System Design for Design a URL Shortener

**Architecture**

**1. High-Level Architecture**

The URL shortener system is composed of several key components:

1. **API Gateway:** Handles incoming requests and routes them to the appropriate service and handle the authorization and authentication at first place.
2. **URL Shortener Service:** Generates short URLs and stores mapping to long URLs.
3. **Redirect Service:** Redirects users from short URLs to long URLs.
4. **Database:** Stores URL mappings and statistics.
5. **Cache:** Caches frequently accessed URLs to reduce database load.
6. **Analytics Service:** Collects and processes usage statistics.
7. **Management Service:** Manages URL updates, deletions, and expired link will be deleted on scheduled job during non-business hours
8. **Front end mS:** Will be designed in Angular or React to interact with the application and perform the necessary task.
9. **Postgres DB :** will be used to store the URL and other details.
10. **Azure active directory:** will be used for user authorization and authentication along with JWT token to secure the API’s exposed by each microservice . On top of this we are Ingress router to connect to internet.

****

**Detailed Service Design**

**1.1. URL Generation Service**

**Responsibilities:**

* Generate short URL aliases.
* Handle custom short links.
* Store and manage URL mappings.
* Scheduled JOB to remove the expired link.

**1.2. Redirection Service**

**Responsibilities:**

* Redirect from short URLs to long URLs.
* Collect and process usage statistics.
* Cache frequently accessed URLs to reduce database load.
* Display the URL for the respective users.

**1.3. Management Service**

**Responsibilities:**

* Manage URL updates and deletions.
* Handle custom expiration settings.
* Handle the archive process.
* Delete the expired link systematically

**1.4. API Gateway**

**Responsibilities:**

* Route requests to the appropriate microservices.
* Handle cross-cutting concerns like authentication, logging

**1.5. Cosmos & Postgres DB**

First Cosmos NoSQL DB will be used for of the shortener service will be used to store the URL shortening request and same data will be replicated to SQL Postgres DB associated with the microservice Redirect and analytics.

Postgres DB will always stay sync with first Postgres DB and will handle all CRUD operation along with redirect request. In this DB tables will be indexed partitioned basis on Range, so that it will support better scalability.

**Note:** Further scalability we can think of database sharding to store the data basis on region in case the application will be enhanced to support multi region.

**Non-Functional Requirements**

1. **Availability:**  
   Ensure high availability and fault tolerance. Would be any one option

* containerized the application and deploy on App service under cloud plat form.
* Read access Geo-redundant storage as replication factor in case any failure in primary location data can be accessed from secondary region.
* Set the notification alert on each microservices. So that necessary action can be taken.
* **Liveness and Readiness Probes:** Continuously monitor the health of your microservices. Kubernetes automatically restarts pods that fail liveness checks and routes traffic away from pods failing readiness checks

1. **Scalability:**  
   Design for horizontal scaling.

Create the Kubernetes yml file and keep 10 replicas and configure the CPU utilization crosses 90 % then it will automatically the brings the other pods active

Deploying under Azure app services will take care on scalability.

1. **Readability:**  
   Ensure short links are readable and typeable.

Use only alphanumeric characters in short links. Avoid special characters

1. **Latency:**  
   **Edge Caching:** Distribute static content (e.g., images, scripts) across global edge locations to ensure that users can access data from a location close to them, reducing latency.

**In-Memory Caching:** Spring Cache for Redis to store frequently accessed data in-memory, reducing the need to fetch data from slower data sources like database

1. **Unpredictability:**  
   Ensure short links are unpredictable to prevent guessing.