Full stack development

Back-end protocols

1. Communication Protocols

How systems talk/communicate with each other

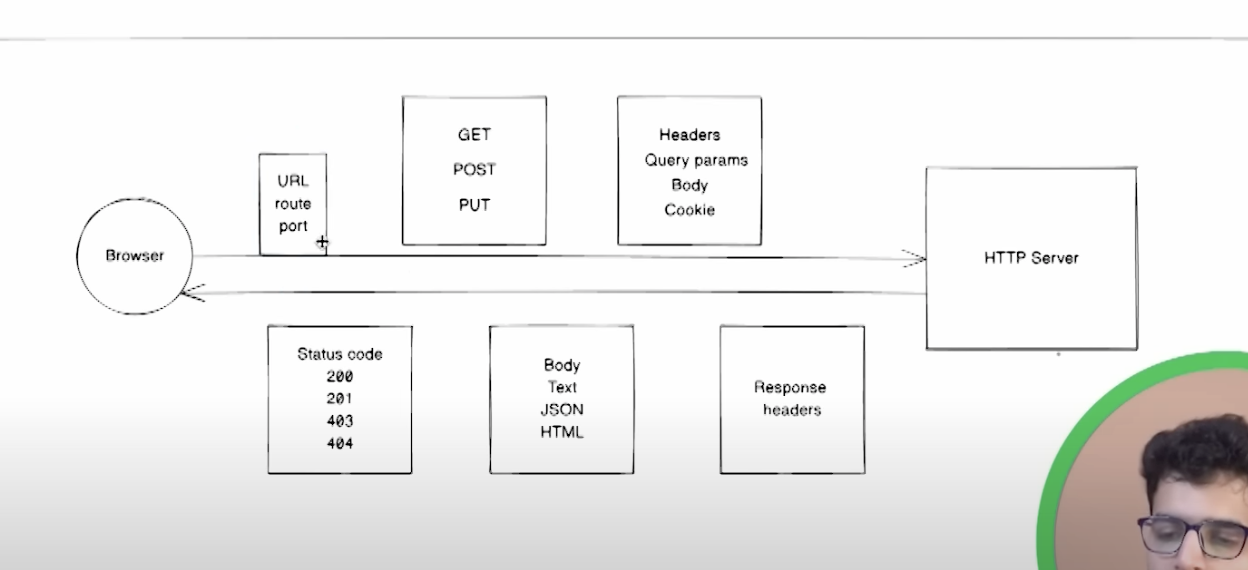
**Transport Layer:** TCP and UDP: packets are thrown and no acknowledgement, if they are received or not. TCP request sends request until an acknowledgement is received

**Application Layer:** HTTP server is a communication protocol browsers use to interact with servers.

Assignment: Build an HTTP server from scratch.

1. Databases: (MySQL, Postgres, MongoDB, Firebase)
2. Messaging Bus/Pub sub: An application that has basic authentication, we can hit an HTTP server which will talk to DB and send a response. In a complicated app, where you want to send an alert before logging, that a new login happened, you should handle it with asynchronous request, which is a better option. Here sending an email or any other verification is important, but not that immediately important. You can login, but email may come in 5-10 secs. So we offload these tasks that need to reach the end user in a while to usually other backend processes. These processes are normally other servers that need to talk to the backened and not the browser. So frontend, sends a request to backend and backend sends another request to your email server which has the responsibility to emailing the other user. This is not done at the backend server, but it isn’t important and that will make it a monolithic architecture, which is not desired. We are following a microservcies architecture, which means there are different services that are doing different tasks.These are also async tasks that can happen in their own time, otherwise it will block the backend server, specifically if email service is down. This is where a messaging bus is introduced. As the name suggests, it is like a messaging queue. You have a server which contains this queue, and your backend server can simply put a message in this queue, and your email server can pick the request from here. Even if the email server is down, it will eventually come up and pick these messages for processing. Now what if the messaging server is down? It should not happen, so you need to have a reliable backend server, or in case it is down then you either invalidate the request or store it in db, that this message gets missed and needs to be done. A lot of times you even have an auth server sitting between front end and backend or maybe even a load balancer between auth server and front end.

Assignment: study pagination



**Creating a backend server In Node.js**

Verify that you have Node

Assignment: Add an object store in the application