System Design Specification (SDS)

App Name

Team Members

## **1. Overview**

Provide a brief description of the application’s functionality.  
*(Example: “The application should allow users to \_\_\_\_\_\_\_ and manage \_\_\_\_\_\_\_. It must support \_\_\_\_\_\_\_.”)*

## **2. Functional Requirements**

List the primary features the application must support.

* **Example for a Chat Application:**
  + User Registration and authentication
  + Real-time messaging
  + Group creation and group chat
  + Multimedia support (e.g., photos, videos)
  + Message history and search functionality
* **Example for a URL Shortener:**
  + URL shortening for long URLs
  + Redirection from short URLs to original URLs
  + User analytics for tracking click rates and regions
  + URL expiration and link management features

*(Fill in the specific functional requirements for your application.)*

## **3. Non-Functional Requirements**

Specify performance and operational characteristics.

* **For Example:**
  + Scalability: Must handle \_\_\_\_\_\_\_ (e.g., millions of users/requests).
  + Availability: Ensure uptime of \_\_\_\_\_\_\_ (e.g., 99.9%).
  + Latency: Responses should occur in \_\_\_\_\_\_\_ (e.g., under 100ms).
  + Security: Implement \_\_\_\_\_\_\_ (e.g., data encryption, user privacy protection).

*(List the non-functional requirements specific to your application.)*

## **4. System Architecture**

Outline the high-level components and flow of data. Include frontend, backend, database, caching, and load balancer components if applicable.

* **Example Structure:**
  + **Frontend**: Interface for \_\_\_\_\_\_\_ (e.g., users to interact with the system).
  + **Backend**: API layer handling \_\_\_\_\_\_\_ (e.g., core functionalities).
  + **Database**: Storing \_\_\_\_\_\_\_ (e.g., user data, chat history, URL mappings).
  + **Cache Layer**: Improving response time for frequently accessed data using \_\_\_\_\_\_\_ (e.g., Redis, Memcached).
  + **Load Balancer**: Distributes incoming traffic to \_\_\_\_\_\_\_ (e.g., multiple backend servers).

*(Fill in the specific components and their roles for your application.)*

## **5. Data Model**

Define the key entities and their relationships.  
*(Include tables/entities such as users, messages, URLs, and other critical components.)*

* **Example for a Chat Application:**
  + **User**: Attributes include \_\_\_\_\_\_\_ (e.g., ID, name, email).
  + **Message**: Stores \_\_\_\_\_\_\_ (e.g., sender, receiver, timestamp).
  + **Chat Room**: Includes members and \_\_\_\_\_\_\_ (e.g., messages).
* **Example for a URL Shortener:**
  + **URL**: Stores original URL, short URL, and access count.
  + **User**: Tracks user details and associated URLs.
  + **Analytics**: Records click data such as timestamp and location.

*(Define your application's entities and relationships.)*

## **6. API Endpoints**

List the APIs needed to support the features.

* **Example for a Chat Application:**
  + **POST /register**: Register a new user with \_\_\_\_\_\_\_ (e.g., user details).
  + **POST /login**: Authenticate user and return \_\_\_\_\_\_\_ (e.g., session token).
  + **POST /message**: Send a message from \_\_\_\_\_\_\_ (e.g., user A to user B).
  + **GET /chatroom/{id}**: Fetch messages for \_\_\_\_\_\_\_ (e.g., a specific chat room).
* **Example for a URL Shortener:**
  + **POST /shorten**: Accepts a long URL and returns \_\_\_\_\_\_\_ (e.g., a short URL).
  + **GET /{shortUrl}**: Redirects to the original URL.
  + **GET /analytics/{shortUrl}**: Fetches usage statistics like click counts.

*(Specify your application’s API endpoints and their functions.)*

## **7. Scaling Strategy**

Detail how you will scale the application as user load increases.

* **Examples:**
  + Scale horizontally by adding \_\_\_\_\_\_\_ (e.g., more backend servers).
  + Implement database sharding for \_\_\_\_\_\_\_ (e.g., distributing user data).
  + Use a messaging broker like \_\_\_\_\_\_\_ (e.g., RabbitMQ) for efficient communication.
  + Caching popular data using \_\_\_\_\_\_\_ (e.g., in-memory databases like Redis).

*(Define the strategies specific to your application.)*

## **8. Security Measures**

Describe how you will secure the system.

* **Examples:**
  + **Authentication**: Use \_\_\_\_\_\_\_ (e.g., JWT tokens) for secure access.
  + **Data Encryption**: Encrypt sensitive data using \_\_\_\_\_\_\_ (e.g., TLS, HTTPS).
  + **Rate Limiting**: Prevent abuse by limiting \_\_\_\_\_\_\_ (e.g., login attempts or API calls).

*(Outline your application's security measures.)*

## **9. Monitoring & Maintenance**

Explain how you will monitor and maintain the system’s health.

* **Examples:**
  + Use monitoring tools like \_\_\_\_\_\_\_ (e.g., Grafana) to track server load and uptime.
  + Set up alerts for \_\_\_\_\_\_\_ (e.g., API failures, high latency).
  + Regular database backups to prevent \_\_\_\_\_\_\_ (e.g., data loss).
  + Automated scaling policies for \_\_\_\_\_\_\_ (e.g., handling peak traffic).

*(Describe your application’s monitoring and maintenance strategies.)*