COMSATS University Islamabad
Abbottabad Campus

- **NAME**: Muttayyab Abdurrehman

- **REGISTRATION NUMBER**: FA22-BSE-046

- **Assignment NO:** 2

- **COURSE**: Software Design and Architecture

1. Zotero (Reference Management)

• Category:
Reference Management

• Description:
Zotero is an open-source reference manager used for organizing and managing bibliographic data and related research materials. It helps researchers collect, organize, cite, and share sources in academic papers.

Architectural Problem:

- **Scalability Issues with Large Datasets**: Zotero is a reference manager that supports users with thousands of references. As libraries grow larger, syncing and managing a vast number of documents can introduce **performance bottlenecks** and **syncing delays** between devices.

Solution:

- Zotero underwent architectural improvements by adopting a **client-server model** with local and cloud syncing. By offloading data management to a cloud server, Zotero ensures scalability. The cloud synchronization improves performance by allowing users to access their references seamlessly across devices without relying on a single monolithic database.

2. CockroachDB (Distributed SQL Database)

• Category:

Database Management System

• Description:

CockroachDB is a distributed SQL database designed for high availability and scalability. It offers a cloud-native approach and is highly fault-tolerant, making it ideal for large-scale, globally distributed applications.

Architectural Problem:

- **Single Points of Failure**: In early stages, like most distributed systems, CockroachDB initially had challenges in handling **single points of failure** across nodes. This posed risks to the availability and consistency of data.

Solution:

- **Sharding and Distributed Consensus**: CockroachDB adopted **Raft** (a consensus algorithm) and **automatic sharding** for horizontal scaling. Each node participates in the consensus process to ensure there is no single point of failure, ensuring high availability. Data is split across multiple nodes (or shards), and each shard is replicated to ensure fault tolerance.

3. TiddlyWiki (Personal Wiki Software)

• Category:

Personal Wiki Software

• Description:

TiddlyWiki is a unique, highly customizable, and self-contained wiki software that enables users to create personal wikis or knowledge management systems. It runs entirely in a web browser and can store all data within a single HTML file.

Architectural Problem:

- **Data Synchronization**: TiddlyWiki operates as a single HTML file, which makes it easy to store and move between devices. However, users working collaboratively often face **issues with data synchronization when multiple people are editing the same file.**

Solution:

- **Server-Side Syncing and Improved Collaboration**: To overcome this, TiddlyWiki evolved to support server-side storage options, allowing users to synchronize and collaborate across devices. Third-party services, like **TiddlySpace**, have been introduced to help sync the wiki across different platforms while maintaining the single-file format.

4. GIMP (GNU Image Manipulation Program)

• Category:

Graphics Editing

• Description:

GIMP is an open-source raster graphics editor used for tasks such as photo retouching, image composition, and image authoring. It's often considered a free alternative to Adobe Photoshop.

Architectural Problem:

- **Performance Issues on Large Files**: GIMP, like many image editors, struggled with performance when handling large images (e.g., large photo-editing projects, high-resolution images, etc.). This was especially problematic for users with lower-end hardware.

Solution:

- **Multi-threading and GPU Acceleration**: To solve performance problems, GIMP introduced multi-threading support, allowing parallel processing of multiple tasks. Additionally, support for **GPU acceleration** was added to leverage hardware for faster processing, reducing the time needed for tasks like rendering or applying filters on large images.

5. Prey (Device Security)

• Category:

Device Security

• Description:

Prey is a multi-platform tracking and security software designed to protect your devices in case they are lost or stolen. It can track your device's location, take screenshots, lock the device, and even wipe data remotely.

Architectural Problem:

- **Battery Consumption and Background Running**: Prey, which runs as a background process on devices to track and protect them in case of theft, initially faced problems related to **battery consumption** and **resource usage** on mobile devices.

Solution:

- **Optimization for Power Efficiency**: Prey addressed this by optimizing the background tracking processes to use less power. They adopted more efficient **background task scheduling** and data syncing mechanisms, such as pushing location updates only when necessary, and using **low-power modes** to reduce resource consumption.

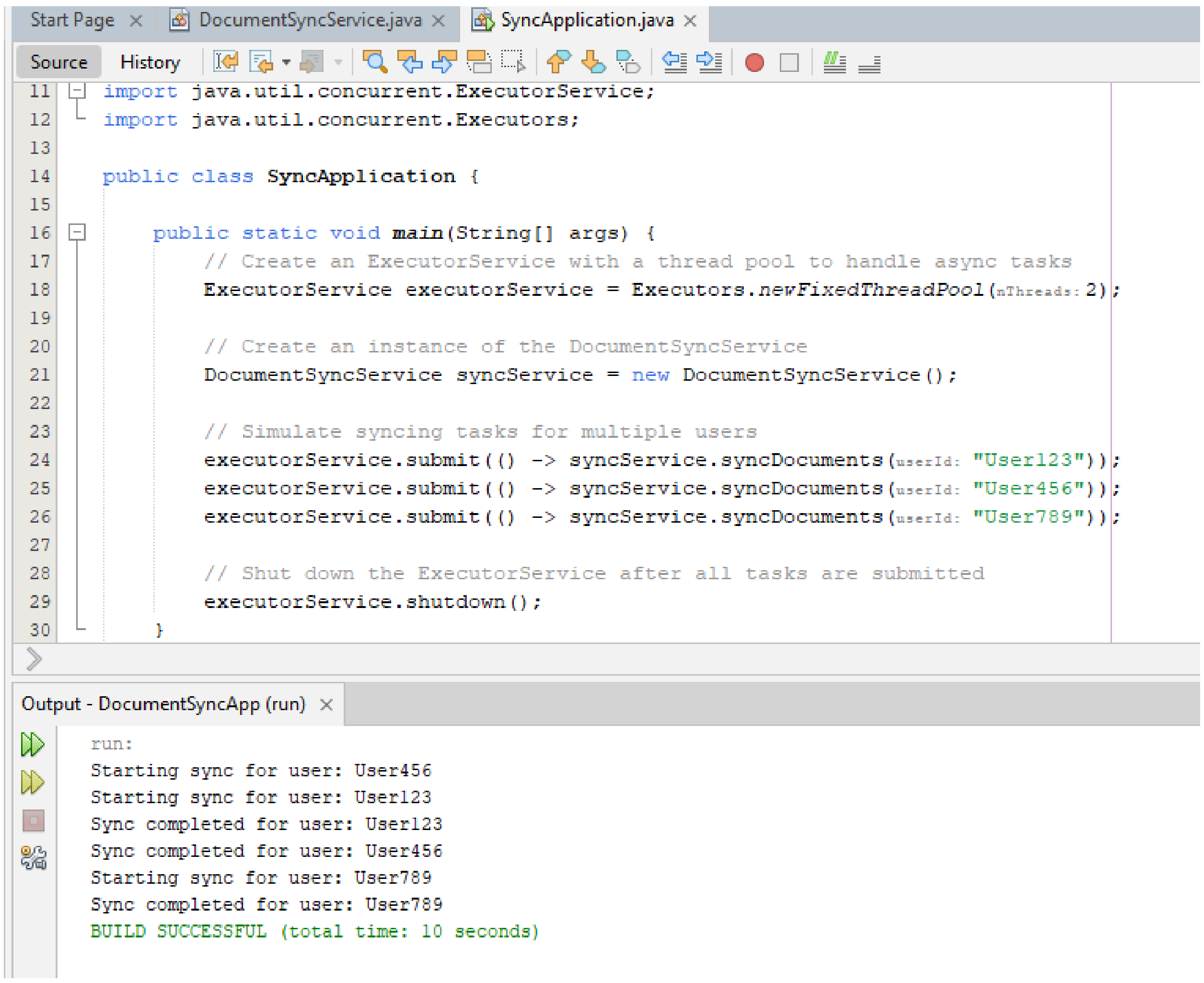
MY CODE:

Problem:

We want to simulate a syncing problem where a user has a large number of documents (e.g., in a Zotero-like system). Instead of blocking the main thread while syncing these documents, we'll use **asynchronous processing** to avoid performance bottlenecks.

Solution:

We'll use **`ExecutorService`** to handle the task of syncing documents asynchronously.



```java
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;

public class SyncApplication {

    public static void main(String[] args) {
        // Create an ExecutorService with a thread pool to handle async tasks
        ExecutorService executorService = Executors.newFixedThreadPool(nThreads: 2);

        // Create an instance of the DocumentSyncService
        DocumentSyncService syncService = new DocumentSyncService();

        // Simulate syncing tasks for multiple users
        executorService.submit(() -> syncService.syncDocuments(userId: "User123"));
        executorService.submit(() -> syncService.syncDocuments(userId: "User456"));
        executorService.submit(() -> syncService.syncDocuments(userId: "User789"));

        // Shut down the ExecutorService after all tasks are submitted
        executorService.shutdown();
    }
```

```
run:
Starting sync for user: User456
Starting sync for user: User123
Sync completed for user: User123
Sync completed for user: User456
Starting sync for user: User789
Sync completed for user: User789
BUILD SUCCESSFUL (total time: 10 seconds)
```