📱 Lecture 07 : DESIGNING A NOTIFICATION SYSTEM :

🔷 SYSTEM INTRODUCTION & OVERVIEW

Notification System ek aisa system hai jo different services ko allow karta hai users tak various channels ke through alerts/notifications bhejne ke liye.

❓ COMPLETE QUESTION-ANSWER SECTION 💡

🔸 Q1: What are the types of notifications?

✅ **Push Notification (Mobile push)**
✅ **SMS Notification**
✅ **Email Notification**
✅ **In-App Notification**

🔸 Q2: Real-time System vs Soft Real-time System

⚙️ Real-time System

✅ *Definition:*
A **Real-time System** wo hota hai jahan response **immediately (turant)** dena zaruri hota hai — delay allowed nahi hota.

🧩 *Example:*

- Airbag system in cars 🚗
- Missile launch system 🚀
- Pacemaker in heart ⏱️

📌 *Key Point:*
Inme har microsecond matter karta hai — agar delay hua, to **system failure ya damage** ho sakta hai.

🕒 Soft Real-time System (Hamara System)

✅ *Definition:*
A **Soft Real-time System** me system **try karta hai** response *jaldi se jaldi* dene ka, lekin *thoda delay* acceptable hota hai.

💡 *Practical Reality:*
Almost **saare daily-use systems** jaise apps, websites, notification systems, etc., **Soft Real-time** pe hi kaam karte hain.

⚠️ Exceptions (Delay Possible When):

- System **down** ho jaye 🧯
- **Server busy** ho 🖥️
- **Network issue** ya koi **technical glitch** ⚡

💬 Example (Real-world Analogy):

👉 Suppose aapne **LinkedIn** par kisi ko *Follow Request* bheji,
Aur notification **5 minute baad** gaya —
to user ko zyada fark **nahi padega**, right?
Isiliye yeh **Soft Real-time System** ka part hai.

🔶 Q3: Device Type

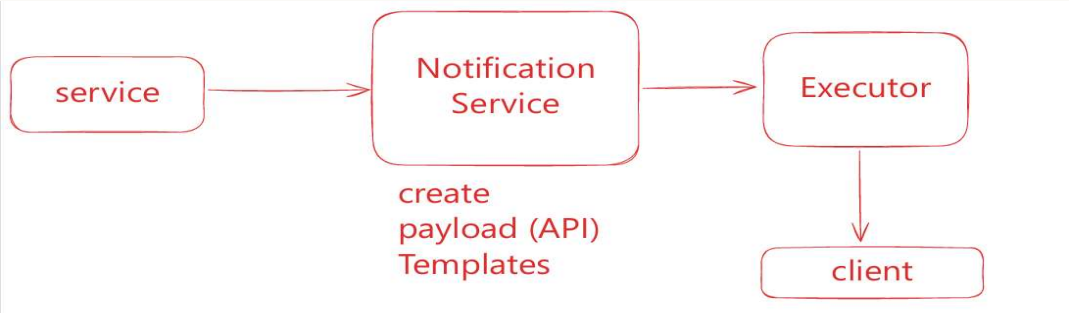
📱 **iOS**
🤖 **Android**
💻 **Web (Windows, macOS)**

🏗️ CORE ENTITIES & COMPONENTS

📌 Main Core Entities:

- **Service** → Notification create karta hai
- **Notification** → Actual message/content
- **Templates** → Pre-defined formats for different notifications
- **Executor** → Notification actually deliver karta hai
- **Client** → End user jo notification receive karta hai

Workflow:



`Service → Create → Payload (API) → Templates → Executor → Client`

📊 BACK OF THE ENVELOPE CALCULATIONS 🧮

📌 Basic Assumptions:

- **Daily Active Users (DAU):** 1 Million
- **Notifications per user:** 10

➡️ Total notifications/day:

```
1 million × 10 = 10 million/day
```

💾 Storage Unit Calculations:

```
1 notification = 200 bytes (content, timestamp, userId, notificationId etc)

Total storage = 10 million × 200 bytes

              = 10 × 10^6 × 200 bytes

              = 2,000,000,000 bytes

              = 2GB per day
```

Detailed calculation:

```
10 × 2^20 × 0.2 × 10^3

= 10 × 2^20 × 0.2 × 2^10

= 2^30 × 2

≈ 2GB/day
```

📦 Monthly Storage:

```
2GB × 30 = ~60GB/month
```

🌐 Bandwidth Calculations:

Definition: Bandwidth = Maximum data transmitted through network in given time.

```
Notification payload = 1KB (upload/download)

Includes: headers, body, request parameters via HTTP


DAU = 1 million

Total notifications/day = 10 million

Total data = 1KB × 10 million = 10GB/day
```

Bandwidth per second:

```
10 million × 1 KB / 86,400 seconds

≈ 115 KB/sec
```

💡 **Server Load:**
Approx **115KB/sec** bandwidth usage — manageable with scaling.

⚙️ QPS (Queries per Second)

Writes (send notifications):

```
10 million / 86,400 = ~115 QPS
```

Reads (fetch logs/templates):

```
~3 × write QPS = ~345 QPS
```

✅ **Total QPS:**

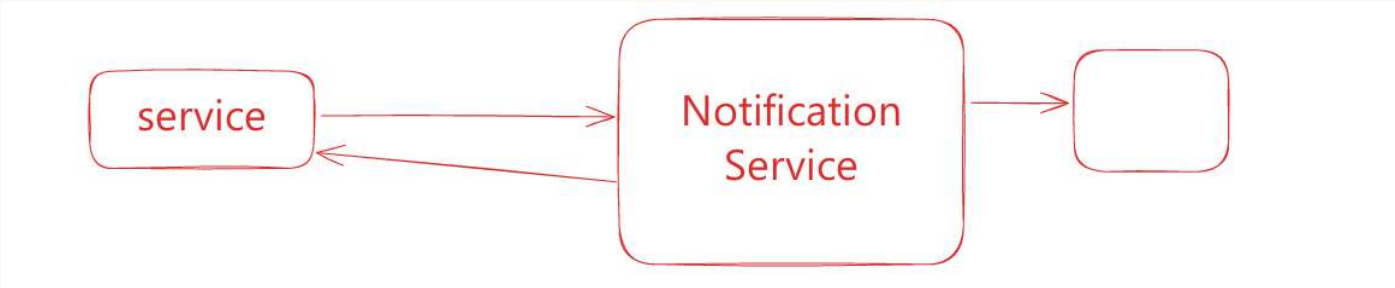
```
115 (write) + 345 (read) ≈ 460 QPS
```

⚙️ SERVER SCALING & API INTERACTIONS

📌 Server Scaling:

- 🧠 **Vertical scaling:** Increase server RAM/CPU.
- 🌍 **Horizontal scaling:** Multiple servers for load distribution.

🛠️ COMPLETE API DESIGN



1️⃣ User Information Fetch API

```
GET : /fetchuser/{{userId}}

Response:

200 OK

{
  "userId" : 101,
  "userName" : "Aditya",
  "notificationPermissions" : {
    "PUSH" : true,
    "Email" : true,
    "SMS" : false
  }
}
```

2️⃣ Send Notification API

```
POST : /sendNotification/{{userId}}

Body :

{

  "from" : null,

  "to" : "202",

  "subject" : "",

  "content" : "",

  "type" : "push/email/sms"

}

Response:

200 OK

{

  "status" : "sent",

  "notificationId" : "301"

}
```

3️⃣ Fetch Notification History API (Optional)

```
GET : /fetchAllNotifications

Query Parameters: filters

- date : from, to

- Type : email/sms etc.

Response:

200 OK

[

  {

    "status" : "sent",

    "content" : ""
```

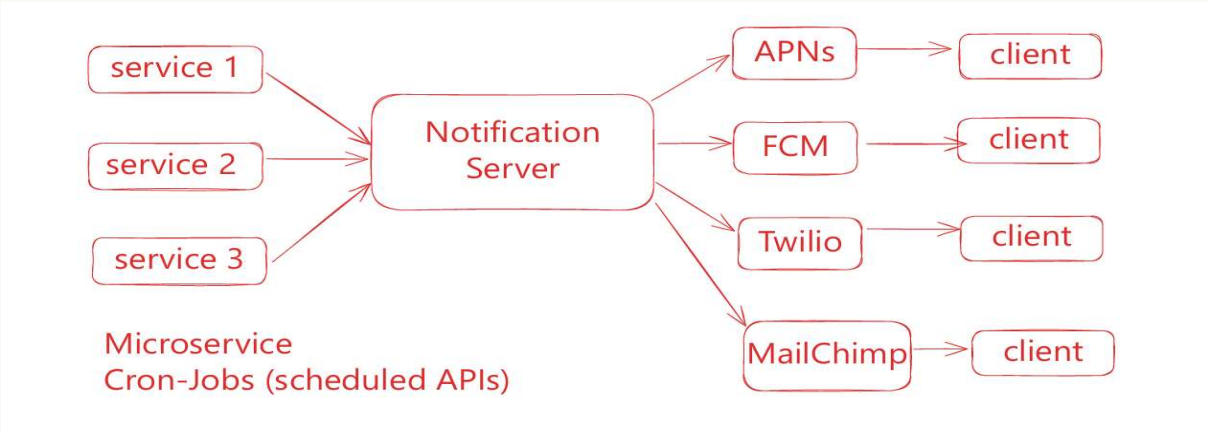
```
  },
  {
    "status" : "sent",
    "content" : ""
  }
]
```

🎯 BASIC SYSTEM DESIGN ARCHITECTURE

🔶 Third-Party Services Mapping

```
Push Notifications → Mobile
├── 🍎 iOS → APNs (Apple Push Notifications)
├── 🤖 Android → FCM (Firebase Cloud Messaging)
├── 📱 SMS Notification → Twilio, Nexmo
└── 📧 Email → MailChimp, SendGrid
```

🧩 Architecture Diagram



🔄 Complete Flow:

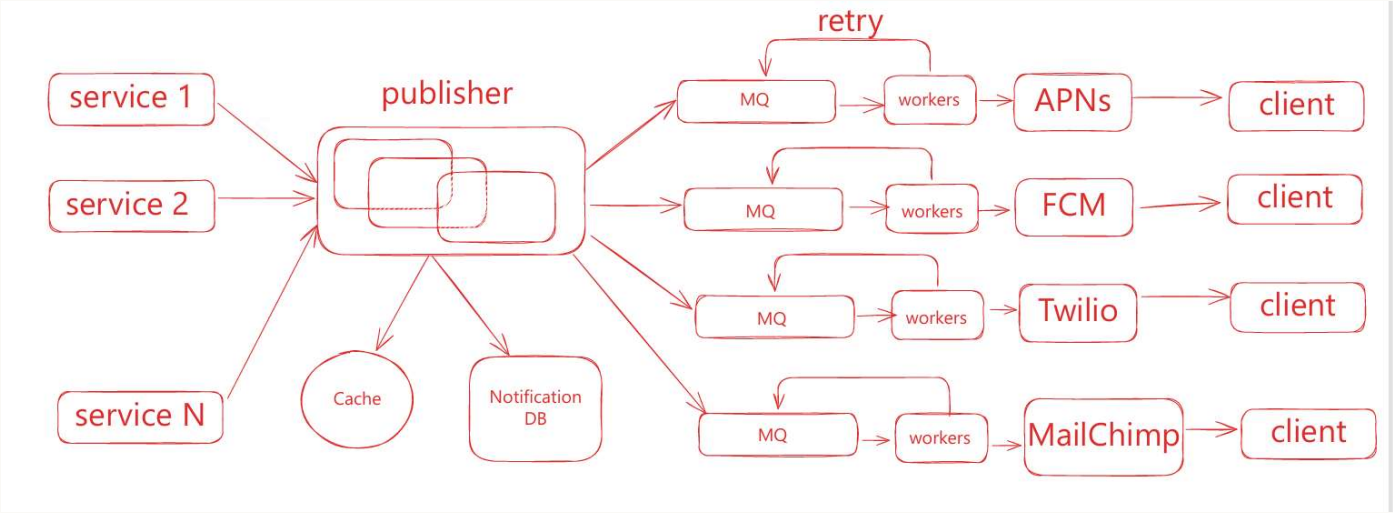
1. Any service (microservice, cronjobs) triggers notifications → sends to Notification Server.
2. Notification Server builds payload → forwards to 3rd-party services.
3. Third-party service actually sends notification to client.

⚠️ COMPLETE PROBLEMS WITH INITIAL DESIGN

Problem	Description	Solution
❌ **SPOF (Single Point of Failure)**	Ek hi notification server hone se failure par system down.	Scale horizontally, multiple servers use karo.
⚙️ **Hard to Scale**	Manual scaling inefficient.	Use load balancers, auto-scaling.
🔁 **Retry Mechanism Missing**	Fail hone par notification resend nahi hota.	Retry logic add karo.
🔗 **Tight Coupling (Synchronous)**	Sab services directly connected → ek slow ho to sab delay.	Async design via MQ.

🚀 IMPROVED ASYNC DESIGN ARCHITECTURE

🔶 New Components Added:



📦 Component Details:

- 📨 **Message Queue (MQ):**
  - Multiple MQ instances → high reliability.
  - Removes dependency between Notification Server & 3rd-party APIs.
  - Converts system → **Async architecture.**
- 💾 **Cache & Database:**
  - Store user info, notification templates, etc.
  - Provide fast access & persistence.
- 👷 **Workers:**
  - Multiple parallel processors.
  - Implement **retry mechanism** for failed notifications.

✅ Improvements Achieved:

- 💪 No SPOF → multiple distributed workers.
- ⚡ Easy scaling → workers can scale independently.
- 🔁 Retry mechanism ensures reliability.
- 🔒 Loose coupling → async services work independently.

✨ ADDITIONAL FEATURES & ENHANCEMENTS

1️⃣ 📋 Notification Templates:

- Pre-defined formats → consistency in messages.

2️⃣ ⚙️ Notification Settings (User-Level):

- Users can enable/disable specific notification types.

3️⃣ 🚦 Rate Limiting:

- Limit notifications per user → prevent spam/abuse.

4 🔐 API Authentication:

- Secure APIs → only authorized services can send notifications.

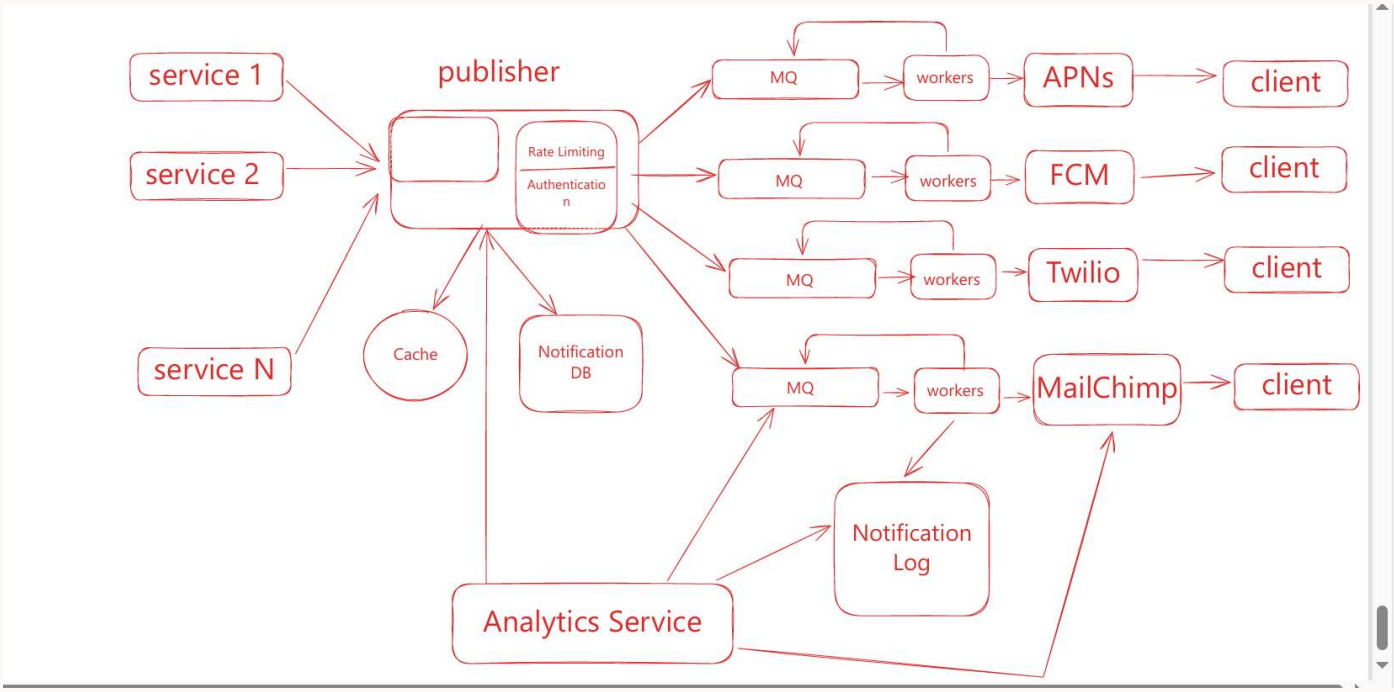
5 📊 Monitoring Logic:

- System performance & analytics tracking.

6 📝 Notification Log:

- Full record of sent notifications for debugging & analysis.

🧩 Final Architecture with New Features :



📚 COMPLETE POWERFUL SUMMARY 🌟

🎯 System Fundamentals:

- **Type:** Soft Real-time System
- **Notification Types:** Push, SMS, Email, In-App
- **Devices:** iOS, Android, Web

📊 Capacity Planning:

- **DAU:** 1 Million
- **Total Notifications/day:** 10 Million
- **Storage:** 2GB/day → 60GB/month
- **Bandwidth:** 115KB/sec
- **QPS:** ~460/sec (115 write + 345 read)

🏗️ Architecture Evolution:

```
Initial → Synchronous → Problems → Improved → Async with MQ
```

🔧 Core Components:

```
Front  → Services, Cron Jobs

Middle → Message Queue, Workers, Cache

Back   → Database, Third-party Services

Extra  → Analytics, Logs, Rate Limiting
```

🚀 Scalability Features:

- Horizontal scaling of workers
- Async processing through MQ
- Load distribution across multiple workers
- Fault tolerance with retry mechanisms

🛡️ Reliability Features:

- Retry mechanism for failed notifications
- Rate limiting to prevent abuse
- API authentication for security
- Monitoring and logging for transparency

💡 Key Design Decisions:

- Introduced **Message Queue** → decoupling
- Used **Worker-based architecture** → scalability
- Added **Cache layer** → performance improvement
- Implemented **Retry mechanism** → reliability
- Integrated **Rate Limiting** → protection

📜 FINAL TAKEAWAY

A **Soft Real-Time, Asynchronous, Distributed Notification System** capable of sending **millions of notifications daily**, built with **Message Queues, Workers, Retry Mechanisms, and Rate Limiting** for maximum **scalability, reliability, and security.** 🚀