Authentication and Authorisation

Created	@May 16, 2022 8:00 PM
Class	
Type	
Materials	
Reviewed	☐

Authentication

So this is a process using which we can uniquely identify users on our applications. It tells about who the user is. The general signup login logout flow is used to authenticate a user.

Authorisation

So this is a process using which we can identify the capabilities of a user, i.e. what a user can do on our application. Example: A seller on flipkart can only add products but we as a normal user cannot.

How to do authentication ?

So there are several ways of doing authentication

- Omniauth (gmail/fb/linkedin login)
- Mobile number based authentication
- Token based authentication / Bearer Authentication

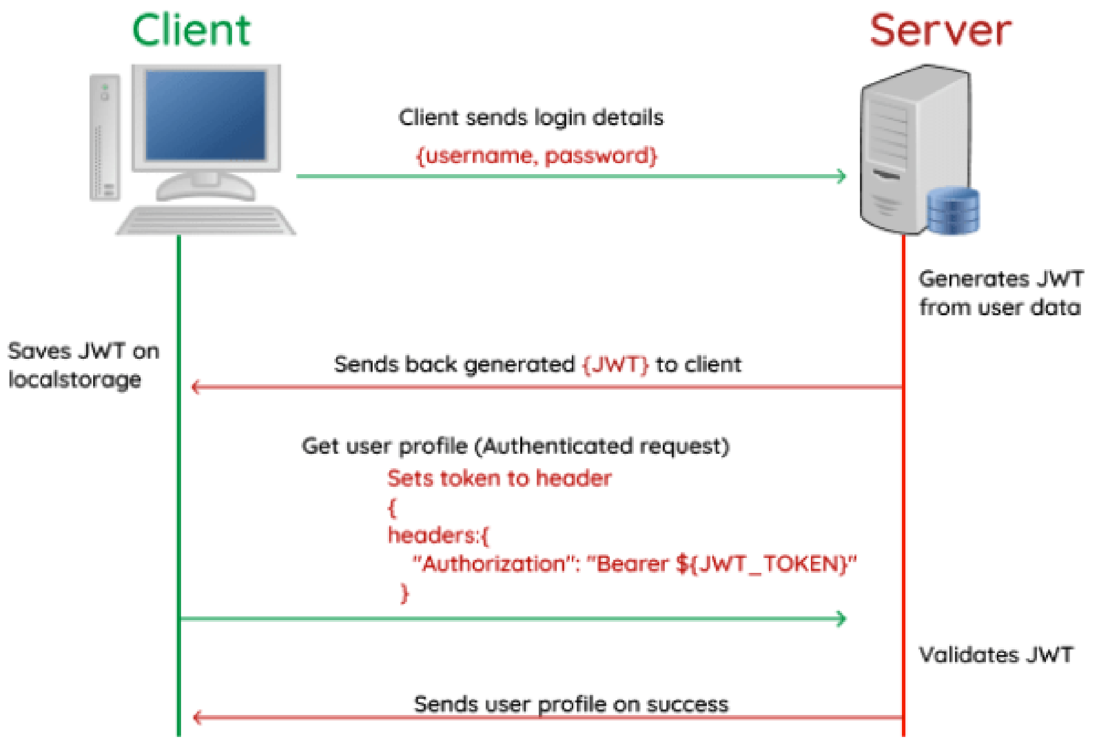
Token based authentication

Whenever a user tries to login/signin into our app, we verify the credentials (email, pass). If the user is a valid registered user then we return them a `JWT` token (`Json Web Token`) . This JWT token is a unique token that will be given to the user. Now whenever the user wants to access any resource from our server in every API call they have to send the token along with the http request. When the server receives the token, it verifies that whether the user is an authenticated user or not ? If not then the server doesn’t process the request other wise the server goes ahead and process it.

This token is not saved on server side, it is the duty of the client to store and send it every time.

If we are not storing the token on the server how do we verify it ? We will figure this out later.

- In order to generate the JWT token, we use the user credentials only.


Client
Server
Client sends login details
{username, password}
Generates JWT from user data
Saves JWT on localstorage
Sends back generated {JWT} to client
Get user profile (Authenticated request)
Sets token to header
{
headers:{
"Authorization": "Bearer ${JWT_TOKEN}"
}
Validates JWT
Sends user profile on success

Token based authentication

- Requirements For authentication
    - We need a user model
    - We need a set of api for registering the user or you can sign up the user. When we will register the user we need to store the new user’s email and password. We don’t want to store password in plain fashion, We should always store then with some encryption.
    - We need a set of api to sign in the user, using which the user will receive a new JWT token.
    - We need a mechanism to verify the token on every authenticated api call.

Creating user model

```
npx sequelize model:generate --name User --attributes email:string,password:string,usernam
e:string
```

This will give us the model and migration files. In the model file we can add few validations.

```
User.init({
    email: {
      type: DataTypes.STRING,
      validate: { // validator for email
        isEmail: true
      }
    },
    password: {
      type: DataTypes.STRING,
      validate: { // validator for email
        len: [5, 10]
      }
    },
    username: DataTypes.STRING
  }
```

Then we will migrate the db

```
npx sequelize db:migrate
```

Setup signup api

auth.controller.js

```
const signup = async (req, res) => {
    console.log(req.body)
    const response = await Auth.signup(req.body);
    return res.json({
        message: 'Successfully created a user',
        success: true,
        code: 200,
        data: response
    });
}
```

auth.service.js

```
const signup = async (data) => {
    const user = await User.create({
        email: data.email,
        password: data.password
    });
    return user;
}
```

auth.routes.js

```
const AuthController = require('../controllers/auth.controller');
const routes = (app) => { // the app parameter is the express app only
    // route -> controller function
    app.post('/ecom/signup', AuthController.signup);
}

module.exports = routes;
```

Hooks in sequelize

Whenever we use sequelize in order to create update delete objects, these operations have a set of functions available based on the lifecycle of the object. For example before object creation or after object creation or before object deletion etc.

These functions are called as Hooks.

We will add a hook for encrypting the password before user creation

```
User.beforeCreate((user) => {
    console.log("User object before encryption", user);
    const encryptedPassword = bcrypt.hashSync(user.password);
    user.password = encryptedPassword;
    console.log("user object after encryption", user);
  });
```

Creation of signin flow

First of all inside the auth service we will create a new function to get a user by email

```
const getUser = async (userEmail) => {
    const user = await User.findOne({
        where: {
            email: userEmail
        }
    });
    return user;
}
```

Whenever the user will try to signin, the user is expected to pass their email and password.

The user will pass password in plain form, but in our database, password is stored in encrypted form. So, we will get the user object from database by email id, in this user object we will get username, email and password all the properties of the stored user. Then we will compare the encrypted password with the password given by user for login. For comparing we will using bcrypt library.