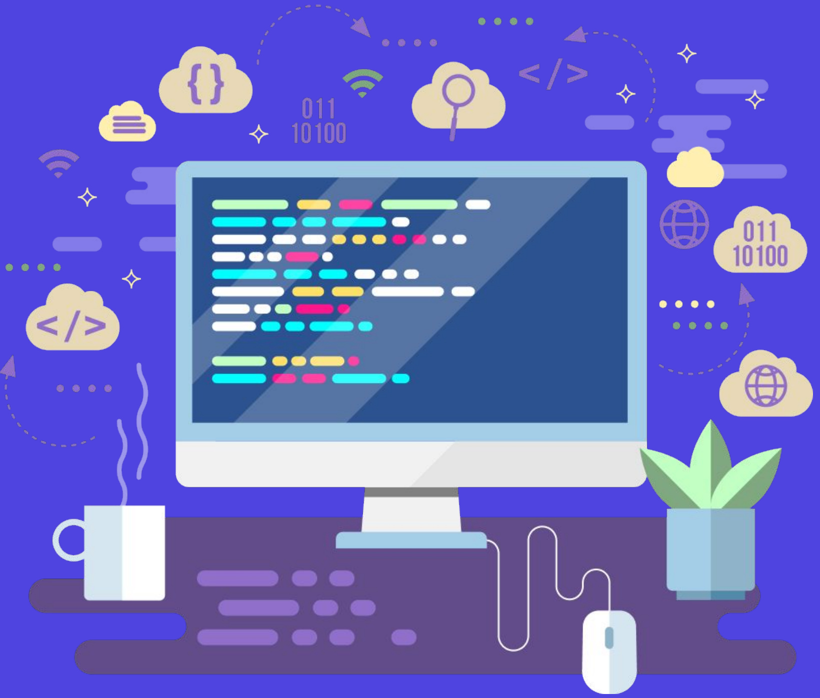
# Building the CRM App: JWT based Authentication and Authorization

Relevel
by Unacademy



## Agenda

1. Authentication
2. Authorization
3. Uses of JWT token for Authorization and Authentication

## Feature of CRM Application to be developed in this

- API for getting the list of all users
- API for getting the user based on UserID
- API for updating the user type and status
- Authenticating and Authorizing above APIs, so that only authenticated ADMIN users will be allowed to perform the above operations
- ENGINEER/ADMIN user should be able to login successfully after the approval from the ADMIN user

## 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 for authentication. Users use different combinations of letters, numbers, 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 -

1) 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.
2) 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.
3) 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.

- In this application, we are going to implement below methods -

1) Token based Authentication

- We are going to use JWT token for authentication

1) Role based Authentication

- We have 3 actors - CUSTOMER, ENGINEER & ADMIN
- All actors have different roles and they will perform different operations based on the role given to them

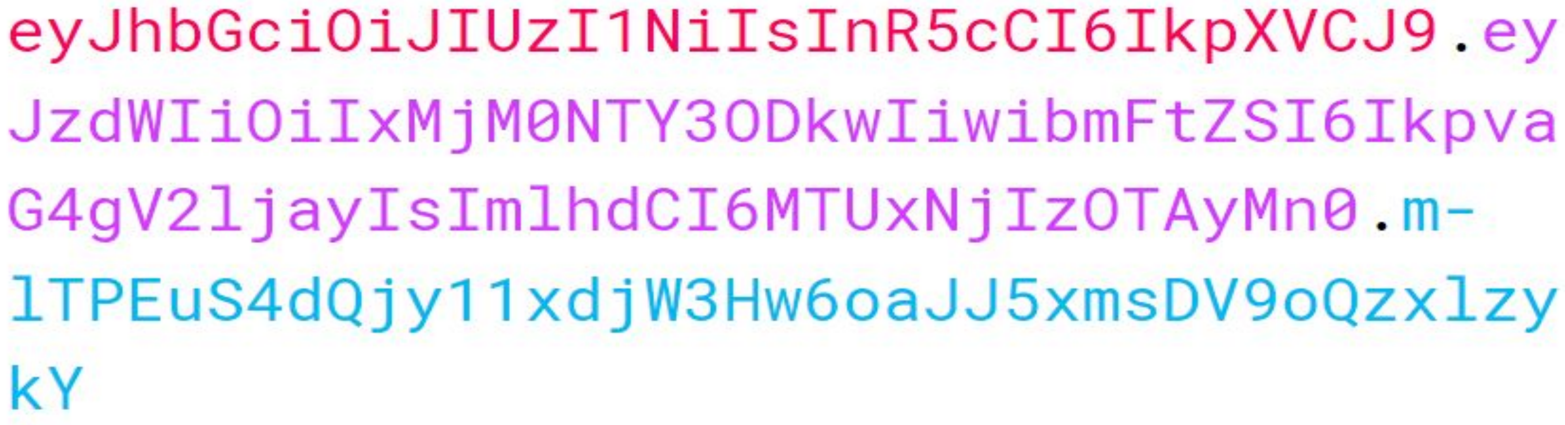
## JWT Tokens

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

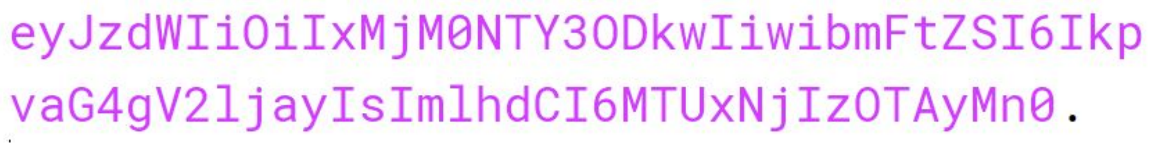


```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.ey
JzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6Ikpva
G4gV2ljayIsImlhdCI6MTUxNjIzOTAyMn0.m-
lTPEuS4dQjy11xdjW3Hw6oaJJ5xmsDV9oQzxlzy
kY
```

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

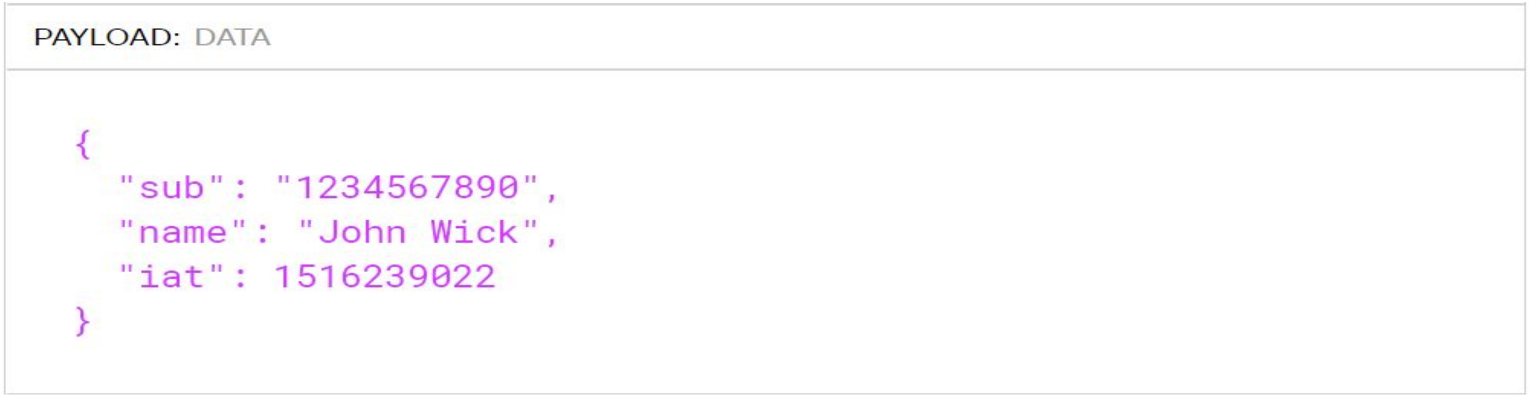
This contains the data that the server sends back to the client. The data is encoded in Base-64 format.



```
eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6Ikp
vaG4gV2ljayIsImlhdCI6MTUxNjIzOTAyMn0.
```

The above data on decoding becomes the following :-

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


PAYLOAD: DATA

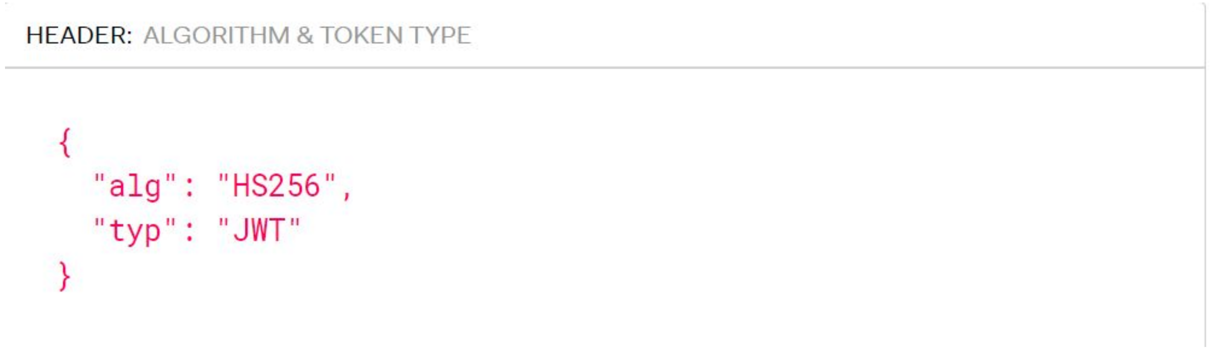
```
{
  "sub": "1234567890",
  "name": "John Wick",
  "iat": 1516239022
}
```

## 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

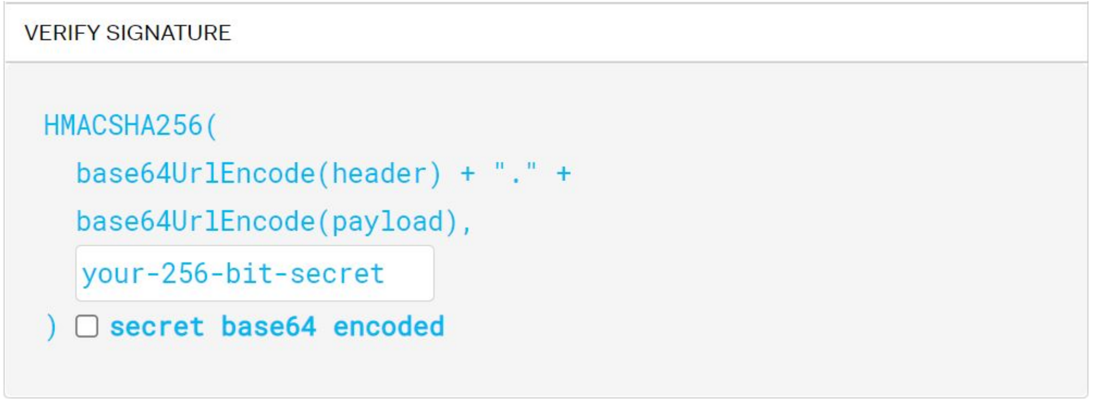
```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

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 :-



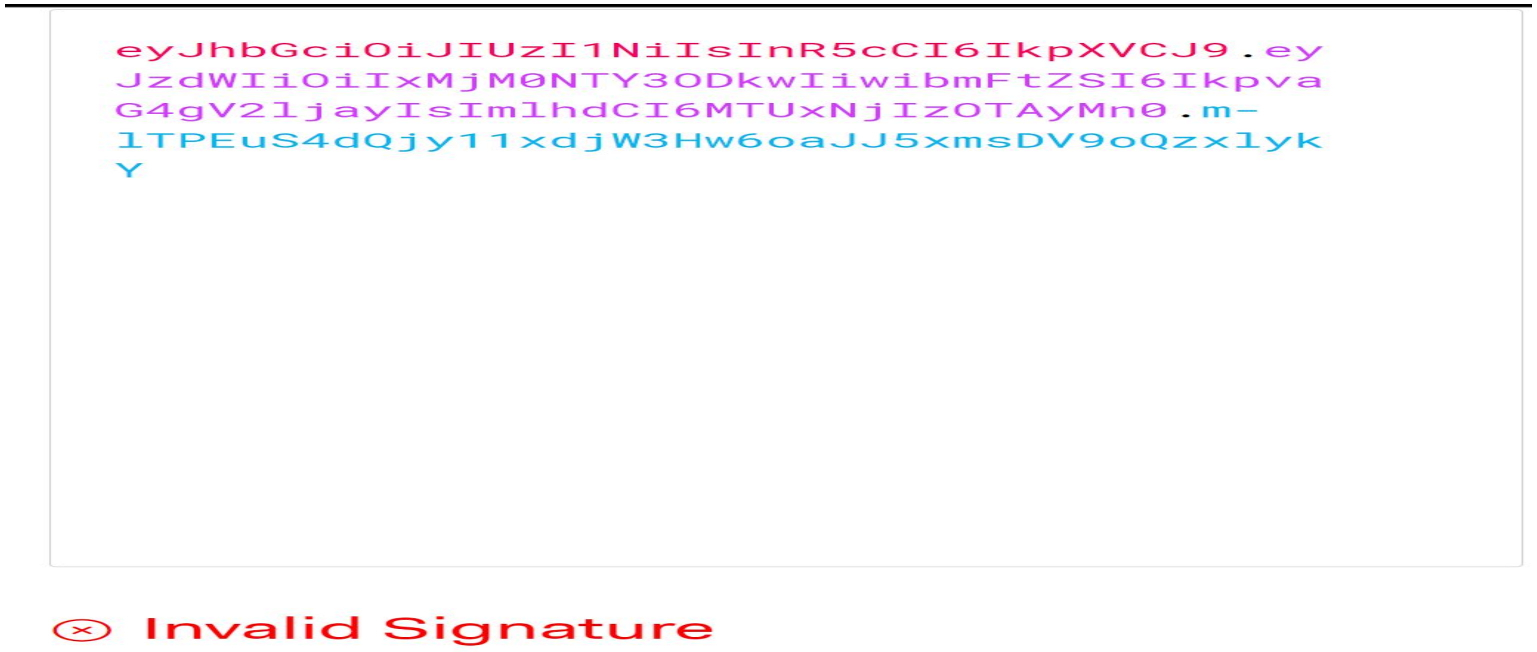
VERIFY SIGNATURE

```
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.ey
JzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6Ikpva
G4gV2ljayIsImlhdCI6MTUxNjIzOTAyMn0.m-
lTPEuS4dQjy11xdjW3Hw6oaJJ5xmsDV9oQzxlyk
Y
```

Invalid Signature

As we have seen in our previous sessions, we are going to use 3 actors which are Customer, Engineer and Admin. We have also see operations which can be performed by Admin which are as follows -

## Admin

- I should be able to see all the customers
- I should be able to see all the Engineers
- I should be able to see all the tickets details
- I should be able to see all the active tickets
- I should be able to filter the tickets based on status
- I should be able to re-assign a ticket to another Engineer
- I should be able to add a new Engineer
- I should be able to remove an Engineer

In this session, we are going to implement below operations of Admin -

- I should be able to see all the customers
- I should be able to see the customer based on userId
- I should be able to update the customer

## API - Getting the list of all users

Our first operation is to fetch the list of all users where we will also implement the authentication and authorization part for admin.

When a user is trying to get the list of users, we need to verify 2 things before sending the response.

Validate the token - We need to validate jwt token sent from the user if that token is valid or not

User is Admin or not - We need to check the user type. We want only the admin to see the list of users and hence, we need to validate the user type.

## Token Validation

```
verifyToken = (req, res, next) => {
    let token = req.headers["x-access-token"];
    if (!token) {
        return res.status(403).send({
            message: "No token provided!"
        });   }
    jwt.verify(token, config.secret, (err, decoded) => {
        if (err) {
            return res.status(401).send({
                message: "Unauthorized!"
            }); }
        req.userId = decoded.id;
        next();
    });};
```

The above code is validating the token sent by the user using a secret key which we have configured in our auth.config.js file

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

## jwt.verify() function -

This method takes an *algorithm* argument which can be customized, *secretOrPublicKey* argument which is the secret token and the other arguments which can be used for validation logic.
It is used to validate the token used. If token is invalid, it returns *JsonWebTokenError* error along with the message - jwt malformed. If we receive this message, we can reject the user request to access the application.

## Validate User type

Let's see below code snippet -

```
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;
    }
};
```

In the above code, we are comparing the user type and if it is not admin then we are sending error message - “Require Admin Role”

Once the above steps are completed and verified, we will call the below function to fetch the users list.
Let’s have a look at below code snippet -
https://p.ip.fi/DfOj

**STEP 1 -** We will support below query parameters to filter out the users
User type
User name
User status

```
//Supporting the query param
    let userTypeReq = req.query.userType;
    let userStatusReq = req.query.userStatus;
    let userNameReq = req.query.name;
```

**STEP 2 -** When we pass user name as query parameter, below code will execute where we are using find() function of mongoDB on User schema and passing username as argument

```
if (userNameReq) {
        try {
            users = await User.find({
                userName: userNameReq
            });
        } catch (err) {
            console.err("error while fetching the user for userName : ",
userNameReq);
            res.status(500).send({
                message: "Some internal error occured"
            })
        }
    }
```

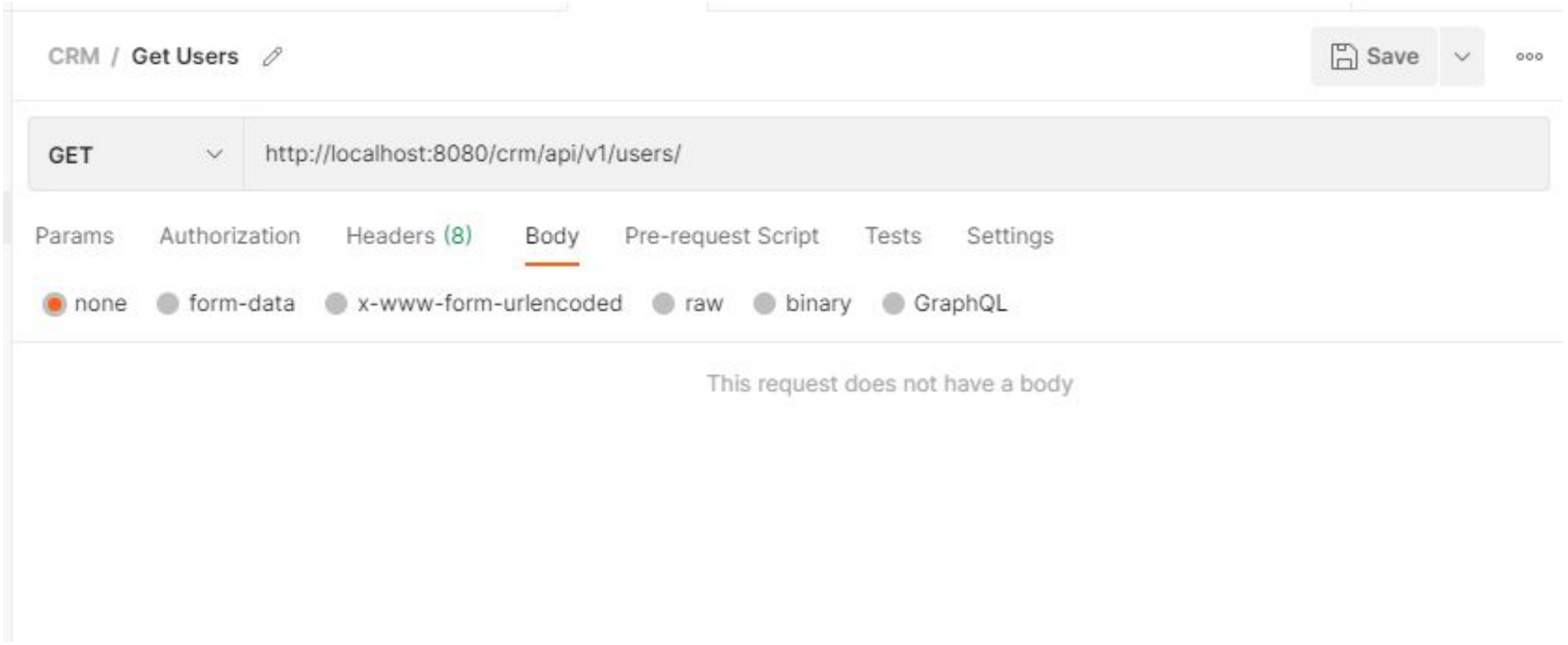
**STEP 3** - if user pass user type and user status both, we will pass both arguments in find function

```
else if (userTypeReq && userStatusReq) {
        try {
            users = await User.find({
                userType: userTypeReq,
                userStatus: userStatusReq
            });
        } catch (err) {
            console.err(`error while fetching the user for userType [${userTypeReq}]
and userStatus [${userStatusReq}]`);
            res.status(500).send({
                message: "Some internal error occured"
            })
        }
    }
```

**STEP 4** - Finally if there are no query parameters, we will send all the users present in our User schema

```
else {
        try {
            users = await User.find();

        } catch (err) {
            console.err(`error while fetching the users `);
            res.status(500).send({
                message: "Some internal error occured"
            })
        }
    }
```

## Request

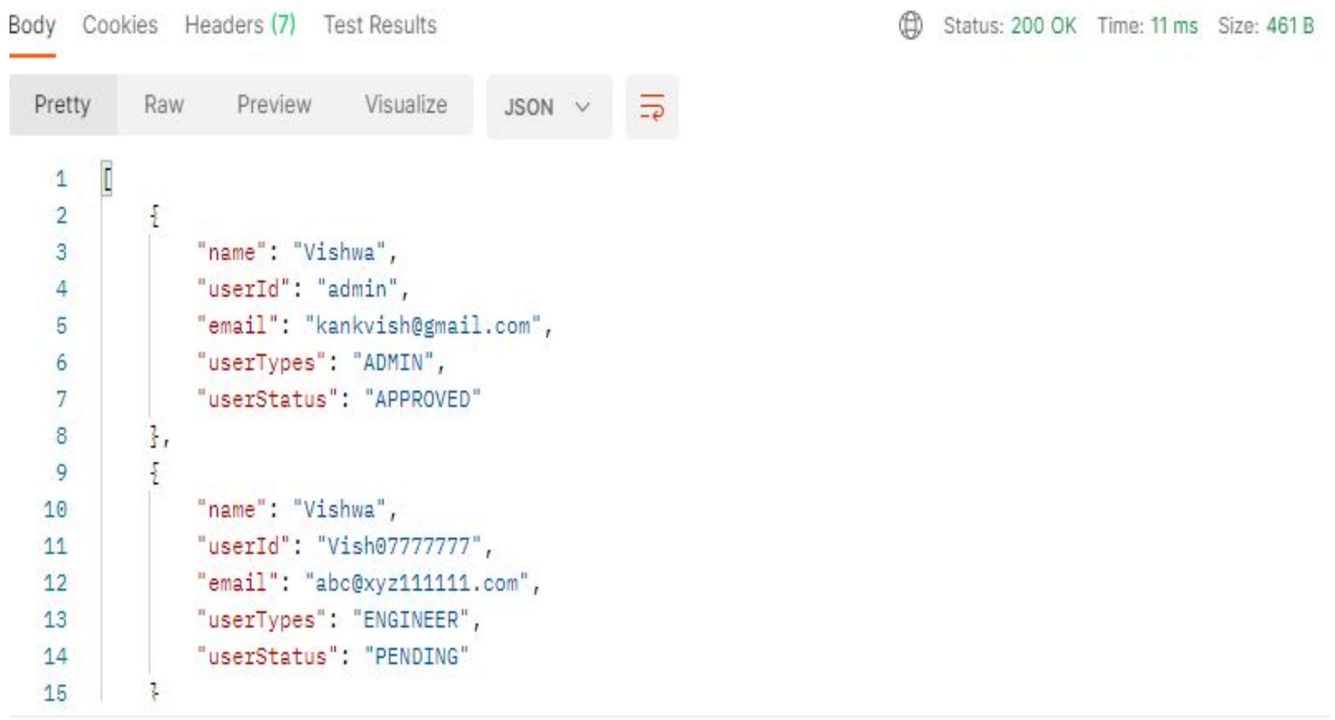


GET http://localhost:8080/crm/api/v1/users/

[✓ none] form-data x-www-form-urlencoded raw binary GraphQL

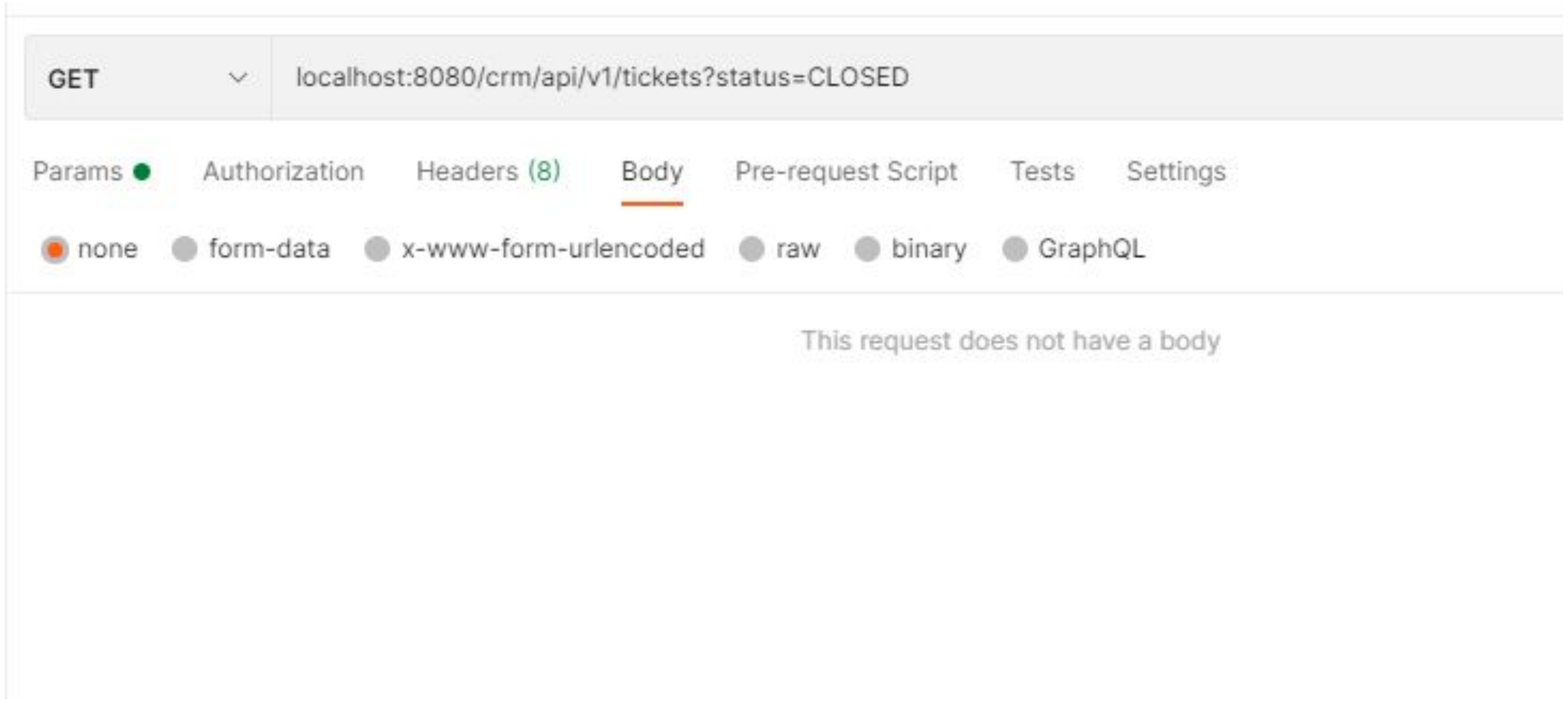
This request does not have a body

## Response



```
[
    {
        "name": "Vishwa",
        "userId": "admin",
        "email": "kankvish@gmail.com",
        "userTypes": "ADMIN",
        "userStatus": "APPROVED"
    },
    {
        "name": "Vishwa",
        "userId": "Vish07777777",
        "email": "abc@xyz111111.com",
        "userTypes": "ENGINEER",
        "userStatus": "PENDING"
    }
```

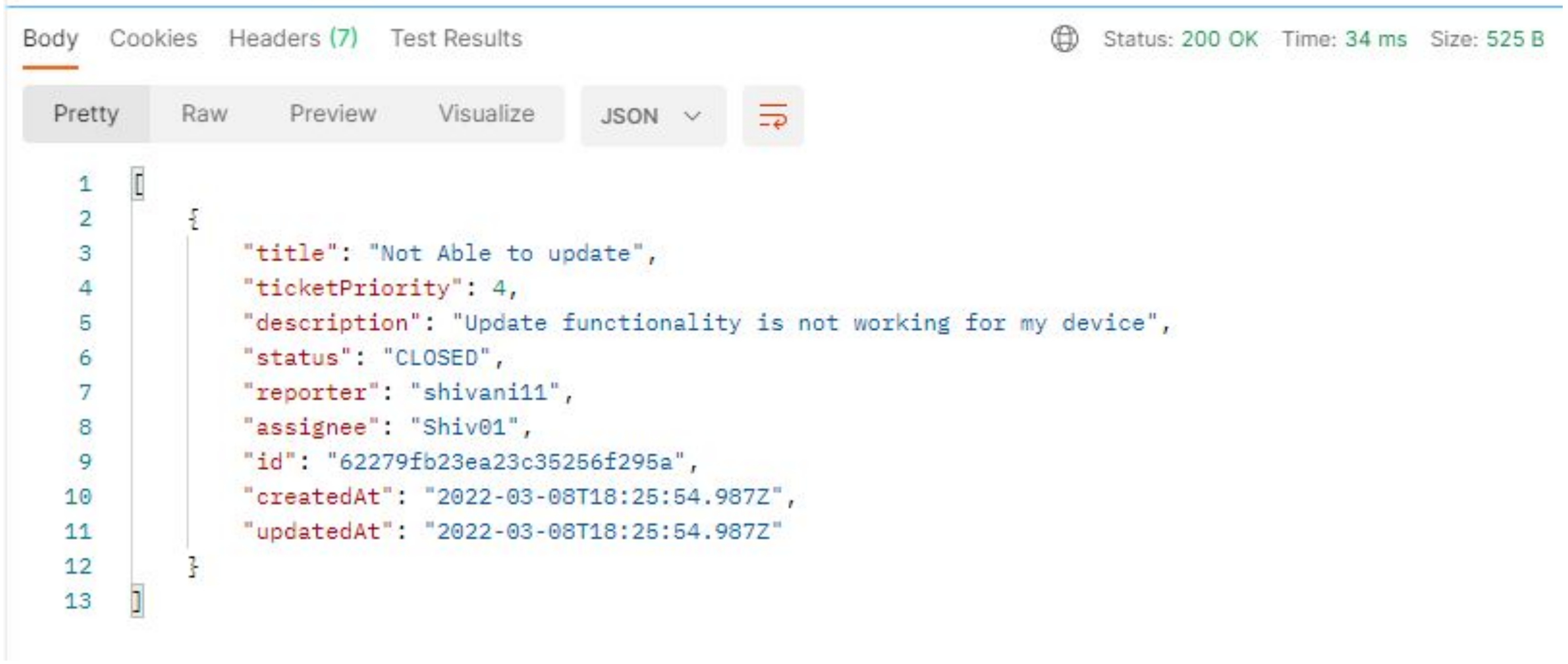
## Request


GET localhost:8080/crm/api/v1/tickets?status=CLOSED

[✓ none] form-data x-www-form-urlencoded raw binary GraphQL

This request does not have a body

## Response



```
[
    {
        "title": "Not Able to update",
        "ticketPriority": 4,
        "description": "Update functionality is not working for my device",
        "status": "CLOSED",
        "reporter": "shivani11",
        "assignee": "Shiv01",
        "id": "62279fb23ea23c35256f295a",
        "createdAt": "2022-03-08T18:25:54.987Z",
        "updatedAt": "2022-03-08T18:25:54.987Z"
    }
]
```

## API - Getting the User based on UserId

Our operation is to fetch the user based on the userId given in input and we will also implement the authentication and authorization part for admin.

When an admin is trying to get the user detail based on userId, we will call verifyToken and isAdmin function again similar to what we have seen in the previous API.

Once verification is done, we will call the below function to fetch the user

```
exports.findById = async (req, res) => {
    const userIdReq = req.params.userId;

    const user = await User.find({
        userId: userIdReq
    })
    if (user) {
        res.status(200).send(objectConvertor.userResponse(user));
    } else {
        res.status(200).send({
            message: `User with this id [${userIdReq}] is not present`
        })
    }}
```

Again we are calling the find() function on User schema and passing userId as a parameter. This will fetch the user details from the schema present in our database.

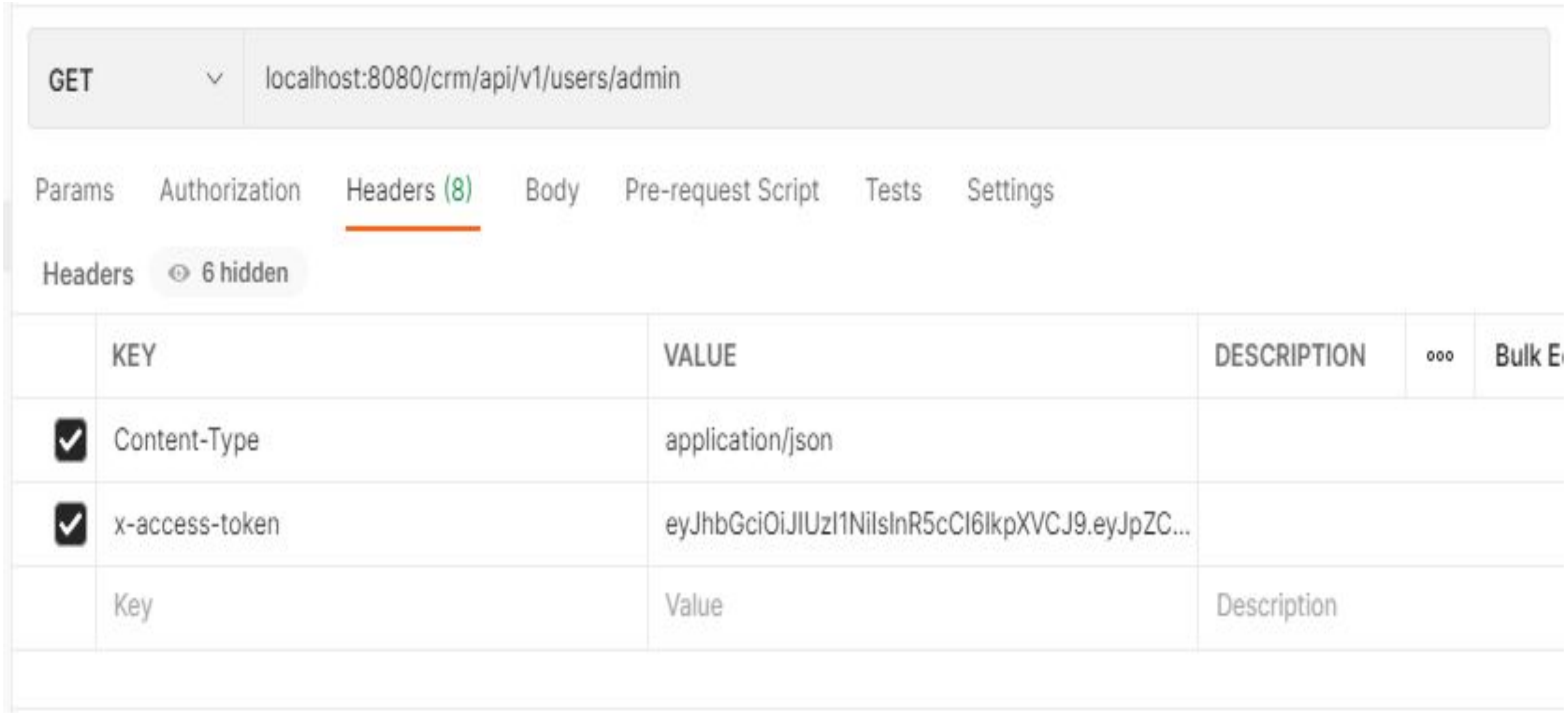
If the user is not found in the database, then we will send a message like “User with this id is not present”

**Functionality of objectConvertor** - This is used to convert an object from one object type to another object type. userResponse function is calling below code -

```
exports.userResponse = (users) => {

    usersResult = [];
    users.forEach(user => {
        usersResult.push({

            name: user.name,
            userId: user.userId,
 email: user.email,
            userTypes:
user.userType,
            userStatus:
user.userStatus});
    });
    return usersResult;

}
```

Here, we are converting const users to User model. We are mapping its different attributes to User attributes and returning usersResult schema as response.

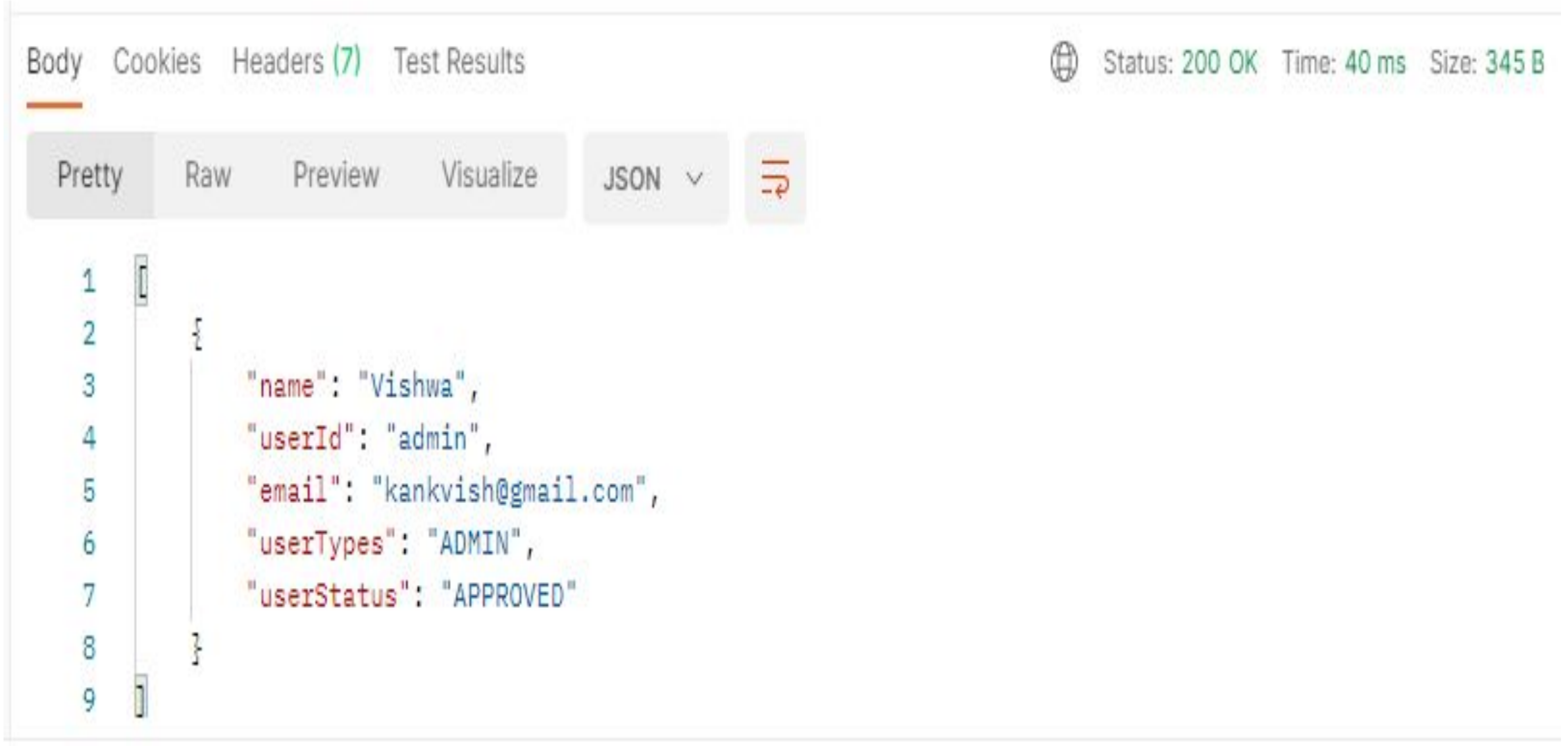
## Request



GET localhost:8080/crm/api/v1/users/admin

| | KEY | VALUE | DESCRIPTION | ooo | Bulk E |
|---|---|---|---|---|---|
| ✓ | Content-Type | application/json | | | |
| ✓ | x-access-token | eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZC... | | | |
| | Key | Value | Description | | |

## Response



```
[
    {
        "name": "Vishwa",
        "userId": "admin",
        "email": "kankvish@gmail.com",
        "userTypes": "ADMIN",
        "userStatus": "APPROVED"
    }
]
```

## API - Update the User information - Type and Status

Our operation is to update the user type and status and we will also implement the authentication and authorization part for admin.
When an admin is trying to update the user detail, we will call verifyToken and isAdmin function again similar to what we have seen in the previous API to validate the user.

Once verification is done, we will call the actual function to update the user. Let’s have a look at the type and status list which we have defined for different users.
Based on the actors which we have discussed in our intro session, we have defined three types of users and also 3 types of status.

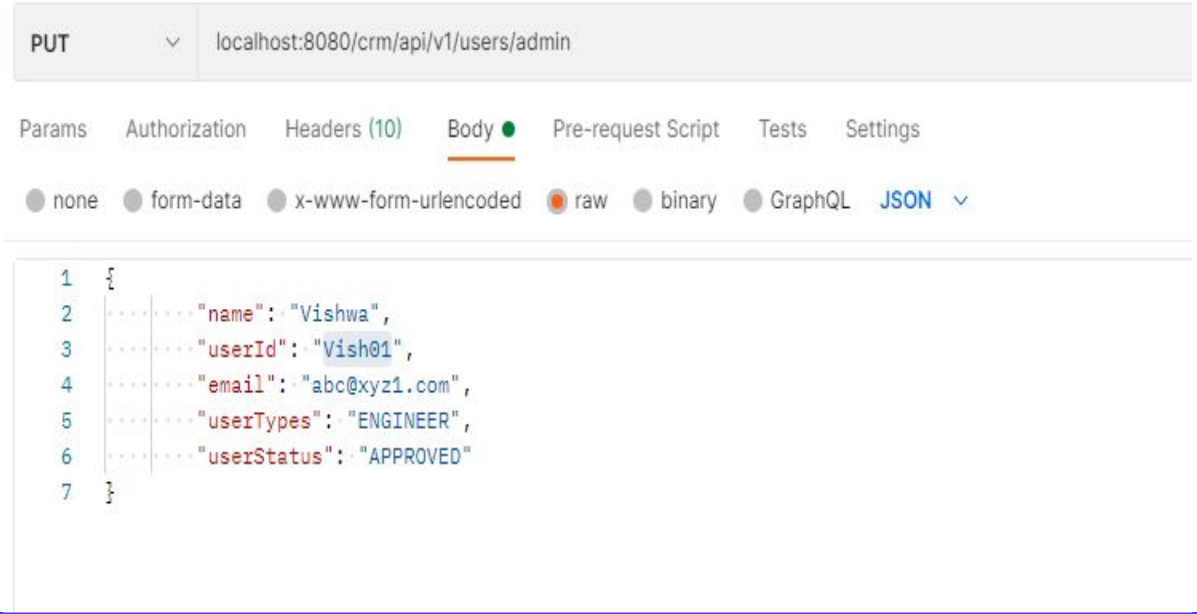
```
userTypes: {
        customer: 'CUSTOMER',
        engineer: 'ENGINEER',
        admin: 'ADMIN'
    },
    userStatus: {
        pending: 'PENDING',
        approved: 'APPROVED',
        rejected: 'REJECTED'
    }
```

Now, let's check function which we are calling to update the details

```
exports.update = async (req, res) => {
    const userIdReq = req.params.userId;
    try {
        const user = await User.findOneAndUpdate({
            userId: userIdReq
        }, {
            userName: req.body.userName,
            userStatus: req.body.userStatus,
            userType: req.body.userType

        }).exec();
        res.status(200).send({
            message: `User record has been updated successfully`
        });
    } catch (err) {
        console.err("Error while updating the record", err.message);
        res.status(500).send({
            message: "Some internal error occured"
        }) };
}
```

We are calling findOneAndUpdate function on User schema where we are passing userId to locate the user and then userName, userType, and userStatus to update the details.
If the update is successful, then we will send message - “User record has been updated successfully”

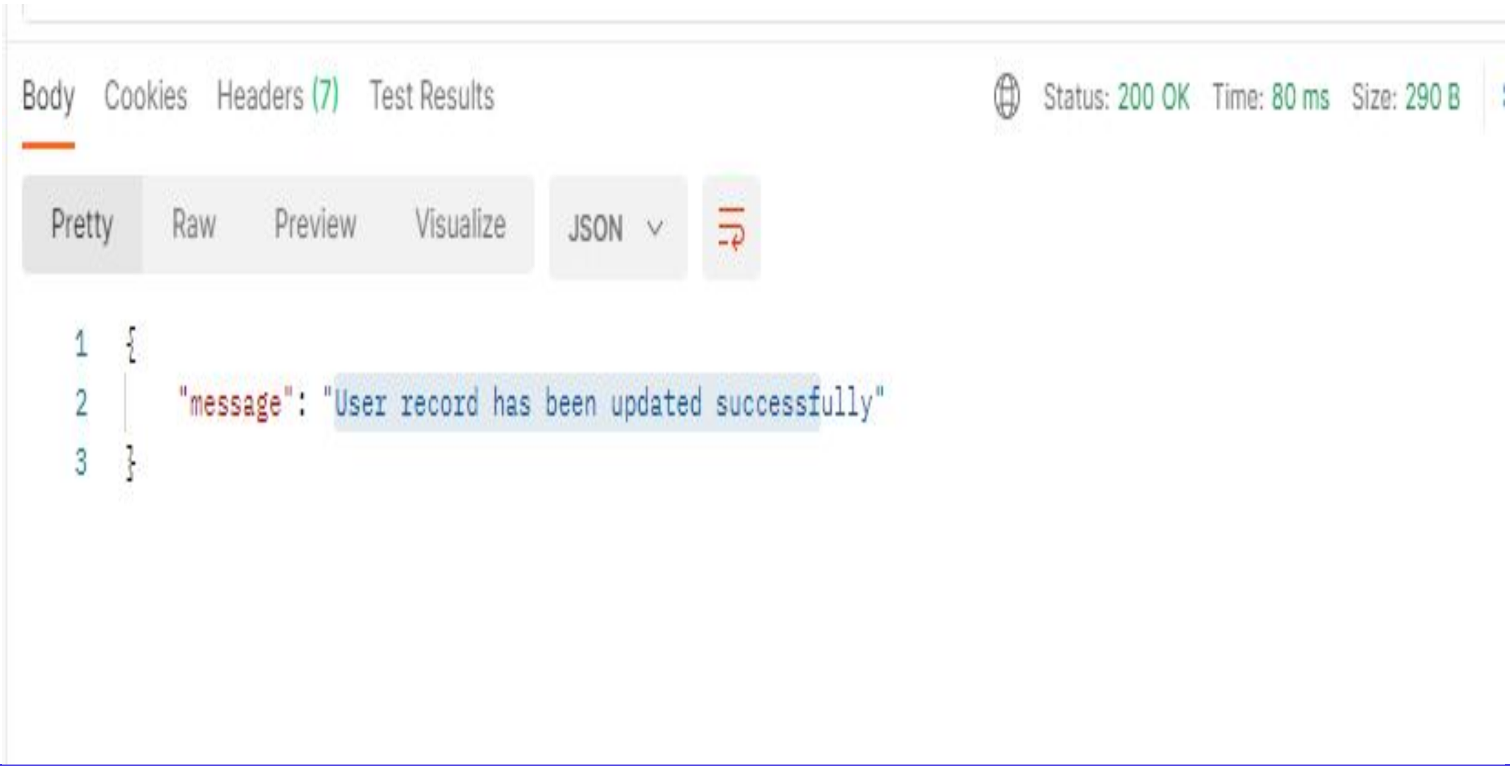
## Request



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

```
{
    "name": "Vishwa",
    "userId": "Vish01",
    "email": "abc@xyz1.com",
    "userTypes": "ENGINEER",
    "userStatus": "APPROVED"
}
```

## Response



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

## Practice Code

- Write differences between Authentication and Authorization with example
- Write API to update user email, user type, and user status based on user id given in the input

## MCQ Questions

1) Which statement is not true about authentication and authorization?

A) Authentication is based on the passwords entered by the user while Authorization is based on the roles of the user

B) Authentication is the first step while Authorization takes place after Authentication

**C) Authorization is the first step while Authentication takes place after Authorization [Correct Answer]**

D) None

2) Which function is used to fetch all the data present in the schema in MongoDB?

A) find

**B) findAll[Correct Answer]**

C) findById

D) None

3) Which function is used to fetch the data based on a specific attribute present in the schema in MongoDB?

A.) find

B.) findOne

C.) findAll

**D.) All of the above [Correct Answer]**

## MCQ Questions

4) Which type of parameter is present in the below URL?
GET /crm/api/v1/users/{userId}
**A.) path parameter [Correct Answer]**
B.) query parameter
C.) A and B
D.) None

5) What is the functionality of the PUT method in HTTP?
**A.) update or create specific resource [ Correct Answer ]**
B.) create a resource
C.) delete a resource
D.) get a resource

THANK YOU