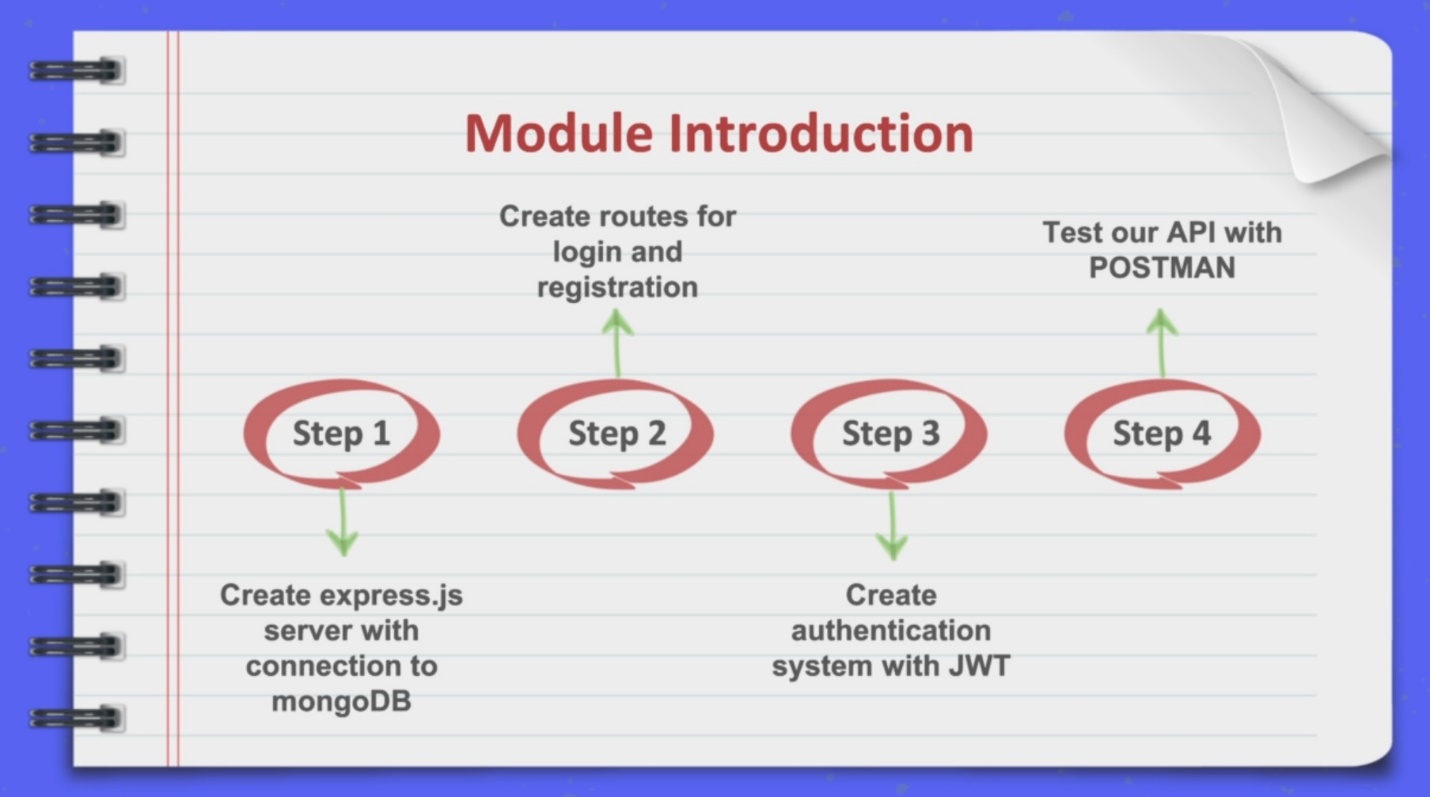
**MERN stack Video chat app**



**Step1:** We will be creating an **Express server** and we will also connect that to MongoDB and we’ll be storing all of the information about the users and all of the messages which will be created and on everything in our application in database and will be going for the **MongoDB in cloud**.

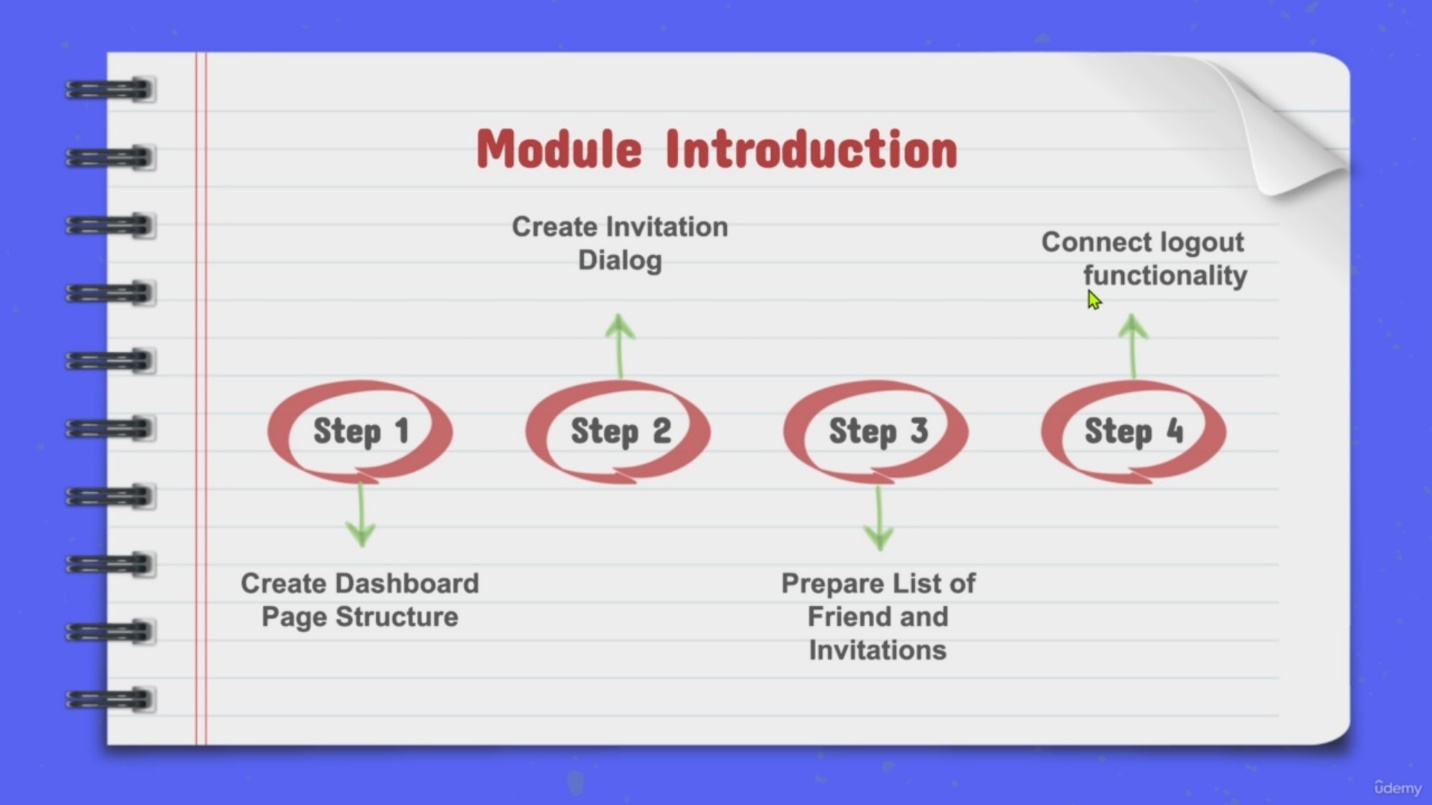
**Step2:** We’ll take care about creating **routes** for login and registration because first thing which we will be creating when we start creating our React app will be **a form**, which will allow us to **login or register**.

**Step3:** We’ll create the **authentication system with JWT token** that if the user will login or register successfully, the token will be returned to client site app and client will be able to access the **protected resources** at the server side.

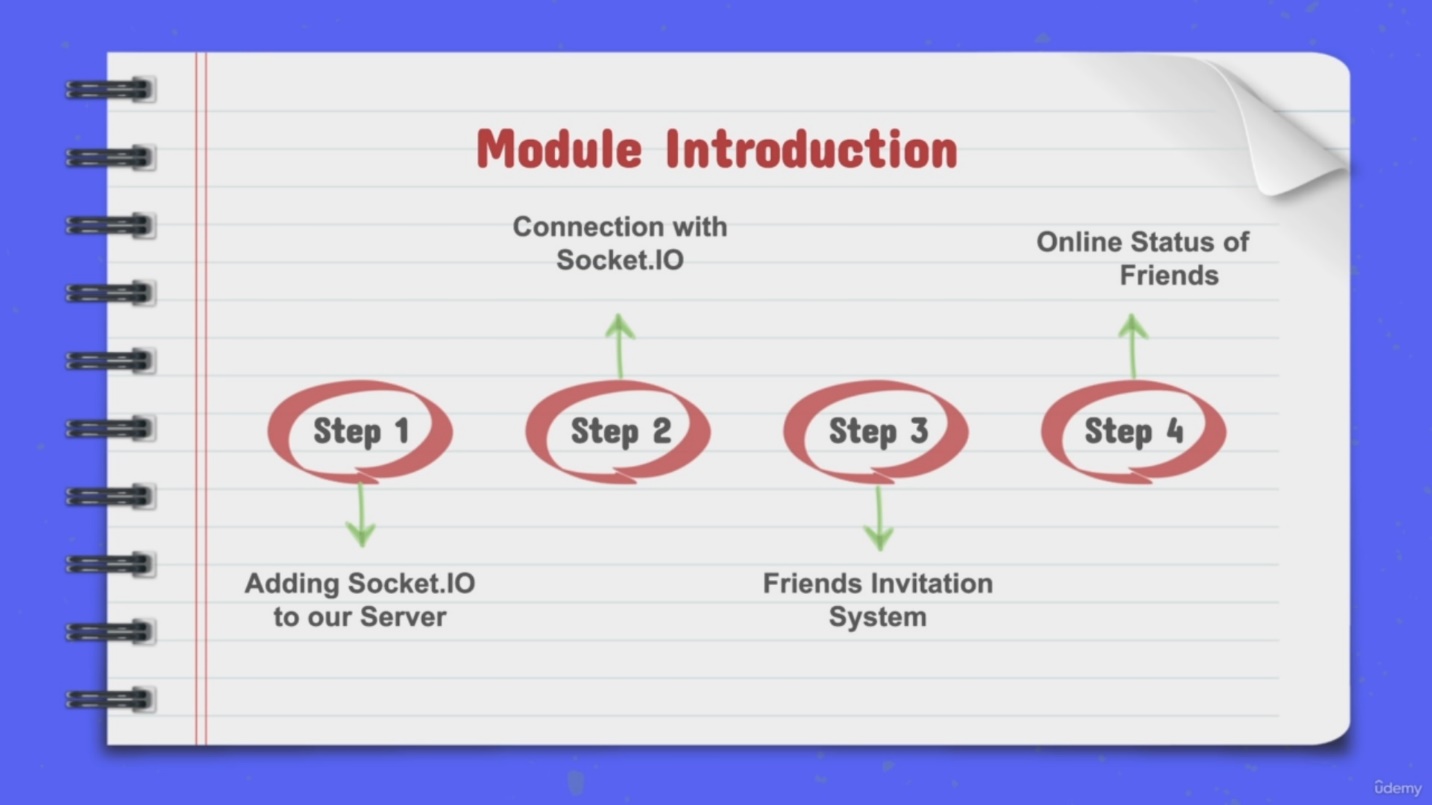
**Step4:** We’ll be **testing** our API with Postman.

/\*\*\*\*/

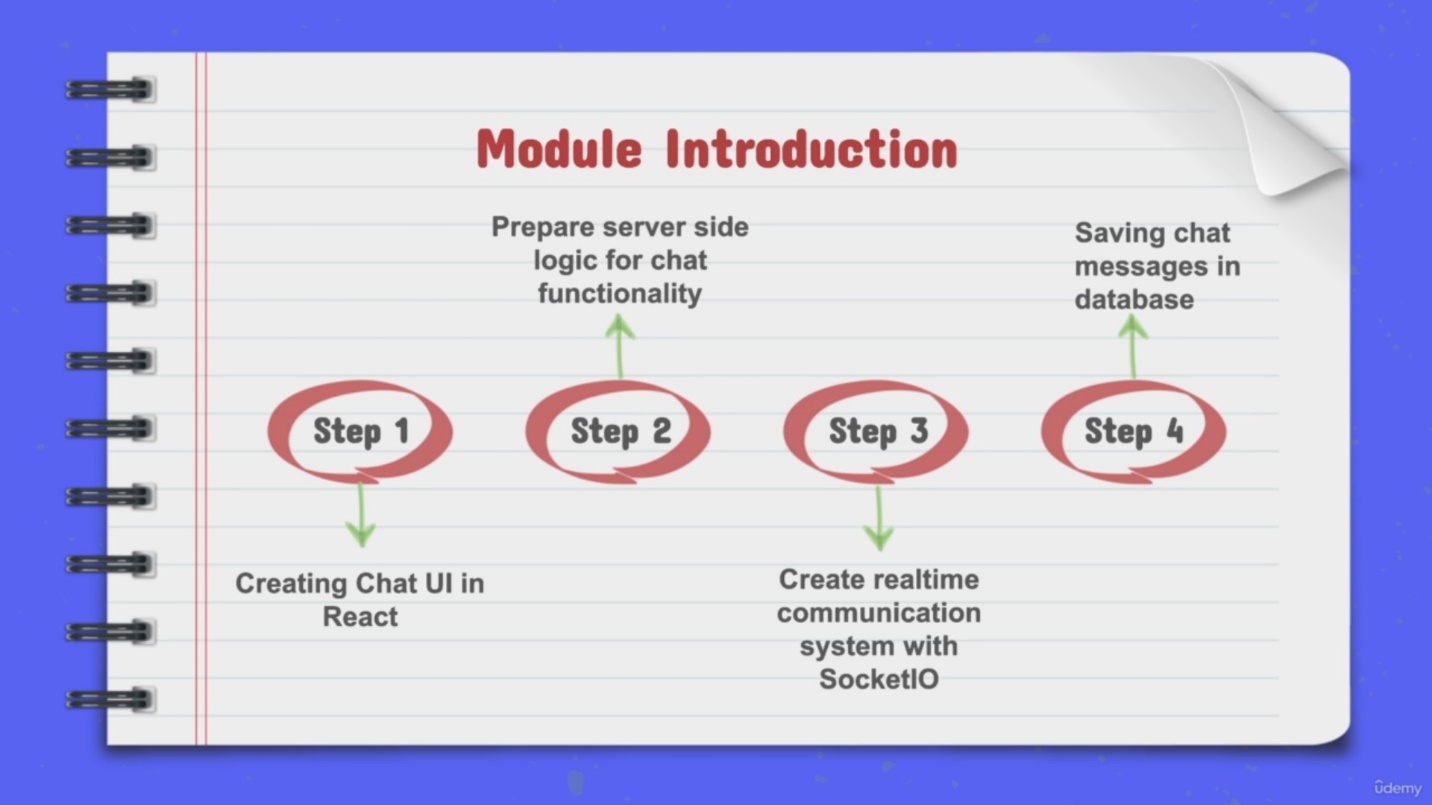
Module 4: Dashboard



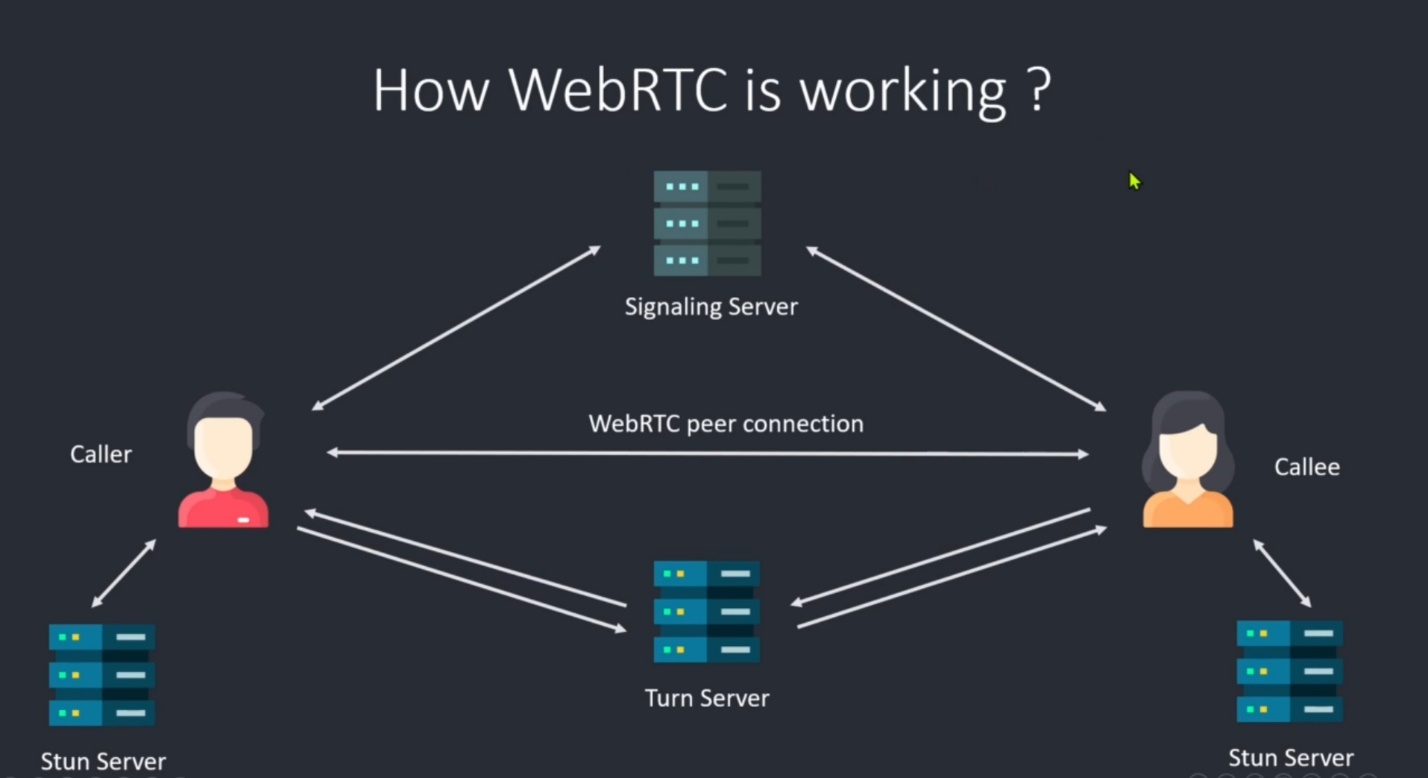
Module 5: Socket.io connection



Module 6:



Module 7: WebRTC:



* WebRTC (Web Real Time Communication) is an open-source project that enables peer-to-peer communication between browsers or the applications (mobile). In other words, WebRTC allows us to exchange any kind of media through the web (such as video, audio and data) without any required plugin or framework.
* Applications which are using WebRTC:
  + Google Meet and Google Hangouts
  + Facebook Messanger
  + Discord
* To establish direct connection between 2 users we need a signaling server.
* Signaling server does not do anything that is WebRTC-specific. Signaling server helps to exchange necessary information which are required to establish direct connection between users. For signaling we can use whatever we like, from webSocket to XMLHttpRequest.
* Before the webRTC peer connection is established we need to exchange some data, this data would be related to our internet connection details and the codecs we are using, information about our browser, etc. in our project we have used webSockets for this purpose.
* STUN (Session Traversal Utilities for NAT) that allows clients to discover their public IP address and the type of NAT they are behind. This information is used to establish the media connection.
* In 15-20% cases STUN server will fail and to establish connection between the peers we will need TURN server.
* TURN server is Traversal Using Relay NAT, and it is a protocol for relaying network traffic. TURN server will be used if STUN server will fail. It is used as an assist to establish connection between the peers.
* TURN servers are not public because of the costs which they can generate because of the traffic which is going through them.
* Before establishing the connection between peers, we need to exchange some data, the first one is the SDP (Session Description Protocol).
* The Session Description Protocol is a format to describing multimedia communication sessions for the purposes of session announcement and session invitation.
* It does not deliver the media data but is used for negotiation between peers of various audio and video codecs, source address timing information of audio and video.
* So, the working of webRTC in brief can be explained like this:
  + Caller takes his internet connection details from STUN Server and along with his SDP information creates an webRTC offer.
  + He sends this offer to the Callee via signaling server.
  + Callee sees the webRTC offer along with the internet and SDP details of Caller.
  + Callee prepares her answer to the offer. If she wants to establish a connection with the caller then she prepares answer with her SDP and internet details, which she takes from her STUN server and sends back to the Caller through the signaling server.
  + So both users once connected will know each other’s SDP as well as internet details.
  + Exchanging the SDP details is the first step when creating the direct connection using webRTC. The second step is to share internet connection details known as ICE candidates details which we get from STUN server.
* Exchanging information about the media (discussed above in offer/answer and SDP), peers must exchange information about the network connection. This is known as an ICE candidate and details the available methods the peer is able to communicate (directly or through a TURN server). Typically, each peer will propose its best candidates first, making their way down the line toward their worse candidates. Ideally, candidates are UDP (since it’s faster, and media streams are able to recover from interruptions relatively easily), but the ICE standard does allow TCP candidates as well.
* How to establish connection between Peers:

