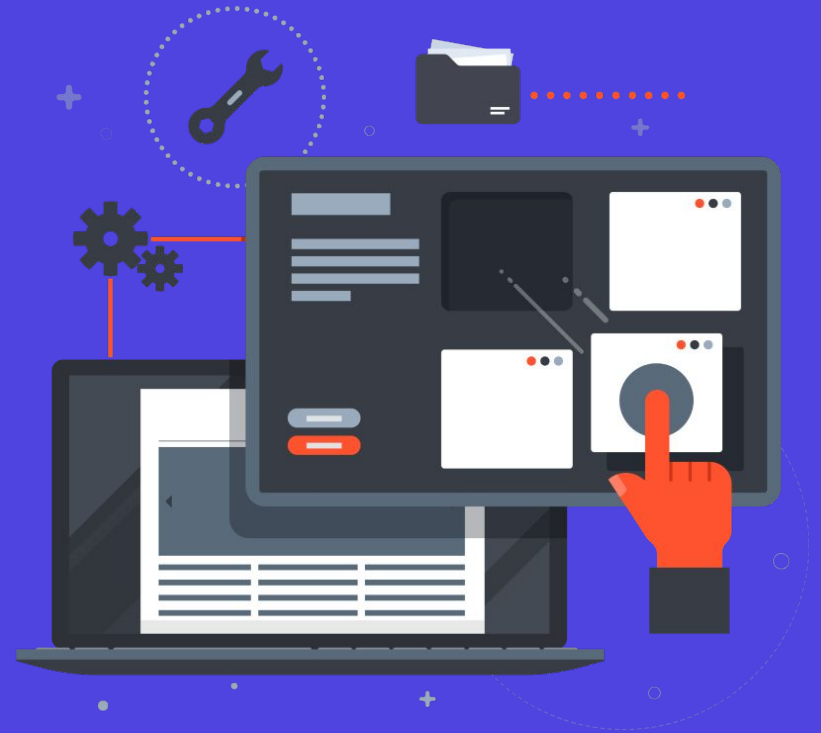


# Security - Authentication and Authorization

**Relevel**  
by Unacademy



# Agenda

- Set up data model for user
- User registration
- Implementation and validation of JWT token
- Login API
- Update password
- Registration of system admin and client

# Feature of Movie Booking Application to be developed in this session

API - Register a customer

API - log in a customer

API - Update the password for a logged-in user

API - Admin approval for the client registration

# Authentication

Authentication represents the user. It recognizes the user and authenticates him to access the application. The user needs to enter a password, one-time password, credentials containing username and password, and so on. There are different ways to perform Authentication. Let's have a look at these -

1. **Password-based authentication** - This is the most common way of authentication. Users use different combinations of letters, numbers, and symbols to create passwords and use them.
2. **Multi-factor authentication** - This method involves multiple stages of authentication for a single user. For example, Captcha tests, facial recognition, and so on.
3. **Biometric authentication** - This involves biological characteristics of a person like a fingerprint scanner, Eye scanner, facial recognition, and so on.
4. **Token-based authentication** - This involves the use of an encrypted string of characters for authentication. Users don't need to enter credentials again and again to authenticate.
5. **Third-party authentication** - This involves a third party that completes the process of authentication and does not have any credentials from the user side for the application. Instead, a third party will authenticate the user and take the input from the user. Users can have profiles on social sites like Google, Facebook, and so on. These platforms provide authentication and serve as a third party.

# Authorization

Authorization represents roles and permission that the user is having to access the application. There can be multiple levels of permissions that can restrict a user to access a specific part of the application. It includes read access, write access, read-write access, and so on. There are different ways to perform Authentication. Let's have a look at these -

**Attribute-Based Access Control** - This method involves the authorization of the user based on some attribute or claim. For example, the age of the user needed for the application can be considered as an attribute.

**Role-Based Access Control** - This method is based on the roles of users and not the user. The role is a collection of permissions. For example, the admin can view a list of users that can be considered as a role.

**Relationship-Based Access Control** - This is based on the relationship between user and resource or some action. For example, a user is a member of the role group related to some resource access.

# JWT Tokens

We are going to use JWT tokens for authentication and authorization purposes. JWT stands for Json Web Token. JWT is a mechanism for user/client authorization in most web apps today. It's extensively used in most web services to securely exchange the client and the server's information.

# Generating JWT

## JWT structure

Here is a sample JWT

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gV2ljayIsIm1hdCI6MTUxNjIzOTAyMn0.m-1TPEuS4dQjy11xdjW3Hw6oaJJ5xmsDV9oQzx1zykY
```

There are three different components separated by ‘.’. The first part is the header, the middle part is the payload and the third component is the signature.

## JWT payload

This contains the data that the server sends back to the client. The data is encoded in Base-64 format.

```
eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gV2ljayIsIm1hdCI6MTUxNjIzOTAyMn0.
```



The above data on decoding becomes the following :-

The server can add as many fields to the payload as it wants.

PAYLOAD: DATA

```
{  
  "sub": "1234567890",  
  "name": "John Wick",  
  "iat": 1516239022  
}
```

# JWT Header

The header is also in a base64 format and has to be decoded on the server.

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9

By decoding the above header we get the following :-

HEADER: ALGORITHM & TOKEN TYPE

```
{  
  "alg": "HS256",  
  "typ": "JWT"  
}
```

The header contains the type of the token and the algorithm used to sign the payload. In the above examples, the algo is HS256.

# JWT Signature

The third component is the signature. The server uses the signature to verify whether the payload has not been tampered with and is the correct JWT that the user is presenting.

Following method is used to validate the signature :-

VERIFY SIGNATURE

```
HMACSHA256(  
  base64UrlEncode(header) + "." +  
  base64UrlEncode(payload),  
  your-256-bit-secret  
) ☐ secret base64 encoded
```

The signature is generated using the secret key, Base-64 encoded header, and payload. The secret key is only with the server.

If the user tampers with the JWT by modifying any of the three fields, then the signature value would change, and verification of JWT would fail.

If signature is not valid, the page would display an invalid signature.

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ  
JzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6Ikpva  
G4gV21jayIsIm1hdCI6MTUxNjIzOTAyMn0.m-  
1TPEuS4dQjy11xdjW3Hw6oaJJ5xmsDV9oQzx1yk  
Y
```

⊗ Invalid Signature

# API - Register a customer

In this API, we will do the signup operation of a customer i.e. register a customer.

Let's have a look at all the steps needed one by one -

[Code Link] -

[https://github.com/Vishwa07dev/mba\\_backend/blob/session4/controllers/auth.controller.js](https://github.com/Vishwa07dev/mba_backend/blob/session4/controllers/auth.controller.js)

# API - Register a customer – STEP 1

We will fetch the user status from the request. If it is not present, we will check userType. If userType is customer, we will set user status as APPROVED, else we will set it as PENDING

```
var userStatus = req.body.userStatus;
if(!req.body.userStatus){
    if(!req.body.userType ||
req.body.userType==constants.userTypes.customer){
        userStatus =
constants.userStatus.approved;
    }else{
        userStatus = constants.userStatus.pending;
    }
}
```

## API - Register a customer – STEP 2

Now, we will capture all the details from the request which are user name, user id, user email, user type, password and user status

```
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
}
```

## API - Register a customer – STEP 3

We will use create function of the User schema to create a new user object inside the database.

```
const userCreated = await User.create(userObj);
```

Once, the user got created, we will send a response having the user information.

```
const postResponse = {  
  name : userCreated.name,  
  userId : userCreated.userId,  
  email: userCreated.email,  
  userTypes : userCreated.userType,  
  userStatus : userCreated.userStatus,  
  createdAt : userCreated.createdAt,  
  updatedAt : userCreated.updatedAt  
}
```



# Request

POST

localhost:8080/mba/api/v1/auth/signup

Params

Authorization

Headers (8)

Body

Pre-request Script

Tests

Settings

☐ none

☐ form-data

☐ x-www-form-urlencoded

☒ raw

☐ binary

☐ GraphQL

JSON

1

2

3

4

5

6

7

8

1

2

3

4

5

6

7

8

```
{
  "name": "Roshan Kumar",
  "userId": "rosh07",
  "email": "rosh@gmail.com",
  "userType": "CUSTOMER",
  "password": "Welcome"
}
```

# Response

Body Cookies Headers (7) Test Results

Status: 201 Created Time: 189 ms Size: 432 B

Pretty Raw Preview Visualize JSON

```
1  {
2    "name": "Roshan Kumar",
3    "userId": "rosh07",
4    "email": "rosh@gmail.com",
5    "userTypes": "CUSTOMER",
6    "userStatus": "APPROVED",
7    "createdAt": "2022-04-10T18:10:43.270Z",
8    "updatedAt": "2022-04-10T18:10:43.270Z"
9  }
```

# API - Login a customer

In this API, we will do the signin operation of a customer i.e. login a customer.

Let's have a look at all the steps needed one by one -

[Code Link] -

[https://github.com/Vishwa07dev/mba\\_backend/blob/session4/controllers/auth.controller.js](https://github.com/Vishwa07dev/mba_backend/blob/session4/controllers/auth.controller.js)

# API - Login a customer – STEP 1

We will fetch user from the database from user id in the request. If the user is not present, we will send message saying “Failed! Userid doesn’t exist!”

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

## API - Login a customer – STEP 2

Next, we will check user status. If user status is not APPROVED, we will send message saying that “Can’t allow login as user is in status - {{userStatus}}”

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

## API - Login a customer – STEP 3

Now, we will validate the password. For this we will use bcrypt library. It is a password hashing function which can be used to save password in encrypted form and we can also compare them from database while login using compareSync() function. If password is not valid, we will send message saying that “Invalid Password!!”

```
var passwordIsValid = bcrypt.compareSync(  
    req.body.password,  
    user.password  
);  
  
if (!passwordIsValid) {  
    return res.status(401).send({  
        accessToken: null,  
        message: "Invalid Password!"  
    });  
}
```

## API - Login a customer – STEP 4

Next, we will use JWT based authentication where we will use sign function and pass user id, secret key and expiration time.

```
var token = jwt.sign({ id: user.userId },  
  config.secret, {  
    expiresIn: 120 // 2 minutes  
  });
```

Secret Key is saved in below path -

[https://github.com/Vishwa07dev/mba\\_backend/blob/session4/configs/auth.config.js](https://github.com/Vishwa07dev/mba_backend/blob/session4/configs/auth.config.js)

```
module.exports = {  
  secret: "Vishwa-Mohan-secret-key"  
};
```

## API - Login a customer – STEP 5

At last, we will send user information in the response

```
res.status(200).send({  
    name : user.name,  
    userId : user.userId,  
    email: user.email,  
    userTypes : user.userType,  
    userStatus : user.userStatus,  
    accessToken : token  
})
```



# Request

POST

localhost:8080/mba/api/v1/auth/signin

Params

Authorization

Headers (8)

Body

Pre-request Script

Tests

Settings

☐ none

☐ form-data

☐ x-www-form-urlencoded

☒ raw

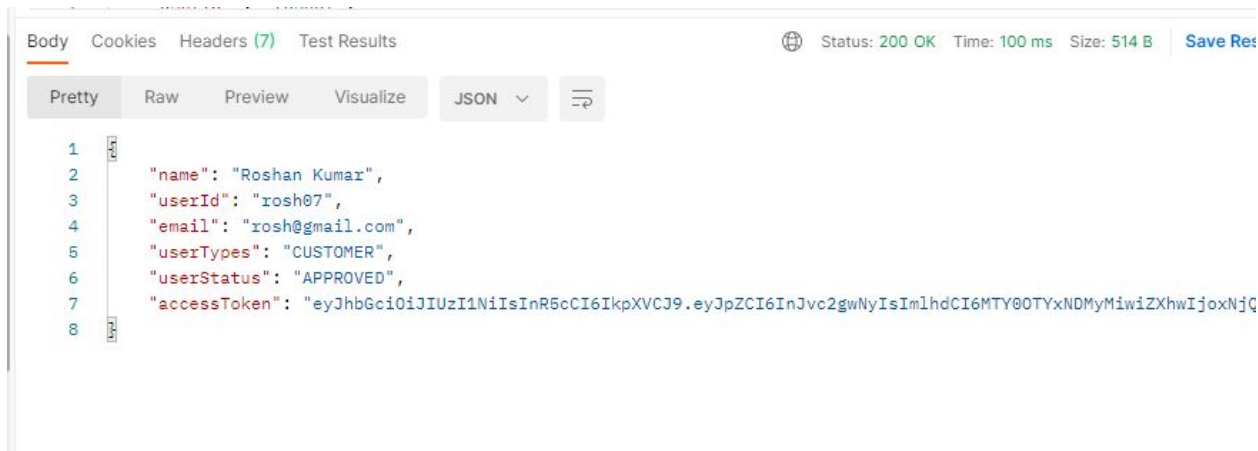
☐ binary

☐ GraphQL

JSON

```
1 {
2   ... "userId" : "rosh07",
3   ... "password" : "Welcome"
4 }
```

# Response



# API - Update the password for a logged-in user

In this API, we will update the password for a logged-in user.

Let's have a look at all the steps needed one by one -

[Code Link] -

[https://github.com/Vishwa07dev/mba\\_backend/blob/session4/controllers/user.controller.js](https://github.com/Vishwa07dev/mba_backend/blob/session4/controllers/user.controller.js)

# API – Verify the token – STEP 1

We will fetch token from headers of the request -> x-access-token

```
let token = req.headers["x-access-token"];
```

- If token is not provided, we will send message saying that “No token provided!!”

```
if (!token) {  
    return res.status(403).send({  
        message: "No token provided!"  
    });  
}
```

Next, we will verify our token using JWT verify function. We will pass token and secret key to verify the token. If verification failed, we will send message saying that “Unauthorized!”

```
jwt.verify(token, config.secret, (err, decoded) => {  
    if (err) {  
        return res.status(401).send({  
            message: "Unauthorized!"  
        });  
    }  
    req.userId = decoded.id;  
    next();  
});
```

## API – Update the password – STEP 2

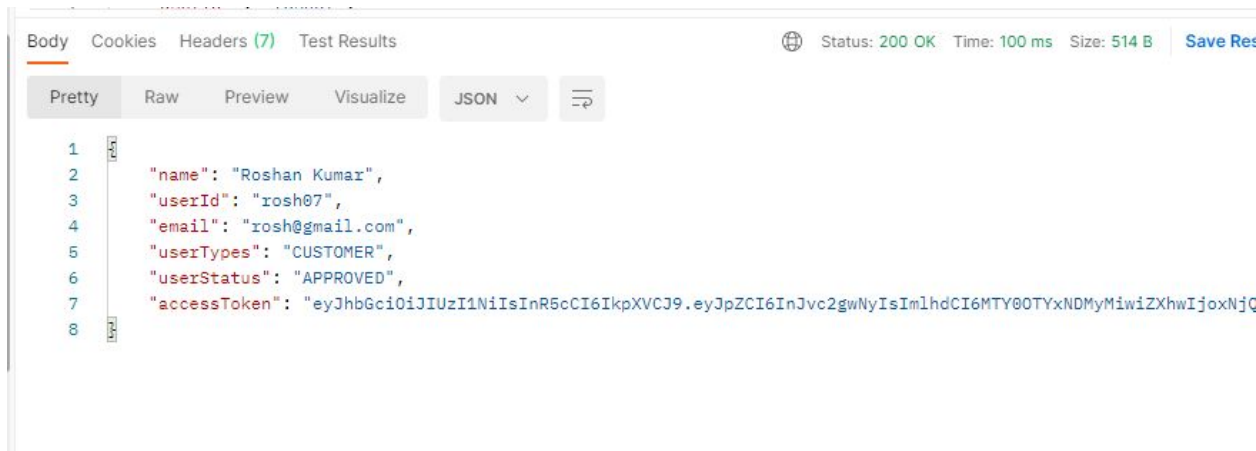
Fetch the user id from the request

```
const userIdReq = req.userId;
```

Next, we will use `findOneAndUpdate()` function of User schema and pass user id and password to update. For password, we will use `hashSync` function of `bcrypt` library which is used to generate hash of password.

```
try {
  const user = await User.findOneAndUpdate({
    userId: userIdReq
  }, {
    password: bcrypt.hashSync(req.body.password, 8)
  }).exec();
  res.status(200).send({
    message: `User record has been updated successfully`
  });
} catch (err) {
  console.error("Error while updating the record", err.message);
  res.status(500).send({
    message: "Some internal error occurred"
  })
};
```

## Request - Login and fetch the token form response



## Request - Use copied token in x-access-token header of the API

PUT

localhost:8080/crm/api/v1/users

Params

Authorization

Headers (9)

Body

Pre-request Script

Tests


Settings



Headers


8 hidden


	KEY	VALUE	DESCRIPTION	...
<input checked="" type="checkbox"/>	x-access-token	eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZC...		
	Key	Value	Description	


# Request - Body

MovieBookingApplication / session4 / Update the password 

 Save 

PUT  localhost:8080/crm/api/v1/users

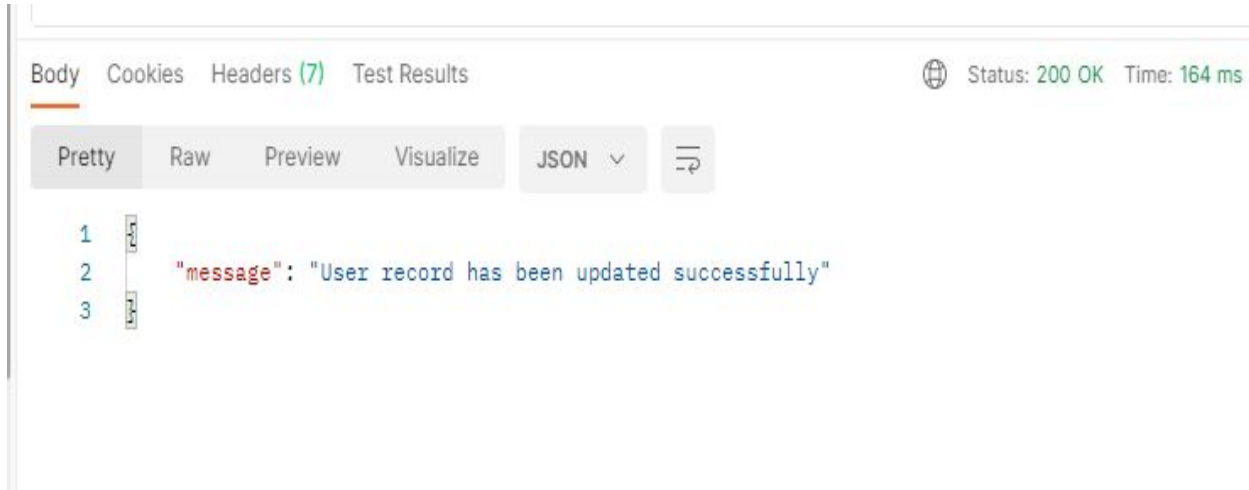
Params Authorization Headers (9) **Body**  Pre-request Script Tests Settings

☐ none ☐ form-data ☐ x-www-form-urlencoded ☒ raw ☐ binary ☐ GraphQL **JSON** 

```
1 {
2   ... "password" : "welcome12345"
3 }
```



# Response



Body Cookies Headers (7) Test Results

Status: 200 OK Time: 164 ms

Pretty Raw Preview Visualize JSON ↕

```
1 {  
2   "message": "User record has been updated successfully"  
3 }
```

# API - Admin Approval for Client Registration

In this API, we will update the user profile. Here only Admin can update the user and hence we need to check user type as well.

Let's have a look at all the steps needed one by one -

[Code Link] -

[https://github.com/Vishwa07dev/mba\\_backend/blob/session4/controllers/user.controller.js](https://github.com/Vishwa07dev/mba_backend/blob/session4/controllers/user.controller.js)

# API – Verify the token – STEP 1

We will fetch token from headers of the request -> x-access-token

```
let token = req.headers["x-access-token"];
```

- If token is not provided, we will send message saying that “No token provided!!”

```
if (!token) {  
    return res.status(403).send({  
        message: "No token provided!"  
    });  
}
```

Next, we will verify our token using JWT verify function. We will pass token and secret key to verify the token. If verification failed, we will send message saying that “Unauthorized!”

```
jwt.verify(token, config.secret, (err, decoded) => {  
    if (err) {  
        return res.status(401).send({  
            message: "Unauthorized!"  
        });  
    }  
    req.userId = decoded.id;  
    next();  
});
```

## API – Update the User – STEP 3

Here, we will fetch user id from the request and check if user type is admin or not. If user is not admin, we will send message saying “Require Admin Role!!”

```
isAdmin = async (req, res, next) => {  
  
  const user = await User.findOne({  
    userId: req.userId  
  })  
  if (user && user.userType == constants.userTypes.admin) {  
    next();  
  } else {  
    res.status(403).send({  
      message: "Require Admin Role!"  
    });  
    return;  
  }  
};
```

# Scenario 1 - When user type is not Admin

## Request - Login and fetch the token form response



The screenshot shows a web browser's developer tools interface. The 'Body' tab is selected, displaying a JSON response. The status bar at the top indicates a 200 OK status, 100 ms response time, and 514 B size. The JSON response contains user information and an access token.

```
1 {
2   "name": "Roshan Kumar",
3   "userId": "rosh07",
4   "email": "rosh@gmail.com",
5   "userTypes": "CUSTOMER",
6   "userStatus": "APPROVED",
7   "accessToken": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6InJvc2gwNyIsIm1hdCI6MTY0OTYxNDMyMiwiZXhwIjoxNjQ"
8 }
```

## Request - Use copied token in x-access-token header of the API

MovieBookingApplication / session4 / register client Save

PUT localhost:8080/crm/api/v1/users/rohit07

Params Authorization Headers (9) Body Pre-request Script Tests Settings

Headers 8 hidden

	KEY	VALUE	DESCRIPTION	∞
<input checked="" type="checkbox"/>	x-access-token	eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZC...		
	Key	Value	Description	

# Request - Body

MovieBookingApplication / session4 / register client Save

PUT localhost:8080/crm/api/v1/users/rohit07

Params Authorization Headers (9) **Body** Pre-request Script Tests Settings

☐ none ☐ form-data ☐ x-www-form-urlencoded ☒ raw ☐ binary ☐ GraphQL **JSON**

```
1 {
2   "name": "Rohit Kumar",
3   "userTypes": "CLIENT",
4   "userStatus": "APPROVED"
5 }
```

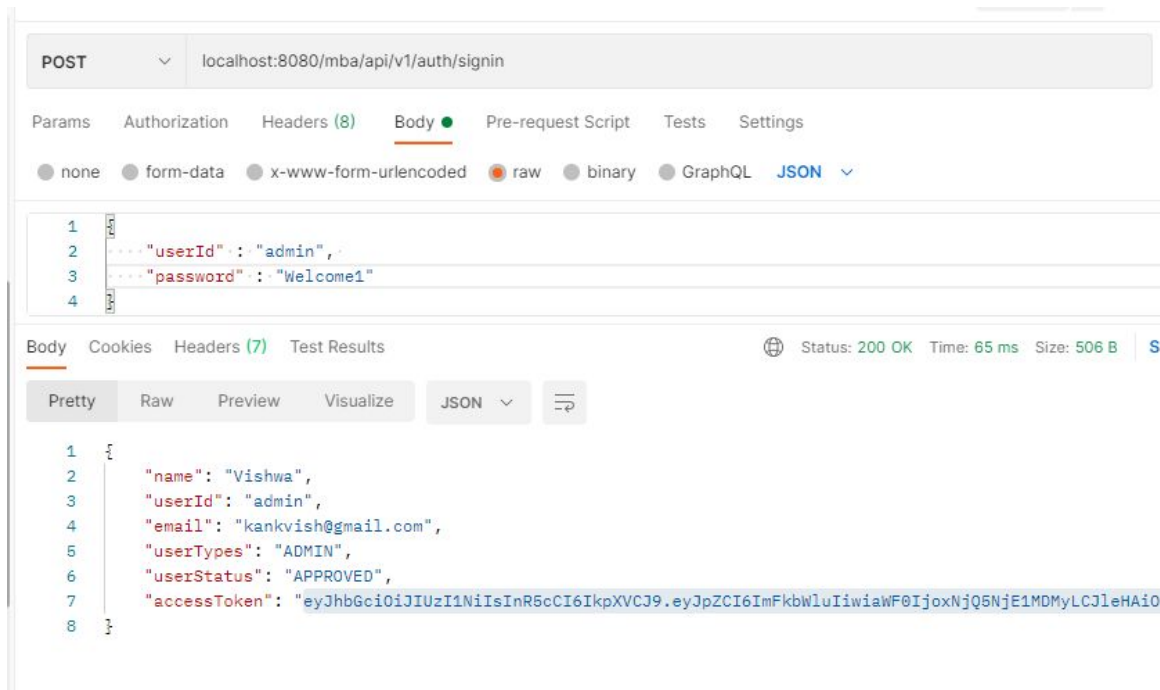
## Response - Since user type is not Admin





## Scenario 2 - When user type is Admin

### Request - Login and fetch the token form response



## Request - Use copied token in x-access-token header of the API

MovieBookingApplication / session4 / register client Save

PUT localhost:8080/crm/api/v1/users/rohit07

Params Authorization Headers (9) Body Pre-request Script Tests Settings

Headers 8 hidden

	KEY	VALUE	DESCRIPTION	∞
<input checked="" type="checkbox"/>	x-access-token	eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZC...		
	Key	Value	Description	

# Request - Body

PUT

localhost:8080/crm/api/v1/users/rohit07

Params

Authorization

Headers (9)

Body ●

Pre-request Script

Tests

Settings

☐ none

☐ form-data

☐ x-www-form-urlencoded

☒ raw

☐ binary

☐ GraphQL

JSON ▾

1

2

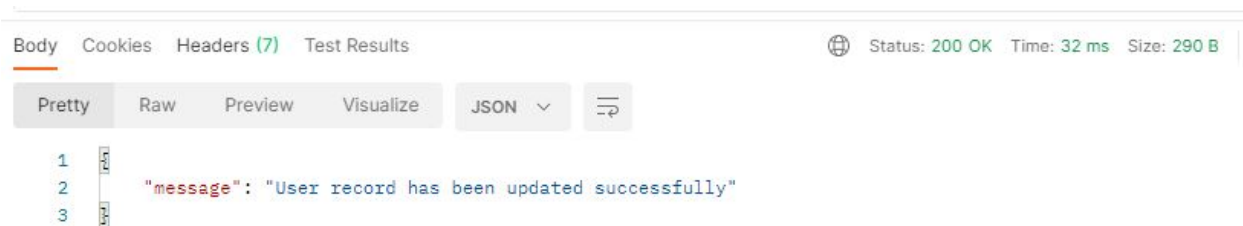
3

4

5

```
{
  "name": "Rohit Kumar",
  "userTypes": "CLIENT",
  "userStatus": "APPROVED"
}
```

## Response - Since user type is Admin



# Practice Questions

1. Write differences between Authentication and Authorization with example
2. Write a flow for Sign Up where the Signup operation will be failed and send a message saying that “Username/UserId not provided!!” if the username or user id will not be provided in the request.

**Thank You!**