Software Design and Architecture

Please remove help/template/example texts and tips from your submission (everything but headers).

*The goal of this document is to create a common understanding of the solution behind the system for various stakeholders. Does the chosen architecture fit the solution? Is the architecture appropriate? Note that the Product Requirements Document (PRD), the Software Development Process (SDP) document, and this document are somewhat complementary. Feel free to take some liberties about the structure. We want you to have a holistic view/understanding of the whole process and we provide these templates to help structure your thoughts. All companies do things a little differently.*

[Introduction](#_ltfxz43lvy0c)

[Architectural Goals and Principles](#_lf33tjen5nk3)

[System Overview](#_l9ten4jnpwl5)

[Architectural Patterns](#_71rcj0j51ki6)

[Component Descriptions](#_lrkv8tz6v3dj)

[Data Management](#_1xlbae4qa4n)

[Interface Design](#_yde8z4fio3au)

[Considerations](#_6vq934e8i85y)

[Security](#_6rwiw2gdazc9)

[Performance](#_dzluejn5gvsr)

[Maintenance and Support](#_8lc935dnx8we)

[Deployment Strategy](#_70pa7uh8tm26)

[Testing Strategy](#_hyz1vkpij8lw)

[Glossary](#_fj5bnerrh6yx)

# Introduction

*Briefly provide an overview of the project and its objectives. Explain the importance of the architecture in achieving project goals. What are the goals of the system? Your code doesn't tell the whole story.*

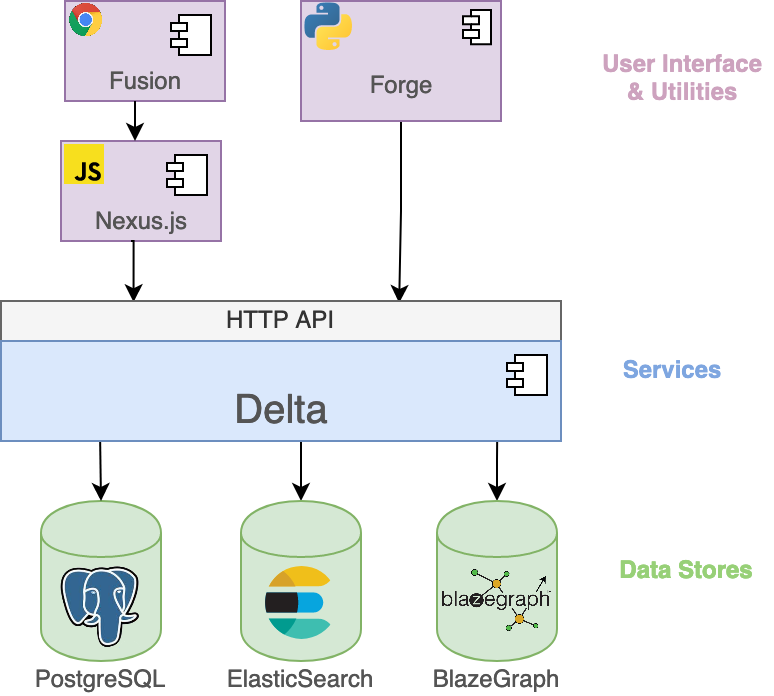
*For instance, "This document outlines the architecture for a web-based project management application. A well-defined architecture ensures scalability, maintainability, and reliability, it aligns with the company-mandated IT strategy, etc."*

# Architectural Goals and Principles

*Outline the key objectives the architecture aims to achieve, e.g., scalability, security, ease of integration with third-party services, etc. Specify architectural principles guiding the design, e.g., modularity, separation of concerns, minimal coupling, etc.*

# System Overview

*High-level description of the entire system, its main components, and the interaction between them (incl. technology, i.e., tech stack), e.g., user interface, backend server, database, third-party integrations. A diagram is a good way to represent this, e.g. this simplified overview of a more complex system:*

**

*Source:* [*Architecture · Blue Brain Nexus*](https://bluebrainnexus.io/docs/delta/architecture.html)

# Architectural Patterns

*Describe the chosen* [*architectural patterns*](https://www.redhat.com/architect/14-software-architecture-patterns) *(MVC, microservices, layered architecture, etc.) and the rationale for selecting these patterns based on project requirements. What are the architectural decisions and their arguments?*

*Discussion about microservices from Prime Video:* [*Scaling up the Prime Video audio/video monitoring service and reducing costs by 90% - Prime Video Tech*](https://www.primevideotech.com/video-streaming/scaling-up-the-prime-video-audio-video-monitoring-service-and-reducing-costs-by-90?utm_source=thenewstack&utm_medium=website&utm_content=inline-mention&utm_campaign=platform)

# Component Descriptions

*Explain in some detail each major component/module: their responsibilities and their role in the system. Here’s a simplified generic example:*

* *User Interface: Handles user interactions, displays data, and provides navigation.*
* *Backend Server: Processes requests, business logic, and interfaces with the database.*
* *Database: Stores and manages data using a relational database system.*

# Data Management

*How data is structured (e.g., relational database schema for user profiles, projects, and tasks), stored, and accessed (e.g., RESTful API endpoints for CRUD operations on user data) within the system. It’s very useful to come up with a database schema design and data flow diagrams.*

“Bad programmers worry about the code. Good programmers worry about data structures and their relationships.” ― **Linus Torvalds**

# Interface Definitions

*Complementary to the PRD’s user interaction and design section. Once you’ve established a first draft of what you need in your user interface, you need to think about what kind of interactions and data you need to support it. This includes APIs and communication protocols. You can provide API endpoints and methods, e.g., "GET /tasks returns a list of tasks, POST /tasks creates a new task." Later, you will probably have an API Reference documentation.*

# Considerations

## Security

*Identify potential security risks and measures to mitigate them. Define authentication, authorization, and data protection strategies. Some examples: SSL/TLS for data transmission, role-based access control for user data, etc.*

## Performance

*Discuss performance requirements and strategies to meet them, i.e., scalability, load balancing, and caching mechanisms.Some examples: horizontal scaling using container orchestration, local storage for the web application, etc.*

## Maintenance and Support

*Plan for future maintenance, updates, and support of the system. Part of the section is already addressed in the SDP document, i.e., versioning, issue tracking, etc. This section is concerned with who (person, team, organization) is going to take over your project, and what their skills, platform, and infrastructure look like. Another key aspect is how to collect feedback and provide support to users. This is complementary to the PRD’s section on constraints and dependencies.*

# Deployment Strategy

*This section is very similar to the one in the SDP document. You’ll complement it by explaining how the target architecture will be deployed in the various environments, and that includes choosing the right services, the right sizing (compute, memory, storage, … requirements), and resources on how to configure them. Later, you’ll probably have deployment documentation.*

# Testing Strategy

*This section is very similar to the one in the SDP document. You’ll complement it by explaining how the target architecture and their components will be tested in the various environments (incl. locally). This includes unit tests, integration tests, and larger tests.*

# Glossary

*Definitions of technical terms and acronyms used in the document, e.g., API: Application Programming Interface, CRUD: Create, Read, Update, Delete.*

***NOTE:***

*The final draft of your Software Design and Architecture document should contain the following ideas. Use this as a checklist:*

* *architecture is covered comprehensively and aligns with the stated goals and principles*
* *the chosen architectural patterns and decisions reflect the stated objectives*
* *the rationale behind the pattern selection is well-founded based on project requirements*
* *the system's major components and their interactions, including technology stack*
* *data schema, data flow, data storage, and access explanations*
* *identification of security risks and the appropriateness of security measures and strategies*
* *performance requirements and strategies to meet them, including scalability and caching mechanisms*
* *interface design, including user interfaces and APIs, along with example endpoints and methods*
* *component/module descriptions, including their responsibilities and roles in the system*
* *testing strategy, including unit tests, integration tests, and testing in various environments*
* *future maintenance, updates, and support are adequately addressed, including versioning, issue tracking, and user feedback mechanisms*
* *deployment strategy is explained, including choices of services, sizing, and configuration details*
* *optionally, strategies for scalability and performance optimization*