Instagram

1) Store/Get Images →
2) Like + Comment
3) Follow Someone.

Likes

UUID	Type	Active	ActivityID	UserID	Timestamp
	Comment or Post				

Post

→ UserID

PostID	Text	ImageUrl	TimeStamp

Activity

Activity ID	Likes	

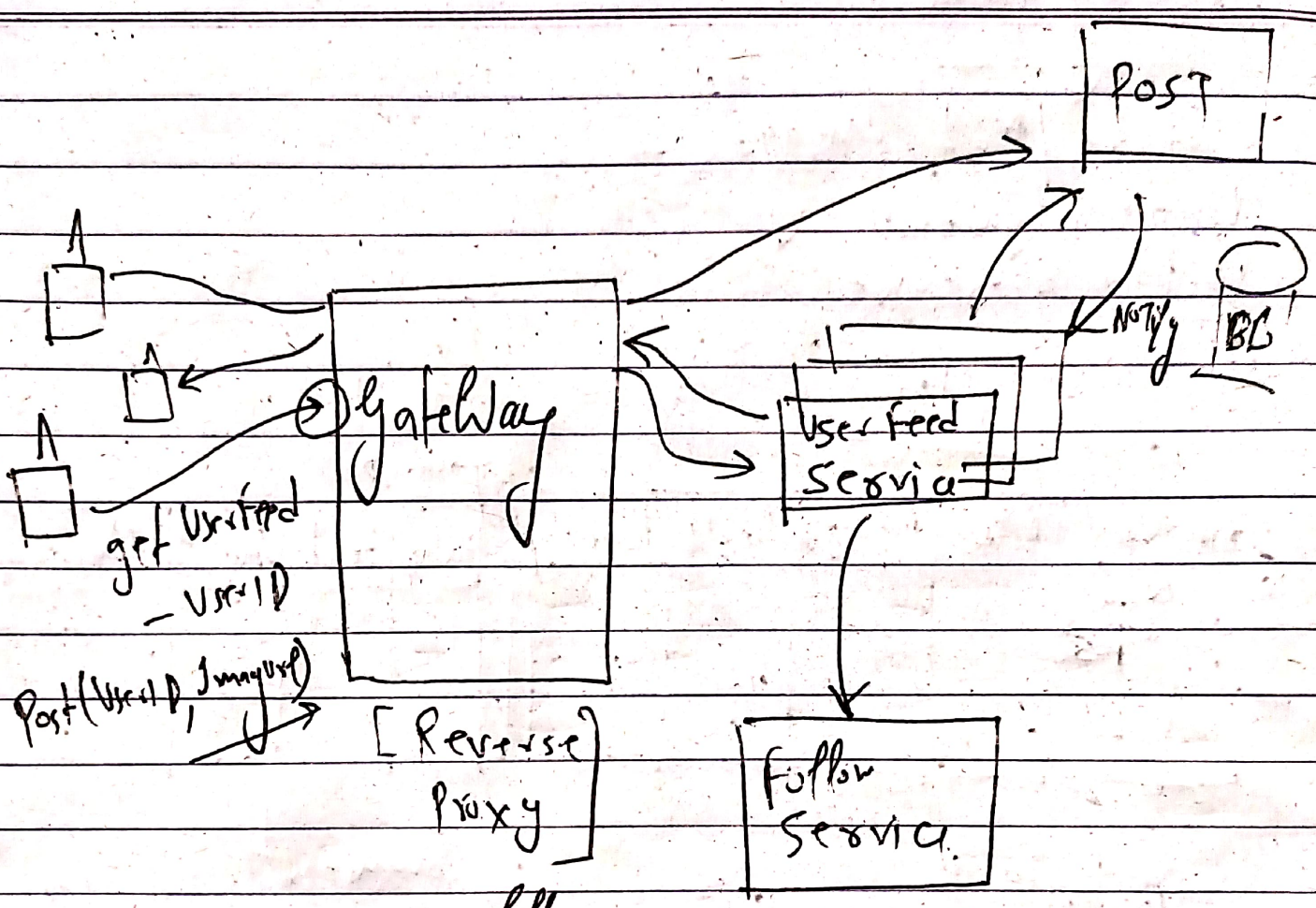
Comment

ID	Text	Timestamp	Activity ID

Select count(*) from likes
where ActivityID = 'id'.

FollowerID	FolloweeID	Timestamp

1) Who follows user X?
2) Which users does User X follow?



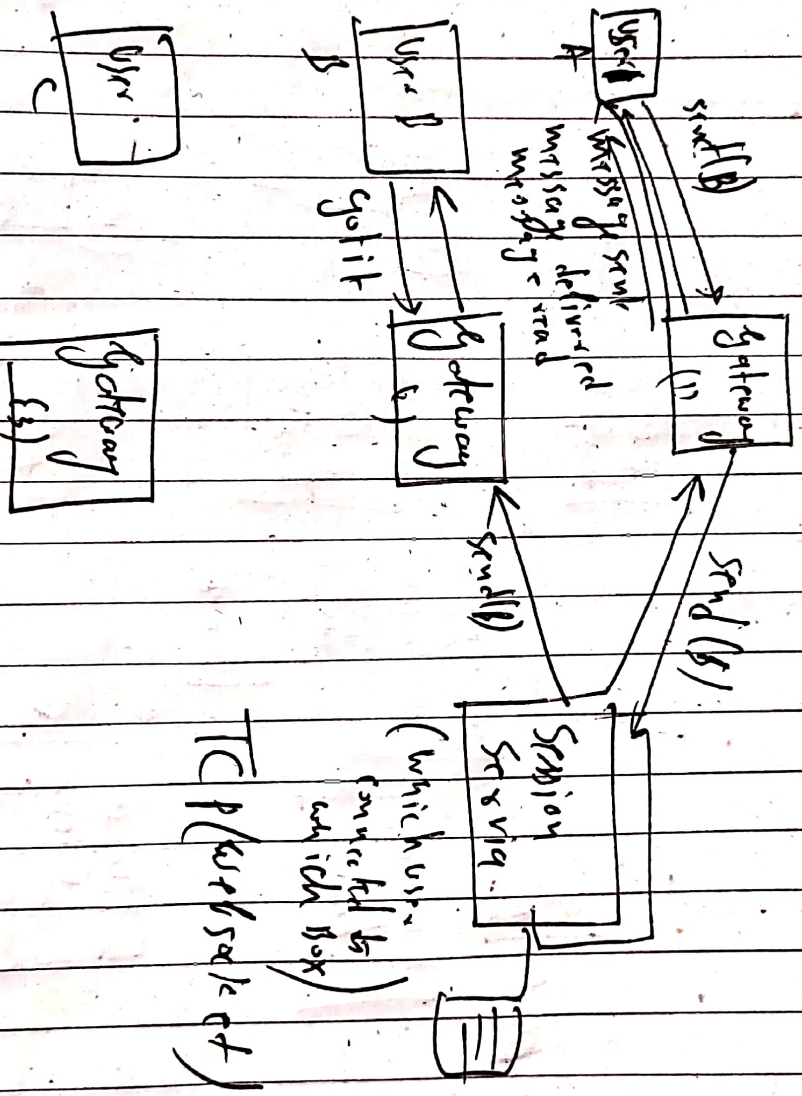
Follower.

Small No. of User → WebSocket (Notify)

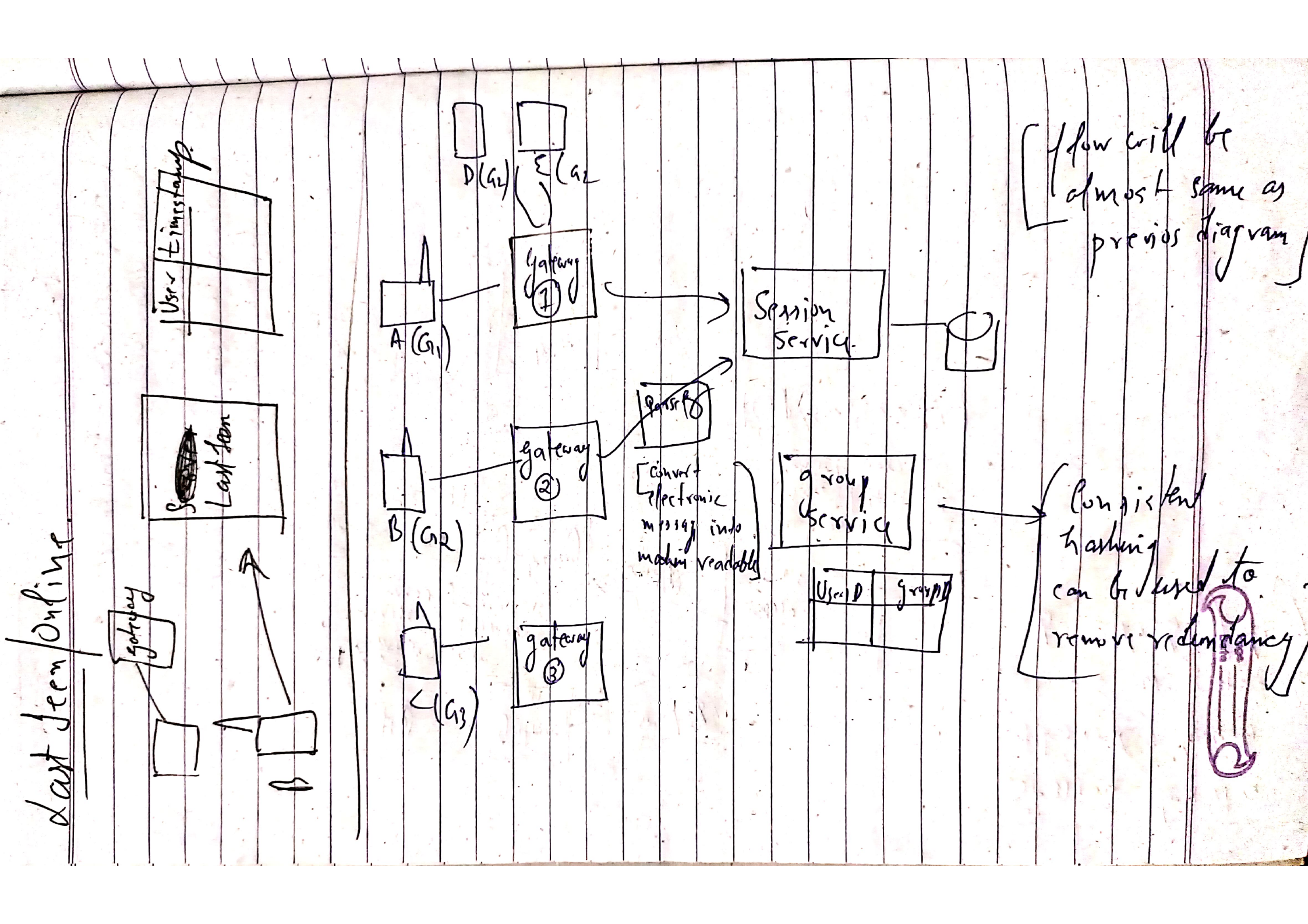
Large no. of Followers → Pulling.

WhatApp

1) Group Message
2) Sent + Delivered + Read Receipt
3) Online / last seen
4) Image sharing
5) Chats are temporary / parmanent
6) One to one chat.



Last seen Online



Date Page

Tinder Architecture

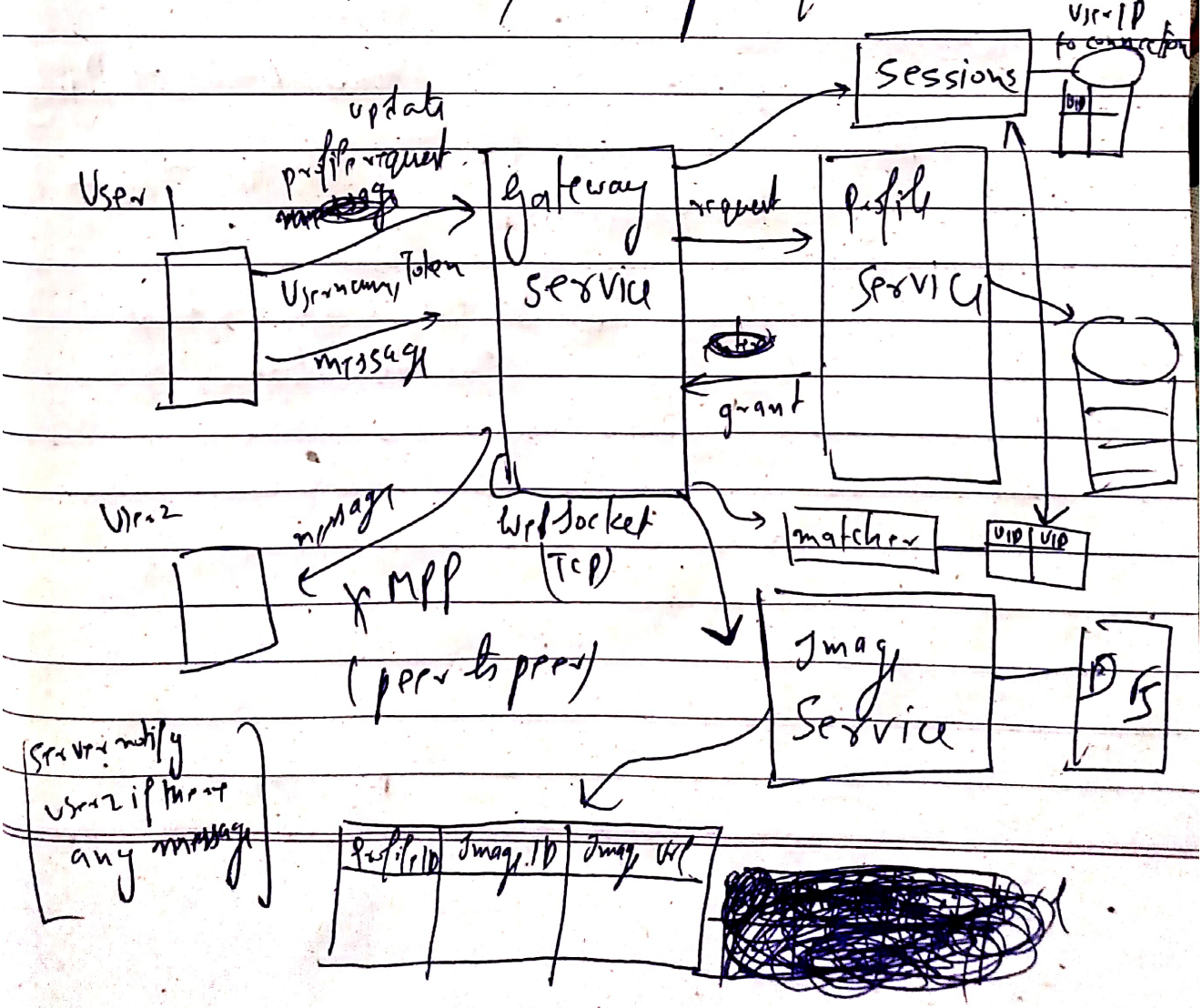
1) Store Profile (Images) – 5 images per user
2) Recommend match (No. of active user)
3) Note matches
4) Direct messaging

File Vs Blob

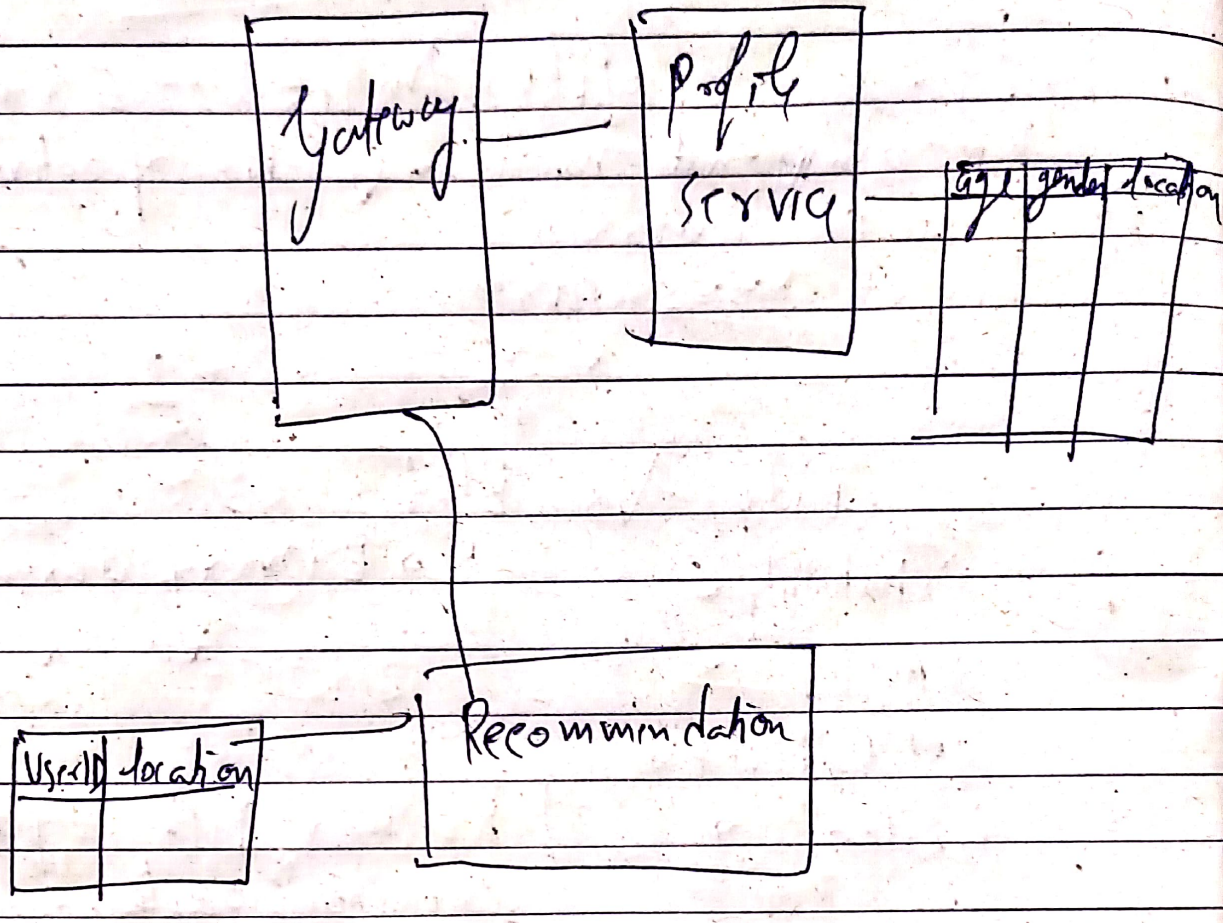
↳ Binary large object

1) Mutability ✗
2) ACID ✗
3) Refactoring ✗
4) Index ✗

Rather file system are cheaper & faster.



Recommendations



1) Nosql

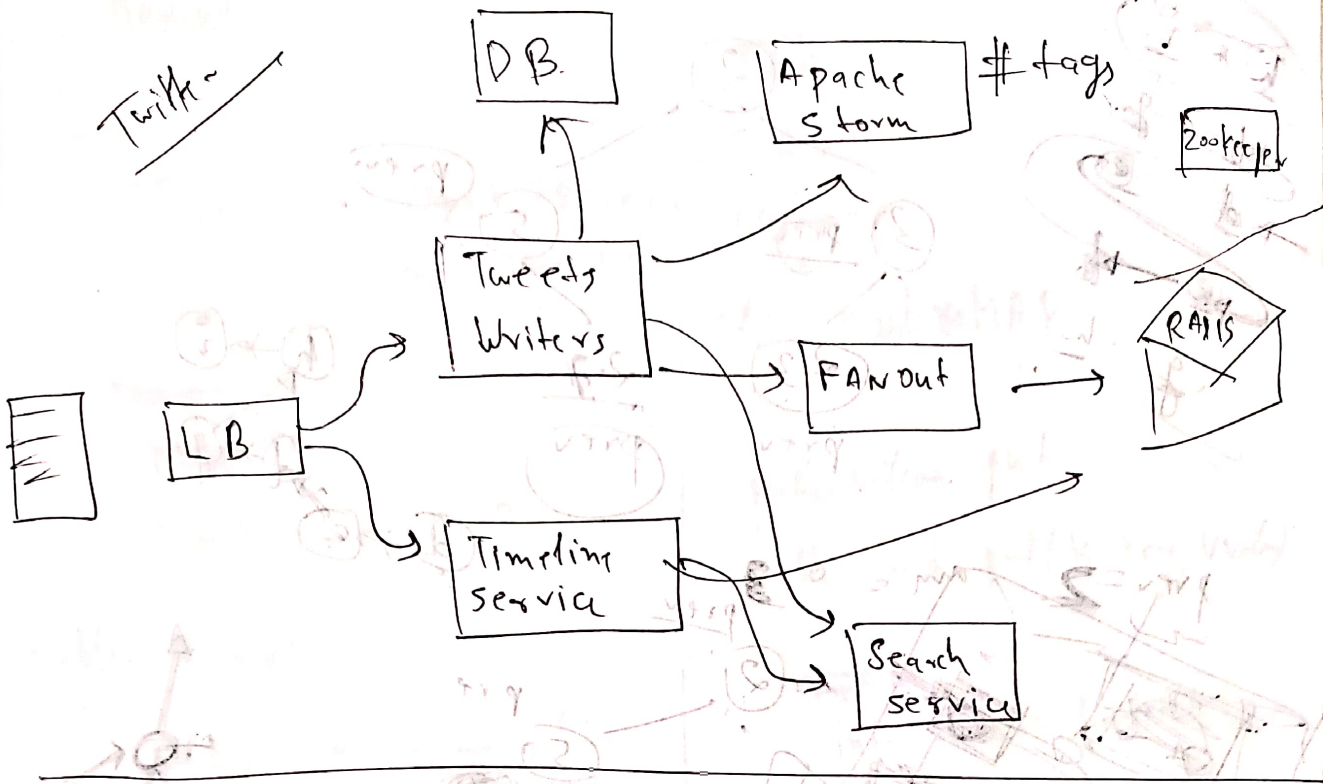
2) Sharding (Horizontal Partitioning)

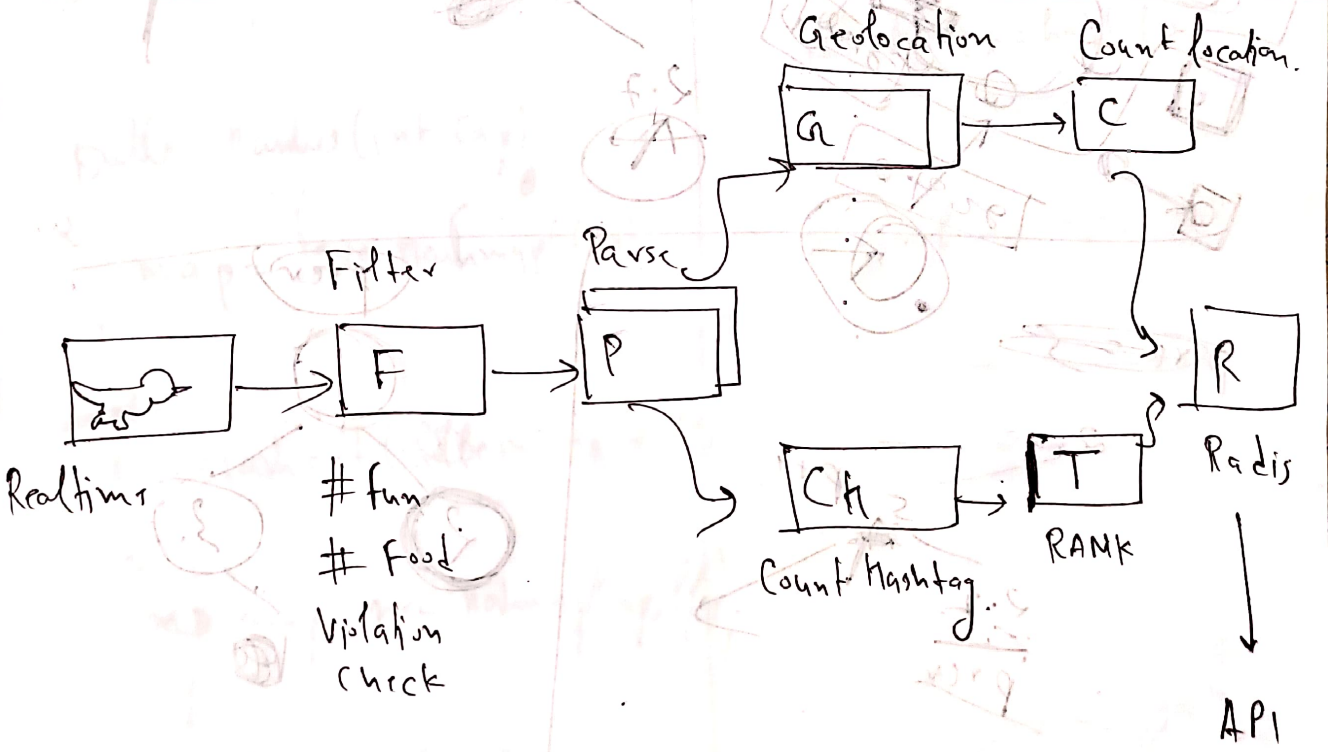
Range (A–J) → DB 1

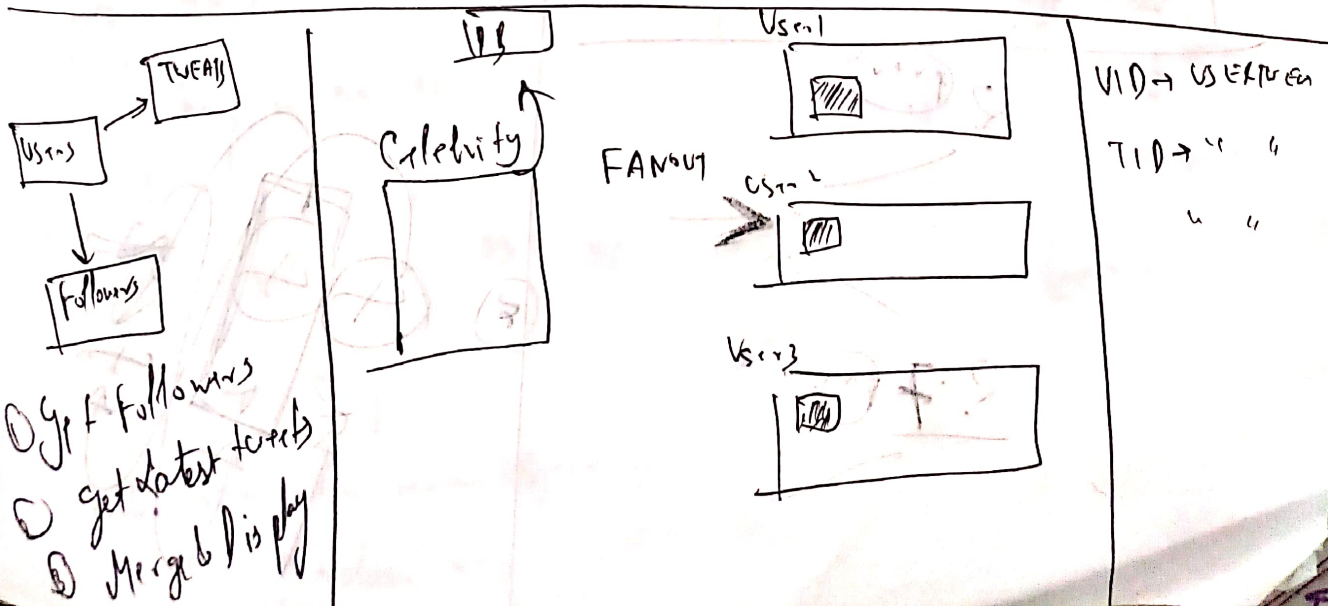
(K–P) → DB → 2

So we can store the location.

Twitter







1) Get followers
2) get latest tweets
3) Merged Display

```
class HashSet < E >
{
    private static final object PRESEN
                  = new Object();

    ~~map = new HashMap<>();~~
    HashMap<> map;

① Public HashSet()
   {
      map = new HashMap<>();
   }

② Public HashSet(int Capacity)
   {
      map = new HashMap(Capacity);
   }

③ Public HashSet(int Capacity, float lf)
   {
      map = new HashMap(Capacity, lf);

④ Public HashSet(Collection E)
   {
      map = new HashMap(E);
   }
```

hs.add("TOM")

hs.add("Peter")

hs.add("Naveen");

public Boolean add(E e)

{ return map.put(e, PRESENT)

==null;

}

public Boolean remove(Object o)

{ return map.remove(o) == PRESENT

;}

MAP

public V put(K key, V value)

{ int hash = hashCode(key);

int index = hash%(n-1);

n = 16;

}

Node<K, V>

hash%NULL

key

V = Value

is

int hashCode

Node<K, V> next

```
public Boolean equals(Object o)
{
    if(o == null || o.getclass != this.getclass())
    { return false;
    }
    if(this == o)
        return true;
        Employee e = (Employee) o;
    return this.getid == o.getid();
}
public hashcode()
{
    return getid();
}
}
```

interface Map<K, V>

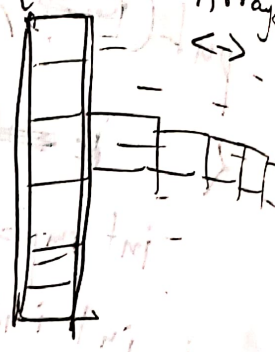
{

void put(key K, V value);

V get(key K);

boolean contains(key K);

}



class HashMap implements Map<K, V>

{

AL<Int-->> V[] = new AL[n];

for(i=0 to n)

V[i] = new AL<Int-->>();

ArrayList<Integer> A[] = new ArrayList<>();

void put(key K, V value)

{

int hash = K.hashCode();

int index = hash % n;

boolean flag = false;

for(Pair p : V[index])

if(p.first == K)

{

for(int i=0; i< v[index].size; i++)

if(v[index][i].first = K)

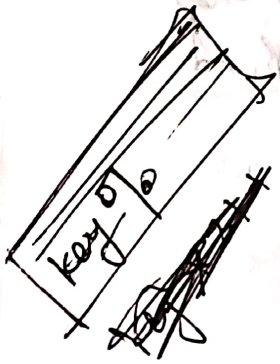
{

v[index][i].second = value;

flag = true;

}

}



```
if(flag)
    if(flag==false)
    {   v[index].padd(new Pair(K,V));
    }
}
void get(key k)
{   int hash = k.hashCode();
    int index = hash%n;
    for(int i=0; i<v[index].size; i++)
    {   if(v[index][i].first=k)
            return v[index][i].second;
    }
    return null;
}
bool contains(key K)
{   int hash = k.hashCode();
    int index = hash%n;
    for(int i=0; i<v[index].size; i++)
    {   if(v[index][i].first==k)
            return true;
    }
    return false;
}
```

```
class Pair
{
    int key;
    int value;
    int first;
    int second;
}

ArrayList<Integer> AL[]
    = new ArrayList[n];
for(int i=0; i<n; i++)
{
    AL[i] = new ArrayList<>();
}

ArrayList<Integer> A[]
    = new ArrayList[n];

class HashSet<E>
{
private static final PRESENT =
        new Object();
    HashMap hashmap;
    HashMap<E,Object> hashmap;

    HashSet()
    {
        hashMap = new HashMap<>();
    }
    HashSet(int cap)
    {
        hashMap = new HashMap<>(cap);
    }
    HashSet(int cap, int ld)
    {
        hashMap = new HashMap<>(cap, ld);
    }
    HashSet(int cap,
```

```
boolean get(int index)
    add(k key)
boolean get(k key)
{
    return map.put(k, PRESENT)
            == null;
}
boolean remove(Object o)
{
    return map.remove(o)
            == PRESENT;
}
boolean contains(K k)
{
    return map.containsKey(k);
}
```

Creational → create object

Structural → adding some functionality (nested object)

Behavioral → Communication

Factory DP (creational)

```
public class factoryMain
{
    public static void main()
    {
        Os obj = new Android();
        obj.spec();
        OperatingSystemFactory osf = new OperatingSystemFa
        Os obj = osf.getIstance("Open");
        obj.spec();
    }
}
public interface Os
{
    void spec();
}
public class Android implements Os
{
    public void spec()
    {
    }
}
public class Ios implements Os
{
    public void spec()
    {
    }
}
```

```
class OperatingSystemFactory
{
    public OS getInstance(String str)
    {
        if(str.equals("Open"))
            return new Android();
        else if(str.equals("Closed"))
            return new IOS();
    }
}
```

Builder DP (creational)

```
public class Phone
{
    private String OS;
    private int ram;
    private String processor;

    public Phone(String os, int ram, String pro)
    {
        this.os = os; this.ram = ram;
        this.processor = pr;
    }
}
```

```
public class PhoneBuilder
{
    private String os;
    private int ram;
    private String processor;
    public PhoneBuilder setOs(String os)
    {
        this.os = os;
        return this;
    }
    public PhoneBuilder setRam(int Ram)
    {
        this.ram = Ram;
        return this;
    }
    public PhoneBuilder setProcessor(String pr)
    {
        this.processor = processor;
        return this;
    }
    public Phone getPhone()
    {
        return new Phone(os, ram, pros);
    }
}
```

```
int main()
{
    Phone p = new PhoneBuilder().setOS("Android")
                .setRam(2).getPhone();
}
```

Adaptor DP (structural)

```
public class PenAdaptor implements Pen
{
    PilotPen pp = new Pilot();

    @Override
    public void write(String str)
    {
        pp.mark(str);
    }
}

class PilotPen
{
    public void mark(String str)
    {
    }
}
```

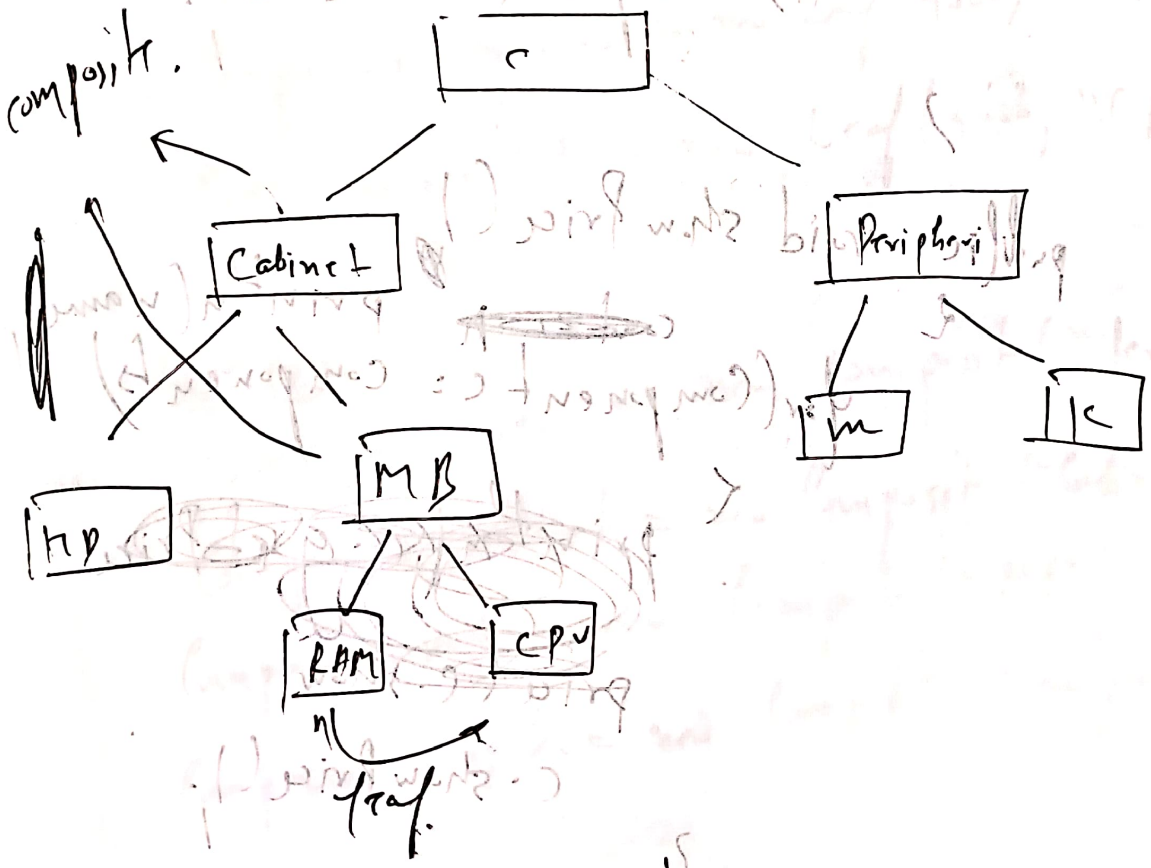
mark

we don't know implementation

```
public class AssignmentWork
{
    private Pen p;
    public Pen getP()
    { return p;
    }
    public void setP(Pen p)
    { this.p = p;
    }
    public void writeAssignment(String str)
    {
        p.write(str);
    }
}
class mainClass
{
    public static void main(String arg[])
    {
        Pen p = new PenMightier();
        AssignWork aw = new AssignmentWork();
        aw.setP(p);
        aw.writeAssignment("Hi Mam");
    }
}
```

public interface Pen
{
    public void write(String str);
}

Composite DP (structural)



interface Component
{
    void showPrice();
}

class Leaf implements Component
{
    String name;
    int price;
    void showPrice()
    {
        System.out.println(name + " " + price);
    }

class Composite implements

{

List<Component> components = new ArrayList

public String name;

addComponent(Component c)

{

this.components.add(c);

}

public void showPrice()

{

System.out.println(name);

for(Component c : components)

{

c.showPrice();

}

}

public Composite(String name)

{

this.name = name;

}

}

```
public class MainClass
{
    public static void main(String[] arg)
    {
        Component hd = new Leaf(4000, "HDD");
        Component mouse = new Leaf(4000, "Mouse");
        Component monitor = new Leaf(4000, "Monitor");
        Component CPU = new Leaf(4000, "CPU");

        Composite ph = new Composite("Peri");
        Composite cabinet = new Composite("Cabinet");
        Composite mb = new Composite("MB");
        Composite computer = new Composite("Computer");

        mb.addComponent(cpu);
        mb.addComponent(ram);
        computer.addComponent(mouse);
        computer.addComponent(monitor);
        computer.addComponent(mb);
        ph.showPrice();
        hd.showPrice();
    }
}
```

Prototype DP (creational)

```
public class Book
{
    private int bid;
    private String bname;
}

public class BookShop implements Cloneable
{
    private String shopName;
    List<Book> books = new ArrayList<>();

    public void loadData()
    {
        for(int i=1; i<=10; i++)
        {
            Book b = new Book();
            b.setBid(i);
            b.setBname("Book"+i);
            this.getBooks().add(b);
        }
    }
```

```
protected BookShop clone() throws CloneNotSupportException
{
    BookShop bs = new BookShop();
    for(Book b : this.getBooks())
    {
        shop.getBooks().add(b);
    }
    return bs;
}

public static void main(String[] args)
{
    Book bs = new BookShop();
    bs.setShopName("Novelfy");
    bs.loadData();

    BookShop bs1 = bs.clone();
    bs.getBooks().remove(2);
    bs1.setShopName("A1");

    System.out.println(bs1);
    print(bs1);

}
```

not cloading for second time using bs (error bs)

Observer DP (Behavioral)

```
public class Subscriber
{
    private String name;
    private Channel channel = new Channel();

    public Subscriber(String name)
    {
        super();
        this.name = name;
    }

    public void update()
    {
        System.out.println("Key" + name +
            "video Uploaded:" + channel.
    }

    public void subscribeChannel(Channel ch)
    {
        channel = ch;
    }
}
```

```
public class Channel
{ private List<Channel>
  private List < Subscriber> subs;
  private String title
public void subsribe (Subscriber s)
   {  subs.add(s);
   }
public void unSubscribe (Subscriber s)
   {     subs.remove(s);
   }
public void notifySubscribers()
   {  for(Subscriber sub : subs)
      {  sub.update();
      }
   }
public void upload (String title)
   { this.title = title;
     notifySubscribers();
   }
}
```