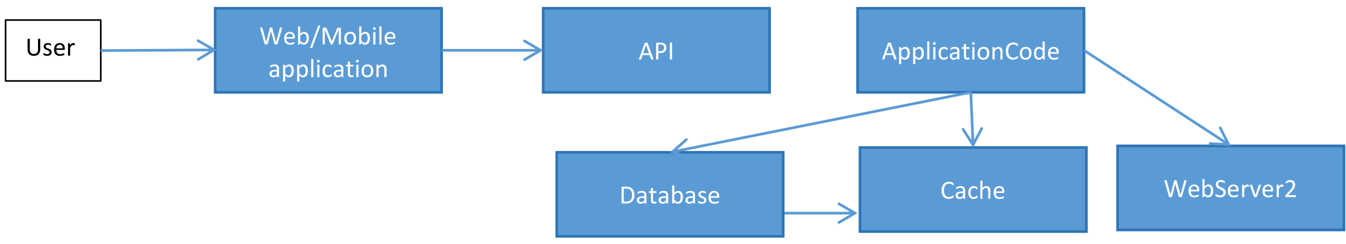
Introduction to System design
What's a system?
A system means we will have many different components and we will solve a common purpose or a business problem combining them.

A system design we do for Instagram is different from a System design we do for WhatsApp

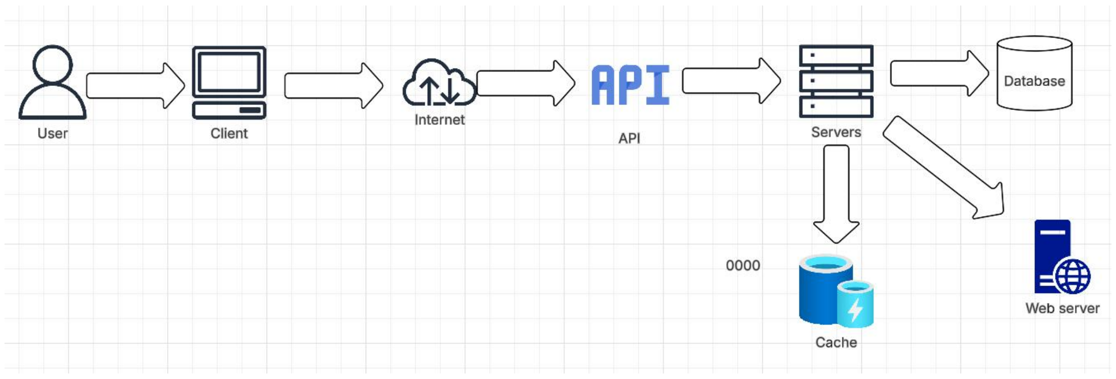
If we remove some features or differentiating features from Instagram, then Instagram might behave like WhatsApp

What two components working together mean?
A user is a person



User sends request to API then we have to provide a valid response
There are various kinds of requests: POST, GET, PUT / PATCH, DELETE
POST request is made when user is trying to add some data
PUT / PATCH request --> User is trying to update some data
GET --> Get some data
DELETE --> Delete some data. Delete requests are built to delete something like deleting photos from application

These are the functionality. API requests are going to our application code, in the web server.
Cache is a temporary kind of memory. There is a Database to save our data also. Our server could reach out to other web servers also



Basic architecture how a basic application works

What's cache?
Lets consider Amazon, we have users logging in to see their order history
I see my order history again and again
My application can process the request, it goes through API, then ApplicationCode on the server, then it needs to call the Database to see my order. This a heavy and costly operation to call DB each time. So we add a layer to cut down this heavy operation and to improve response time. There is an additional request to call our database. To cut Database requests, we extend this database

information and pass it to Cache. Cache is temporary storage of information. It will store our information temporarily so that Database will no longer have to process similar/repetitive requests. Say if we get the same request again from User, instead of calling the database, we get the information from the temporary store (Cache) and return to User. We cannot have all the data in the cache. Advantages of cache are we get data faster, it is more efficient but the issue is outdation in cache. Say if user is trying to get the Order History, Cache could have outdated information. So whenever we have new information, yes it will go update the database, with that we got to update cache also.

Student example

Student --> GetStudent API --> we will have some data in return
There are different formats of data --> XML (Extensible Markup Language), JSON (Javascript Object Notation), Toon (Token-Oriented Object Notation)

XML
```
<student>
	<id> 1 </id>
	<name> ABC </name>
</student>
```

We shifted to JSON from XML for the following reasons
1. XML is hard to read
2. It gets heavy with more data
If we have multiple fields, XML gets very heavy with all the extra tags

JSON --> it works in Key-Value pair

```
{
	“id”: 1,
	“name”: “ABC”
}
```

With JSON also, kind of becomes heavy. Do we have any other tool better than JSON?

How’s Toon better than JSON?
It’s readable, so it is valid for us to use
Then it takes less amount of tokens to process that particular information

In Toon, they have created a bunch of rules over which tokens are generated. One character is not one token, one word is also not one token. If I write a line, they are dividing my line into particular sections and they are going to assign tokens
LinkedIn is flooded with tokens
JSON
```
[	{“id”: 1, “name”: “ABC”},
	{“id”: 2, “name”: “XYZ”}    ]
```

These two combined are entire information or data, which are trying to exchange

TOON:
```
student[2]: {id, name}:
	1, ABC
	2, XYZ
```

Toon captured the keys and it avoided the redundancy of calling the same keys again and again
Toon removes redundant key information

Save tokens, data which we are processing is easily readable.

When we add logs, we sometimes have to process the entire object. that's why we need the format of the object to be easily readable.
APIs are the endpoints through which you send your requests to web server then it provides a response code and responses
Response code:
404 --> User is trying to access some page, which is not present in our system
201 --> Associated with creating of an object. Whenever an object is successfully created, we respond with a 201
200 --> Status Ok response, when the systems are working fine
500 --> returned when we have some or other issue in our Backend code, respresents a Backend issue

Postman is an application, through which we could send requests to APIs. We can test the Backend application using Postman. If there are issues, they could get to know earlier.

What could go wrong to disrupt the system?
We have a single point of failure. Do we have any perfect system in the world? No, system in this entire world is perfect. Every system design will have their downfall.
In the above-design, if the web-server goes down, client will not be able to access our ApplicationCode, in that case, our application will go down. However, if cache goes down, there is not much impact on our system. There wont be any data-loss, but our system response time will be increased. Only cache failure is tolerable, we cannot afford our web-server to go down nor database to go down clearly. Say if some of the services of Amazon go down for even a few min, there will be a loss of millions of dollars. Our application cannot have those kind of failures. So we should know what are the faults in our system first. For example, for database, which one to choose? SQL or NoSQL? Which one will suit our application better?

Two types of applications: Compute-intensive and Data-intensive
Compute-intensive:
Lets say we want to create a Gaming application
BGMI, CounterStrike, GTA, RDR2
All of these games, they want good user-experience with moving objects, waves in water, shadow from sun, gravity etc. They are compute-intensive applications, we cant wait for something to react, that means we got to put our focus more on compute resources. We have to make realistic changes where computation is more important than storing of data.
Data-intensive:
Meta, Amazon, Netflix, Instagram, YouTube --> we are not rendering anything real-time in the Frontend, instead it involves massive rendering, capturing of data, and storing of data. On Netflix, say you pause a movie, then close that and if you come back, the movie starts from that exact point where you had paused. That means, they have stored that user-pause data somewhere.

Lets say, we are building WhatsApp
Features: one feature Chat feature --> Someone can type and send messages, we can see our messages then we can type our own messages.

What could be the utmost requirements?
Functional (F) and Non-functional (N) requirements
Functional is like feature of the application, Security, Latency are not functional therefore.
Any application, if you feel something is a feature then it is a Functional requirement
Security (N)
Data-intensive application
End-to-end encryption (N)
Latency (N)
Synchronous (F)
Compliance (N)

Data integrity (No data loss) (N)
Good UI (F) --> Features is functional
Availability of characters and emotions (F)
Reliability (N)
Performance (N)
Data storage (F)
User should be able to write their message (F)
Android / iOS platforms (F)
User should be able to see messages (F)
Blue tick sent and read should be real-time (F)
Delete chat (F)

For application, we have some functional and non-functional requirements

Homework: Find an application then identify Functional and Non-functional requirements of the application.