JWT implementation:

1. **Getting a Token**:

- When a user logs in, the server creates a token containing important details about the user, like their ID, username, and role.

- The server signs the token to make sure it's authentic and can't be tampered with.

2. **Storing the Token**:

- The token is stored on the user's device, usually in the browser's memory or in a special cookie.

3. **Sending the Token**:

- When the user wants to access something that requires logging in, they send the token along with their request. the client includes the JWT in the Authorization header of the HTTP request.

The JWT is typically prefixed with the word "Bearer" (e.g., Authorization: Bearer <token>).

4. **Checking the Token**:

- The server checks if the token is real by verifying the signature. If it's real, the server uses the user's info from the token to see if they're allowed to do what they're asking for.

5. **Token Expiry**:

- Tokens have an expiration date. Once they expire, the user has to log in again to get a new one.

6. **Revocation and Refresh Tokens:

- In some scenarios, JWTs can't be invalidated once they are issued. To get this, a combination of short-lived JWTs and refresh tokens can be used.

-Refresh tokens are long-lived tokens used to obtain new JWTs after they expire without requiring the user to re-enter their credentials.

Cookies are like little notes that websites leave in your browser to remember things about you.

They can store information such as your login status, language preference, and items in your shopping cart.

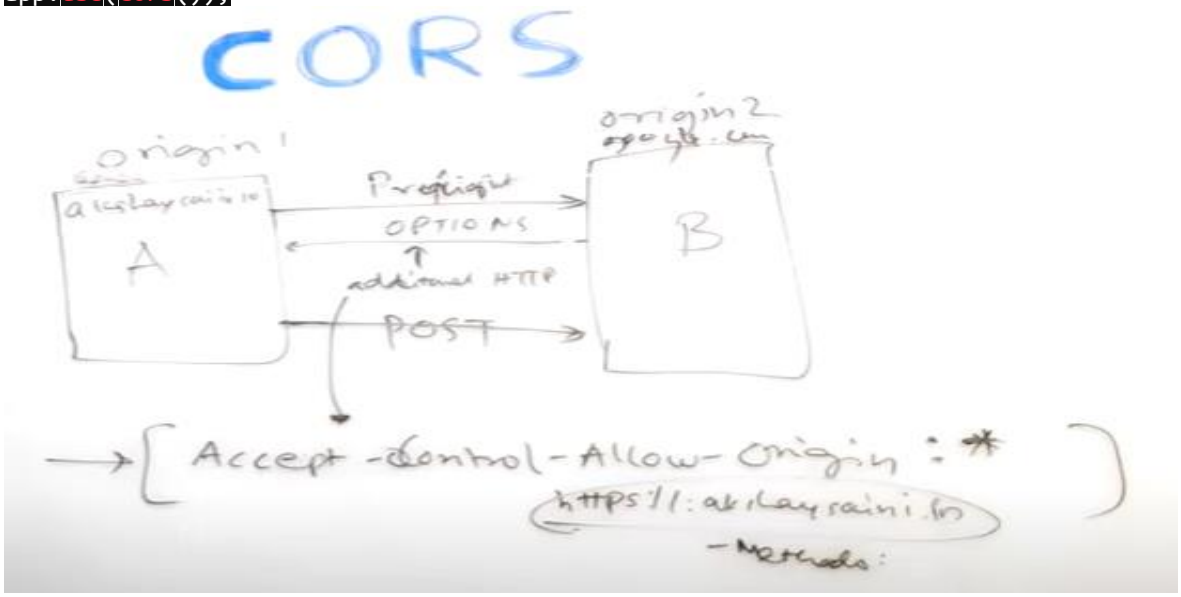
> **Cookies** have a small storage capacity limit of typically 4KB per domain. This limit is imposed by browser specifications.
>
> **Local Storage/session storage** provides a much larger storage capacity, typically around 5MB per domain. This makes Local Storage suitable for storing larger amounts of data compared to cookies.
>
> **Cookies** have an expiration date and time, and they can be set to persist for a specific duration or be session-only (deleted when the browser is closed).
>
> **Local Storage** data persists indefinitely until explicitly deleted by the user or cleared by the application.

Session Storage: Data stored in Session Storage persists only for the duration of the user's session. Once the user closes the browser tab or window, the data stored in Session Storage is cleared.

CORS:

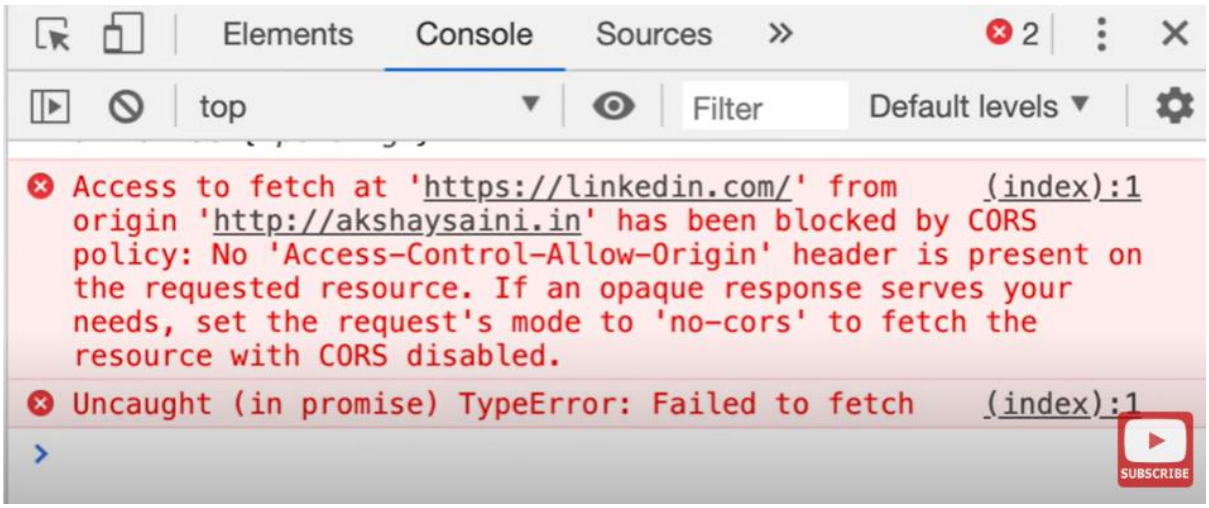
CORS (Cross-Origin Resource Sharing) is a mechanism that enables web servers to specify which origins are permitted to access the resources of a web application via cross-origin requests(When a web application hosted on one domain (origin) makes a request to access resources from another domain, sub domain, or PORT, protocol). In Node.js applications, CORS can be implemented using middleware to define the access control policies.

```
const cors = require('cors');
// Enable CORS for all routes
app.use(cors());
```



preflight request is an HTTP request made by a browser as part of the Cross-Origin Resource Sharing (CORS) mechanism. It's sent automatically by the browser when a web application tries to make a cross-origin request with certain methods or headers

CORS Errors:



****Scenario: Building a Blogging Platform****

You've been tasked with building a simple blogging platform using the MERN stack. Users should be able to sign up, create, edit, and delete blog posts, and view posts created by other users. Additionally, users should be able to leave comments on blog posts.

1. **Backend (Node.js with Express and MongoDB):**

- Use Express.js to handle routing and middleware.

- Define MongoDB schemas for users, blog posts, and comments.

- Implement authentication and authorization using JWT (JSON Web Tokens) for user registration, login, and session management.

- Create API endpoints for CRUD operations on users, blog posts, and comments.

- Implement validation and error handling for user input and API requests.

2. **Frontend (React.js):**

- Design a user-friendly interface for signing up, logging in, creating, editing, and deleting blog posts, and leaving comments.

- Use React Router for client-side routing.

- Implement forms for user authentication and blog post/comment creation/editing.

- Fetch data from the backend API using Axios or Fetch API and update the UI accordingly.

- Implement features like pagination, sorting, and filtering for displaying blog posts and comments.

3. **Database (MongoDB):**

- Set up MongoDB to store user data, blog posts, and comments.

- Define schemas for users, blog posts, and comments, including relationships between them.

- Ensure data integrity and consistency by properly structuring database indexes and using appropriate validation rules.

4. **Security and Performance:**

- Implement authentication and authorization mechanisms to secure the application and protect user data.

- Use HTTPS to encrypt data transmitted between the client and server.

- Optimize database queries and API endpoints for improved performance and scalability.

- Implement caching mechanisms (e.g., Redis) to reduce response times and server load.

5. **Testing and Deployment:**

- Write unit tests and integration tests to ensure the reliability and functionality of the application.

- Deploy the application to a production environment using platforms like Heroku, AWS.

Sharding:

Sharding in MongoDB is a technique for horizontally partitioning data across multiple servers to improve scalability and performance. In short, it involves distributing data across multiple machines in a cluster, called shards, based on a shard key.