
POST	/api/v1/auth/login	Authenticates a user and returns a JWT.
POST	/api/v1/submissions	Submits new data. (Requires Auth)
GET	/api/v1/submissions	Retrieves all submissions for the authenticated user. (Requires Auth)

Authentication & Authorization

- Authentication will be handled via JSON Web Tokens (JWTs).
- The `Authorization: Bearer <token>` header must be present for protected endpoints.

Deployment & Operations

CI/CD Pipeline

- A CI/CD pipeline will be set up using GitHub Actions.

- On every push to `main`, the pipeline will build Docker images, run tests, and deploy to the staging environment.
- Manual approval is required for production deployment.

Monitoring & Logging

- **Monitoring:** Prometheus will be used for metrics collection, with Grafana for dashboards.
- **Logging:** A centralized logging solution (e.g., ELK stack) will aggregate logs from all services.
- **Alerting:** PagerDuty will be configured for critical alerts.

Risks & Mitigation

Identify potential risks and outline mitigation strategies.

Risk	Mitigation Strategy
Database becomes a single point of failure.	Use a managed database service (e.g., AWS RDS) with multi-AZ replication and automated backups.
A security breach compromises user data.	Conduct regular security audits, implement strong encryption for data at rest and in transit, and follow the principle of least privilege.

AI Usage Disclosure

Gemini 2.5 Pro generated the initial `main.typ` file for the system design template.