People Tech – Assignment Week 4

Swetha Kare

URL Shortening Service

High-Level Architecture

1. **Client Layer**:
    - Provide both a web interface and a mobile application for users to submit and manage their URLs, enhancing accessibility.
2. **API Gateway with Throttling**:
    - Use an API gateway that not only routes requests but also implements dynamic throttling based on user behavior. For instance, if a user submits too many URLs in a short period, throttle their requests to prevent abuse.
3. **Shortening Service with Customization**:
    - Allow users to create custom aliases for their shortened URLs, giving them branding opportunities and making the URLs more memorable.
4. **Event-Driven Architecture**:
    - Use an event-driven approach with message queues (like Kafka or RabbitMQ) to decouple services. For example, when a new URL is shortened, an event can trigger analytics updates without blocking the shortening service.
5. **Storage Layer with Multiple Backends**:
    - Utilize a multi-database strategy, storing mappings in a NoSQL database (like MongoDB) for rapid access while maintaining a SQL database (like PostgreSQL) for analytics and reporting.
6. **Redirection Service with Geo-Location Support**:
    - Implement geolocation-based redirection, allowing users to direct traffic based on geographic locations (e.g., redirect users in different countries to different landing pages).
7. **Analytics and Reporting Dashboard**:
    - Provide an interactive dashboard for users to view detailed analytics about their shortened URLs, including click-through rates, referrer data, and geographic distribution. Use visualizations for better insights.
8. **Cache Layer with Intelligent Expiration**:
    - Integrate a smart caching mechanism that dynamically adjusts expiration times based on usage patterns, ensuring that frequently accessed URLs remain cached longer.

9. **Multi-Tenant Architecture**:
    - Implement a multi-tenant system to allow different organizations to use the URL shortener while keeping their data isolated. This can include custom branding and usage metrics.
10. **Security Features**:
    - Implement a verification mechanism to check the validity of submitted URLs (e.g., using URL whitelisting).
    - Introduce optional password protection for private shortened URLs, allowing users to share links securely.

Detailed Components

1. Shortening Service

- **Input**: Original URL and optional custom alias.
- **Output**: Shortened URL.
- **Logic**:
    - Validate the original URL and check for existing mappings.
    - Generate a unique identifier or use the provided alias, ensuring it adheres to character limitations.
    - Store the mapping in the appropriate database.

2. Redirection Service

- **Input**: Shortened URL.
- **Output**: Redirect to original URL.
- **Logic**:
    - Parse the shortened URL to retrieve the identifier.
    - Implement a caching layer for rapid access to the original URL.
    - Optionally redirect users based on their geographic location.

3. Database Schema

- **URL Mapping Table (NoSQL)**:
    - short_url: Unique shortened URL identifier.
    - original_url: Long URL.
    - user_id: ID of the user who created the short URL (for multi-tenancy).

    - custom_alias: Optional custom alias.
    - created_at: Timestamp of creation.
    - click_count: Number of times the shortened URL was accessed.
- **Analytics Table (SQL)**:
    - short_url: Foreign key to the shortened URL.
    - click_timestamp: Timestamp of each click.
    - geo_location: Geographic data of the user.
    - referrer: Where the click originated from.

4. Event-Driven Architecture

- Use an event bus to handle URL shortening and analytics processing. For example, when a URL is shortened, an event triggers the analytics service to log the creation without holding up the user's experience.

Example Workflow

1. **User Submits URL with Optional Alias**:
    - The user submits a URL through the API or mobile app, optionally providing a custom alias.
2. **Shortened URL Generated**:
    - The shortening service validates and generates the shortened URL, storing it in both the NoSQL and SQL databases.
3. **User Accesses Shortened URL**:
    - The redirection service retrieves the original URL from cache or database and performs the redirection, optionally using geolocation for tailored experiences.
4. **Analytics Tracking**:
    - Click events are asynchronously processed to update the analytics database and generate reports without affecting performance.

WhatsApp chat application System Design

High-Level Architecture

1. **Client Layer**:
    - Mobile applications for iOS and Android, along with a web version for accessibility.
    - Implement real-time communication features and user-friendly interfaces for chats, groups, and multimedia sharing.

2. **API Gateway**:
    - Acts as the entry point for all client requests. It handles authentication, rate limiting, and routes requests to the appropriate microservices.
3. **User Management Service**:
    - Handles user registration, authentication, and profile management. It may use OAuth or JWT for secure authentication.
4. **Chat Service**:
    - Responsible for managing one-on-one chats and group conversations, storing chat histories, and managing message states (sent, delivered, read).
5. **Notification Service**:
    - Manages push notifications for new messages, mentions, or other alerts. This service can also handle silent notifications for background message checks.
6. **Media Storage Service**:
    - Manages the storage and retrieval of multimedia files (images, videos, documents) shared in chats. This service can utilize cloud storage solutions for scalability.
7. **Search Service**:
    - Allows users to search for messages, users, and media within their chat history. It should provide efficient indexing and querying mechanisms.
8. **Analytics Service**:
    - Tracks user engagement, message statistics, and other metrics to help improve the user experience and monitor app performance.
9. **Caching Layer**:
    - Implements caching (e.g., Redis) for frequently accessed data, such as user profiles and recent messages, to enhance performance.
10. **Monitoring and Logging**:
    - Integrates monitoring tools to track application performance and logs for debugging purposes.

Unique Features and Components

1. User Management Service

- **Input**: User registration data (phone number, name).
- **Output**: User profile and authentication tokens.
- **Logic**:

    - Use phone numbers as unique identifiers for users.
    - Implement two-factor authentication (2FA) for added security during the registration and login process.

2. Chat Service

- **Input**: Messages from users.
- **Output**: Message delivery status and chat history.
- **Logic**:
    - Use WebSockets for real-time messaging. Each client maintains a persistent connection for immediate updates.
    - Store messages in a NoSQL database (like MongoDB) for scalability, organizing them by user ID and timestamp.

3. Notification Service

- **Input**: Events for new messages, mentions, or group activities.
- **Output**: Push notifications to user devices.
- **Logic**:
    - Use Firebase Cloud Messaging (FCM) or similar services for push notifications.
    - Implement a system to prioritize notifications based on user activity and preferences.

4. Media Storage Service

- **Input**: Multimedia files uploaded by users.
- **Output**: URLs to access the stored media.
- **Logic**:
    - Store media files in a cloud storage solution (e.g., AWS S3).
    - Use a content delivery network (CDN) to speed up media access for users.

5. Search Service

- **Input**: Search queries from users.
- **Output**: Relevant messages, users, or media.
- **Logic**:
    - Use Elasticsearch or similar technologies to index messages and allow efficient searching.
    - Implement fuzzy searching to enhance user experience.

Database Schema

1. **User Table**:
    - user_id: Unique identifier for each user (UUID).
    - phone_number: User's phone number.
    - name: User's display name.
    - profile_picture: URL to the user's profile picture.
    - status: User's online status (online, offline).
2. **Message Table**:
    - message_id: Unique identifier for each message.
    - sender_id: ID of the user who sent the message.
    - receiver_id: ID of the user receiving the message (or group ID for group messages).
    - content: The message content (text, image URL, video URL).
    - timestamp: When the message was sent.
    - status: Delivery status (sent, delivered, read).
3. **Group Table** (for group chats):
    - group_id: Unique identifier for each group.
    - group_name: Name of the group.
    - members: List of user IDs belonging to the group.

Example Workflow

1. **User Registration**:
    - A user registers with their phone number. The user management service validates and sends a verification code.
2. **User Authentication**:
    - Upon verification, the user logs in and receives an authentication token.
3. **Sending a Message**:
    - The user sends a message through the chat service, which validates and stores it in the database.
    - The message is then pushed to the receiver in real-time using WebSockets.
4. **Receiving Notifications**:

- When a new message is received, the notification service triggers a push notification to the recipient's device.

5. **Media Sharing**:
    - If a user shares media, it is uploaded to the media storage service, which returns a URL to the chat service. The chat service sends the media link as part of the message.
6. **Searching Messages**:
    - Users can search for messages in their chat history through the search service, which retrieves relevant results from the indexed data