Authentication

Authorization

Express

hardik agnihotri

Authentication and Authorization
Are two essential concepts in web development that play a crucial role in securing applications and controlling access to resources.

What is Authentication? Authentication is the process of verifying the identity of a user or a system. It ensures that the user or system is who they claim to be. In simpler terms, authentication is like showing your ID card to enter a restricted area – it confirms your identity and grants you access. In web applications, users typically authenticate themselves using a username and password or other methods like social logins or fingerprint recognition.

What is Authorization? Authorization, on the other hand, is the process of determining what actions or resources a user or system is allowed to access after they have been authenticated. In our ID card analogy, authorization would be the permissions granted to you based on your identity. For example, some users might have access to read-only information, while others might have privileges to edit or delete data.

hardik agnihotri

Why is Authentication and Authorization Important?

Authentication and authorization are vital for protecting sensitive information and controlling user access to various parts of a web application. Without proper authentication, unauthorized users could gain access to secure data or perform actions they shouldn't be allowed to. Similarly, without authorization, even authenticated users could misuse their access rights, leading to security breaches and data manipulation.

Types of Authentication Mechanisms

Username and Password: Most common method where users enter a username and a secret password to prove their identity. **Social Logins**: Users authenticate using their existing accounts on social platforms like Google, Facebook, or Twitter. **Token-based Authentication**: It involves issuing a token (e.g., JSON Web Token or JWT) after successful login, which is then used for subsequent requests. **Multi-Factor Authentication (MFA)**: Requires users to provide multiple forms of identification, such as a password and a one-time code sent to their phone.

Types of Authorization Mechanisms

Role-Based Access Control (RBAC): Users are assigned roles (e.g., admin, user) that determine their access rights.
Attribute-Based Access Control (ABAC): Access is based on attributes like user attributes, time of access, or location.
Rule-Based Access Control: Access is determined by predefined rules set by administrators.

hardik agnihotri

Authentication

1.**User Identification:** When users sign up for a web application, they need a unique identifier. One option is to use usernames, which are user-specific and chosen by the users themselves. The other option is to use emails, which are unique to each user and serve as a convenient way to identify them. The choice depends on the application's specific needs and the desired user experience. Usernames might provide a sense of individuality, while emails can simplify communication and password recovery processes.

2.**Password Security**: Storing passwords securely is crucial to prevent unauthorized access to user accounts. Instead of storing plain text passwords, they should be hashed using cryptographic algorithms like bcrypt or SHA-256. Hashing converts the password into a fixed-length string of characters, making it challenging for attackers to retrieve the original password. To add an extra layer of security, a unique random value called a salt can be added before hashing the password. Salting ensures that even users with the same password will have different hashed values, making it harder for attackers to use precomputed tables (rainbow tables) for password cracking.

3.**JSON Web Tokens (JWT)** for Stateless Authentication: JWTs are a popular method for implementing stateless authentication. When a user successfully logs in, the server issues a JWT, which is then sent to the client. The client includes the JWT in subsequent requests as an Authorization header. The server verifies the JWT's signature to ensure its authenticity and extracts the user information from it. Since JWTs store all necessary information (such as user ID and roles) in the token itself, the server doesn't need to maintain session data, making it stateless and scalable.

4.**OAuth 2.0** for Third-party Authentication: OAuth 2.0 is widely used for third-party authentication in web applications. It allows users to log in using their existing accounts from popular platforms like Google, Facebook, or GitHub. In the OAuth 2.0 flow, the user is redirected to the service provider's login page, and upon successful authentication, the user is redirected back to the application with an access token. This token is then used to authenticate the user with the application, streamlining user registration and login processes while ensuring the security of user credentials.

5.**Authentication Middleware** in Express: Express.js provides a straightforward way to implement authentication using middleware functions. Middleware functions are executed before the main route handlers and can perform tasks like authentication checks and user identification. In an Express application, authentication middleware can be used to verify JWTs, check user roles, and protect specific routes from unauthorized access. By adding authentication middleware to routes, developers can enforce authentication and authorization rules efficiently and modularly, enhancing the application's security and scalability.

hardik agnihotri

Authorization

Understanding Roles and Permissions Authorization is the process of determining what actions a user is allowed to perform within an application or system. Roles and permissions play a vital role in authorization. A role is a predefined set of permissions that grant access to certain features or resources in the application. For example, roles can be "admin," "user," or "guest." Permissions, on the other hand, define specific actions that a user can perform, such as "read," "write," "delete," etc. Roles and permissions work together to control the level of access each user has within the application.

Role-Based Access Control (RBAC) Role-Based Access Control (RBAC) is a widely used approach to manage authorization in applications. RBAC assigns roles to users based on their responsibilities or job functions. Each role is associated with a set of permissions that define what actions the users with that role can perform. This approach simplifies the management of access control by grouping users with similar access needs under the same role.

Attribute-Based Access Control (ABAC) Attribute-Based Access Control (ABAC) is a more fine-grained approach to authorization. It uses attributes or characteristics of the user, resource, and environment to determine access rights. For example, a user's role, department, location, and time of access can be used as attributes to define access control policies. ABAC provides more flexibility in defining access rules and is suitable for complex authorization scenarios.

Implementing Authorization Middleware in Express In Express, we can implement authorization middleware to protect certain routes or endpoints from unauthorized access. This middleware checks whether the user making the request has the necessary permissions or role to access the requested resource. If the user is authorized, the middleware allows the request to proceed; otherwise, it returns a 403 Forbidden or 401 Unauthorized status code.

To implement authorization middleware, we first need to define roles and permissions for our application. We can store this information in a database or a configuration file. When a user makes a request to a protected route, the middleware fetches the user's role and checks if it has the required permission to access the resource. If the user's role matches the required role or has the necessary permission, the request is authorized; otherwise, the user is denied access. For RBAC, the middleware can check if the user's role is included in the list of roles allowed to access the resource. For ABAC, the middleware evaluates the attributes of the user and the resource against the defined access control policies.

hardik agnihotri

Implementing Simple Authentication & Authorization Server

SET-UP HTTP SERVER USING EXPRESSJS

```
npm init -y
npm install express
```

```
const express = require("express");
const app = express();

app.use(express.json());

app.post("/", (req, res) => {
      res.send("HTTP Server Started");
  });

app.listen(3000, () => {
    console.log("Server is listening on port 3000");
  });
```

hardik agnihotri

GENERATE JWT TOKEN

```
npm install jsonwebtoken
```

```
const jwt = require("jsonwebtoken");

const secretKey = "superS3cr3t1";
// replace this with your own secret key

// Generates a JWT token for a given user
const generateJwt = (user) => {
  const payload = { username: user.username };
  return jwt.sign(payload, secretKey, { expiresIn: "1h" });
};
```

- It takes a user object as input.
- Extracts the username from the user object.
- Uses the jwt.sign function to create a token with the username as the payload.
- Uses a secret key (shared between server and client) to sign the token for security.
- Sets an expiration time of 1 hour for the token.
- Returns the generated token.

hardik agnihotri

AUTHENTICATION

```
// Endpoint for user registration
app.post("/users/signup", (req, res) => {
  const user = req.body;
  const existingUser = USERS.find((u) => u.username === user.username);
  if (existingUser) {
    res.status(403).json({ message: "User already exists" });
  } else {
    USERS.push(user);
    const token = generateJwt(user);
    res.json({ message: "User created successfully", token });
  }
  console.log(USERS);
});
```

- It checks if a user with the same username already exists.
- If the user exists, it returns an error.
- If not, it adds the user to the database.
- It generates a token for the user (JWT) to remember them.
- Sends a success message with the token.

hardik agnihotri

AUTHORIZATION

```
// Endpoint for user login
app.post("/users/login", (req, res) => {
  const { username, password } = req.headers;
  const user = USERS.find(
    (u) => u.username === username && u.password === password
  );
  if (user) {
    const token = generateJwt(user);
    res.json({ message: "Logged in successfully", token });
  } else {
    res.status(403).json({ message: "User authentication failed" });
  }
});
```

- It reads the username and password from the request headers.
- It checks if the user exists in the database and if the provided password matches.
- If user and password match, it generates a token (JWT) for the user.
- Sends a success message with the token.
- If user or password is incorrect, it returns an authentication failure error.

hardik agnihotri

Cross-Origin Resource Sharing (CORS)

1. Sharing Stuff Safely: CORS stands for Cross-Origin Resource Sharing. It's like a set of rules that websites follow to safely share things with each other.
2. Websites Talking: Sometimes, websites want to talk to each other to get information or images. CORS helps to make sure this talking is secure.
3. Same Playground Rule: Imagine each website is a kid in a playground. CORS makes sure that kids from different playgrounds can't just come and take stuff from your playground.
4. Permission Slip: Before one website fetches something from another website, it needs a permission slip from that other website. This slip says, "Hey, I promise to use your stuff nicely."
5. Friendly Websites Allowed: Websites can decide who they want to give permission to. If they don't know or trust a website, they won't give the permission slip.
6. Different Domains: When websites have different names (like google.com and facebook.com), they're from different domains. CORS is needed because browsers want to keep things safe between different domains.
7. Browser Rule Enforcer: Your web browser is like a rule enforcer. It checks the permission slip (CORS headers) to make sure websites are playing nicely and following the rules.
8. No Sneaky Business: CORS prevents sneaky websites from taking or changing things on other websites without permission. It's like web security to keep everything fair and safe.

hardik agnihotri