Q1. Write brief about authorization and CORS.

The browser's same-origin policy blocks reading a resource from a different origin. This mechanism stops a malicious site from reading another site's data, but it also prevents legitimate uses. What if you wanted to get weather data from another country?

In a modern web application, an application often wants to get resources from a different origin. For example, you want to retrieve JSON data from a different domain or load images from another site into a element.

In other words, there are **public resources** that should be available for anyone to read, but the same-origin policy blocks that. Developers have used work-arounds such as JSONP, but **Cross-Origin Resource Sharing (CORS)** fixes this in a standard way.

Enabling **CORS** lets the server tell the browser it's permitted to use an additional origin.

Step 1: client (browser) request

When the browser is making a cross-origin request, the browser adds an Origin header with the current origin (scheme, host, and port).

Step 2: server response

On the server side, when a server sees this header, and wants to allow access, it needs to add an Access-Control-Allow-Origin header to the response specifying the requesting origin (or \* to allow any origin.)

Step 3: browser receives response

When the browser sees this response with an appropriate Access-Control-Allow-Origin header, the browser allows the response data to be shared with the client site.