Solution Design & Architecture

1. Introduction

The Solution Design & Architecture phase is critical in ensuring that the application is built using the right technologies, organized components, and efficient data handling strategies. This document outlines the decisions made during this phase and describes how various elements will interact to deliver a robust, scalable, and maintainable application.

The goal is to create a detailed blueprint that guides the development team, aligns stakeholder expectations, and minimizes implementation risks.

Key objectives:

Select the best technology stack.

Define UI structure and API schema.

Outline data handling and security.

Present system architecture with diagrams.

Prepare for scalability and future enhancements.

2. Tech Stack Selection

Frontend

React.js for interactive UI.

Supports reusable components.

Fast rendering with virtual DOM.

Backend

Node.js (Express) for handling API requests.

Asynchronous processing for improved performance.

Database

PostgreSQL for structured and relational data.

Reliable and supports complex queries.

Hosting

AWS with EC2 instances and RDS.

Scalable and secure environment.

Tools

Docker for containerization.

GitHub Actions for CI/CD automation.

Why this stack? ✓ Easy to integrate

✓ Strong community support

✓ Scalable architecture

✓ Secure and compliant

3. UI Structure

Page Layout

1. Home Page – Overview of features.
2. Dashboard – Data visualization and reports.
3. Add/Edit Forms – Input for products, users, etc.
4. User Profile & Settings – Personal configurations.
5. Notifications – Feedback for user actions.

Design Principles

Minimalist layout.

Consistent color scheme.

Mobile-first responsive design.

Clear navigation paths.

Wireframe Example:

Header → Navigation Sidebar → Main Content → Footer

4. API Schema Design

REST API Endpoints

Endpoint	Method	Description
/api/products	GET	List all products
/api/products	POST	Create new product
/api/products/{id}	PUT	Update product

/api/products/{id}	DELETE	Remove product

Authentication

Login with email and password.

Token-based authentication using JWT.

Refresh token mechanism.

Response Format

```
{
  "success": true,
  "data": {
    "id": 1,
    "name": "Sample Product",
    "price": 199.99
  }
}
```

5. Data Handling Approach

Data Validation:

Both client and server side validation.

Ensures clean and correct input.

Caching:

Use Redis for frequently accessed data.

Improves response times.

Error Handling:

Log errors using Sentry or similar tools.

Display meaningful error messages.

Security:

Encrypt sensitive data like passwords using bcrypt.

Enforce HTTPS for secure transmission.

Backup:

Automated daily backups stored on cloud.

Disaster recovery procedures in place.

6. Component Diagram

[Client UI]

↓

[API Gateway]

↓

[Auth Module] ↔ [User Database]

↓

[Business Logic Layer] ↔ [Product Database]

↓

[External Services: Payment, Email]

Explanation:

UI interacts with API Gateway.

Authentication handled separately.

Business logic processes data requests.

External services enrich the system’s functionality.

7. Basic Flow Diagrams

User Registration

User → Registration Form → API → Validation → DB → Confirmation → UI Feedback

Data Fetching

User → API Request → DB Query → Format Data → Send Response → Display

Login

User → API → Validate Credentials → Issue JWT → Redirect → Dashboard

8. Security & Compliance

Role-based access control to limit permissions.

Encryption at rest and in transit.

Regular audits to identify vulnerabilities.

GDPR-compliant data handling policies.

9. Scalability & Performance

Load Balancing

Requests distributed across multiple servers.

Database Optimization

Index frequently used fields.

Archive old data for efficiency.

Monitoring Tools:

Prometheus for metrics collection.

Grafana for dashboards.

Auto-scaling:

Infrastructure adjusts based on traffic patterns.

10. Summary & Next Steps

This document provides a structured approach to implementing the solution, focusing on technology, design, and performance. The next phase will involve implementation planning, creating detailed development tasks, and setting up CI/CD pipelines.

Key Takeaways

✓ Technology chosen for performance and scalability.

✓ API schema designed for ease of integration.

✓ Data handling secured with validation and encryption.

✓ Architecture modular for future growth.