Building the CRM App: User Registration and User Login

Relevel
by Unacademy

Concepts

1. Authentication
2. Authorization
3. User Schema
4. Registration Login API

What is Web Security ?

- A web security gateway protects people and organizations against online threats by monitoring and filtering internet traffic in real-time and blocking traffic deemed suspicious, malicious, or outside of policy.
- Web security, in general, refers to the preventive measures and processes that businesses use to defend themselves from cyber criminals and threats that use the internet. Web security is essential for business continuity as well as the protection of data, users, and enterprises.
- There are many ways to ensure web security is concrete and cannot be penetrated, two of them are Authentication and Authorization

Authentication

- In the context of computer systems, authentication refers to the assurance and validation of a user's identity. Before attempting to access data stored on a network, a user must first establish their identity and obtain authorization to access the data.

- A user must supply unique log-in credentials, such as a user name and password, before connecting onto a network, a practice aimed to safeguard a network against infiltration by hackers.

Authentication

- To authenticate a user's identity before they can access a network, authentication uses various combinations of data, passcodes, QR codes, passwords, passcards, digital signatures, fingerprint, retinal, face, and voice scans.

- A secure web gateway and the implementation of numerous, integrated security safeguards and solutions, such as next-generation firewall and endpoint protection, are frequently used to enable proper authentication.

Authentication

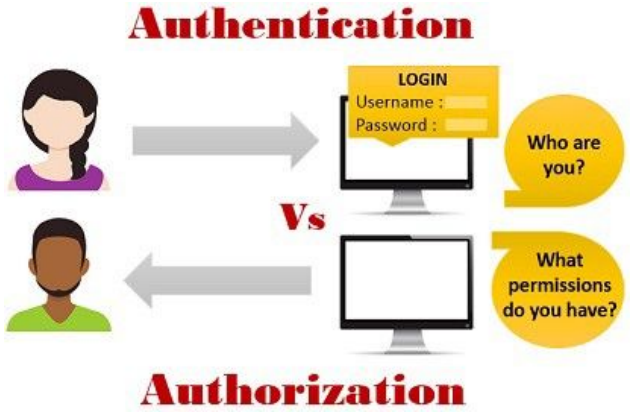
- Allowing users access to systems and apps is now possible thanks to authentication. But wait, there's more! Once the system has identified who the users are, restrictions can be implemented to limit where they can go, what they can do, and what resources they have access to. This is referred to as permission.

  This is where Authorization comes in.

Authorization

- The process by which a server assesses whether a client has permission to use a resource or access a file is known as authorization.

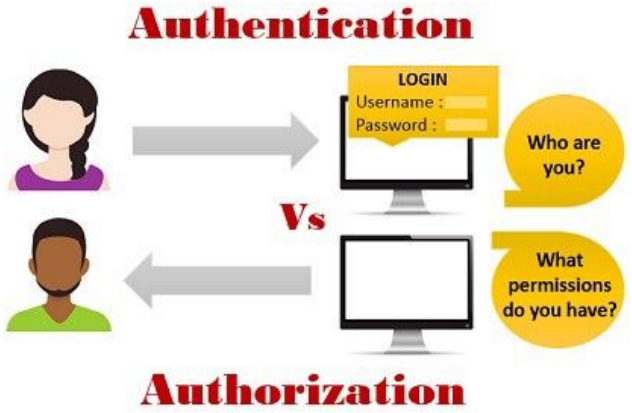
- Authentication is frequently used in conjunction with authorization so that the server knows who the client is who is requesting access.



Authorization

- Authorization may need different types of authentication; passwords may be necessary for certain circumstances but not in others.

- There is no authorization in some circumstances; any user can use a resource or access a file simply by requesting it. The majority of web pages on the Internet do not require any kind of authentication or authorization.



Authorization

- When you wish to restrict viewer access to specific pages, you should utilise authorization. Students at IIT Kharagpur, for example, are not permitted to access specific online pages devoted to professors and administrators. A website's authorization criteria are often defined in the .htaccess file.

- Authorization is crucial because it ensures that users do not have access to more systems and resources than they require. This also allows you to spot when someone is attempting to gain access to something they shouldn't.
- For example, maintaining patient confidentiality by only allowing medical personnel and not administrative people access to patient records.

User Schema

- We will create a file called user.model.js and instantiate mongoose.
- Then we will define a userScheme by using the Schema method of mongoose.
- We will pass the first object and call it *name*.
- It will have two properties, type and required. We will pass String and true as the respective values.

```
const mongoose = require("mongoose");
const userSchema = new mongoose.Schema({
    name: {
        type: String,
        required: true,
    },
```

User Schema

```
userId: {
        type: String,
        required: true,
        unique: true
    },
    password: {
        type: String,
        required: true,
    },

    email: {
        type: String,
        required: true,
        lowercase: true, // it will covert the email into the lower case and then store in the
db,
        minLength: 10,  // anything less than 10 will fail
        unique: true
    },
```

```
createdAt: {
        // I want to default to a new date
        type: Date,
        immutable: true,  // This will ensure the createdAt column is never updated but once in the
start
        default: () => {
            return Date.now();
        }
    },
    updatedAt: {
        type: Date,
        default: () => {
            return Date.now();
        }
    },
```

```
 userType: {
        type: String,
        required: true,
        default: "CUSTOMER"
    },
    userStatus: {
        type: String,
        required: true,
        default: "APPROVED"
    }
```

Finally, we will export the model.

```
module.exports = mongoose.model("User", userSchema);
```

Registration API

```
const User = require("../models/user.model");
const { userTypes } = require("../utils/constants");
const constants = require("../utils/constants");
var bcrypt = require("bcryptjs");
var jwt = require("jsonwebtoken");
const config = require("../configs/auth.config");

exports.signup = async (req, res) => {
    var userStatus = req.body.userSatus;
  if(!req.body.userSatus){
       if(!req.body.userType || req.body.userType==constants.userTypes.customer){
         userStatus = constants.userStatus.approved;
       }else{
         userStatus = constants.userStatus.pending;
       }
    }
```

Registration API

- We will design the controller for the sign up flow. We will create a promise and name it signup, that shall receive two parameters, req & res.

- We will assign the user Status request to a variable of the same name.

- Then we will add a condition, if the userType is customer we will approve the user Status else we will keep it pending.

```
const userObj = {
        name: req.body.name,
        userId: req.body.userId,
        email: req.body.email,
        userType:
req.body.userType,
        password:
bcrypt.hashSync(req.body.password,
8),
        userStatus: userStatus
    }
```

```
try {
        const userCreated = await
User.create(userObj);
        const postResponse = {
            name : userCreated.name,
            userId : userCreated.userId,
            email: userCreated.email,
            userTypes : userCreated.userType,
            userStatus : userCreated.userStatus,
            createdAt : userCreated.createdAt,
            updatedAt : userCreated.updatedAt
        }
        res.status(201).send(postResponse);
    } catch (err) {
        console.log("Some error while saving the
user in db", err.message);
        res.status(500).send({
            message: "Some internal error while
inserting the element"
        })
    }
```

- Using the schema created in the previous lesson, we will send the userObj to MongoDB and pass the response as well.

- We will handle this using the try catch block, where the User module of Mongoose will call the create method, which will receive the userObj as the parameter.
- If the execution fails, then we will handle it using the catch block where depending on the error code we will display messages.

Login API

```
exports.signin = async (req, res)=> {
    const user = await User.findOne({ userId: req.body.userId });
    console.log(user);
    if (user == null) {
        res.status(400).send({
            message: "Failed! Userid doesn't exist!"
        });
        return;
    }

    if(user.userStatus != 'APPROVED'){
        res.status(200).send({
            message : `Can't allow login as user is in statuts : [
${user.userStatus}]`
        })
        return ;
    }
```

Login API

- Now we will create a controller for the signing in the user.
- Similarly, we will create a promise and name it sign in, it will also receive two parameters, req & res.
- Since the goal of this API is to sign in the user, we need to first validate whether the credentials are correct or not, we can do that by fetching the user based on user ID.

- We will be using findOne method of Mongoose which will receive the userId as the parameter and we will assign the return value to a user constant.

- We will put two validations, if the user constant is null we will display a message “Failed! User id doesn’t exist”.

- If the user Status is APPROVED, we will display “Can’t allow login as user is in status” and concatenate user status at the end of the message.

```
var passwordIsValid = bcrypt.compareSync(
        req.body.password,
        user.password
      );
      if (!passwordIsValid) {
        return res.status(401).send({
          accessToken: null,
          message: "Invalid Password!"
        });
      }
      var token = jwt.sign({ id: user.userId }, config.secret, {
        expiresIn: 86400
      });
      res.status(200).send({
        name : user.name,
        userId : user.userId,
        email: user.email,
        userTypes : user.userType,
        userStatus : user.userStatus,
        accessToken : token
```

- Then we will check whether the password matches with what we have in the database.
- We will be using bcrypt's compareSync method for the same, which will require the password from request body and the password from the database.

- We will store the response of compareSync to a variable called passwordIsValid.

- Then we will put up some conditions, if the password is not valid, we will display a message "Invalid Password!".

- If it is valid then we will use the sign method of jwt module and pass the user id, secret key and the duration after which the session will expire as parameters and store the response in a token variable.

- Finally, we will send the response using the res.status(200).send method and we will pass all the properties pertaining to the user

Thank You