# Building the CRM App: User Activities


**Relevel**
by Unacademy

# Topics to be Covered

**Ticket Creation:**
RESTful APIs to create tickets by Users

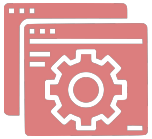
**Fetch tickets:**
RESTful APIs to fetch tickets status and other information by Users

**Update tickets:**
RESTful APIs to update tickets by Users

# Feature of CRM Application to be developed in this session

- API for the authenticated user to raise a request
- API for the authenticated user to update an existing request
- API for an authenticated user to check the status of the request
- API for an authenticated user to check the list of all the requests raised so far
- Registered Engineer if any should be assigned the ticket automatically
- If no Engineer is available, wait for the availability of any registered Engineers.
- In the case of multiple tickets, assign to the available Engineers on the round-robin basis

# Feature of CRM Application to be developed in this session



As we have seen in our previous sessions, we are going to use 3 actors which are Customer, Engineer, and Admin.

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

- Operations of Customer
- I should be able to see all the tickets raised
- I should be able to filter the tickets based on status
- I should be able to raise an issue
- I should be able to modify the issue raised

# API – Raise a Request

Our first operation is to create a ticket and assign it to the engineer if present.
If an engineer is not present, then we will wait till the engineer will be present.
If there are multiple engineers, then we will assign based on the round-robin process.

# API – Raise a Request

**Step1** - Before creating a ticket, we need to validate the request

Below functions are written in verifyTicketReqBody.js file.
These will be added as the middleware for the request validation

Validate the title- we will validate if title is present in the request or not

```
//Validating the title
 if (!req.body.title) {
     res.status(400).send({
         message: "Failed! Title is not provided !"
     });
     return;
 }
```

# API – Raise a Request

Validate the description - We will validate if description is present in the request or not

```
if (!req.body.description) {
        res.status(400).send({
            message: "Failed!
Description is not provided !"
        });
        return;
    }
```

# API – Raise a Request

**Step2** - Now, once validation is successful, we will call our actual function to create a ticket.
Complete function - https://p.ip.fi/rDkW

Create ticketObject by fetching all the information from the request

```
 const ticketObject = {
        title: req.body.title,
        ticketPriority: req.body.ticketPriority,
        description: req.body.description,
        status: req.body.status,
        reporter: req.userId //this will be retrieved
from the middleware
    }
```

# API – Raise a Request

Create a ticket - Once the engineer is present, we will create a ticket.
Then we will add that ticket to the customer.
We will also assign that ticket to the Engineer.
Once done, we will return.

```
const ticket = await Ticket.create(ticketObject);
        if (ticket) {          //Updating the customer
            const user = await User.findOne({
                userId: req.userId            });
            user.ticketsCreated.push(ticket._id);
            await user.save();
            //Updating the Engineer
            engineer.ticketsAssigned.push(ticket._id);
            await engineer.save();
res.status(201).send(objectConvertor.ticketResponse(ticke
t));         }
```

# API – Raise a Request

## Request

CRM / Create Ticket

Save

POST

localhost:8080/crm/api/v1/tickets/

Send

Params Authorization Headers (10) Body Pre-request Script Tests Settings Cookies

none form-data x-www-form-urlencoded raw binary GraphQL JSON Beautify

```
1 {
2     "title": "Not able to use the device",
3     "description" : "Device is not turning on with power"
4 
5 }
```

# API – Raise a Request

## Response

Body Cookies Headers (7) Test Results Status: 201 Created Time: 26 ms Size: 523 B

Pretty Raw Preview Visualize JSON


```
{
    "title": "Not able to use the device",
    "ticketPriority": 4,
    "description": "Device is not turning on with power",
    "status": "OPEN",
    "reporter": "shivani11",
    "assignee": "utkarsh11",
    "id": "62279fb23ea23c35256f295a",
    "createdAt": "2022-03-08T18:25:54.987Z",
    "updatedAt": "2022-03-08T18:25:54.987Z"
}
```

# API – Update the Ticket

Our operation is to update the ticket raised by the customer.

**Steps -**

**Step 1 -** Before updating a ticket, we need to validate the status of the ticket. We will verify the status from the available status list which are “CLOSED”, “BLOCKED”, “IN_PROGRESS, “OPEN”

# API – Update the Ticket

```
validateTicketStatus = async (req, res, next) => {
    //Validating the user type
    const status = req.body.status;
    const statusTypes = [constants.ticketStatus.open,
constants.ticketStatus.closed, constants.ticketStatus.inProgress,
constants.ticketStatus.blocked]
    if (status && !statusTypes.includes(status)) {
        res.status(400).send({
            message: "status provided is invalid. Possible values
CLOSED | BLOCKED | IN_PROGESS | OPEN "
        });
        return;
    }
    next();
}
```

# API – Update the Ticket

**Step 2 -** Now, we will call our actual function to update the ticket.

We need to focus on the points below before updating the ticket.

- Only the user who has created the ticket is allowed to update the ticket
- We will get the ticket from the database and then verify the ticket reporter who is the creator of the ticket and requested user. If both are same, then we will continue our operation to update the ticket
- We will fetch all the information from the request and update the ticket in the database using save() function.

# API – Update the Ticket

```
exports.updateTicket = async (req, res) => {
    const ticket = await Ticket.findOne({ _id: req.params.id });
    if (ticket.reporter == req.userId) {
        //Allowed to update
        ticket.title = req.body.title != undefined ?
req.body.title : ticket.title,
            ticket.description = req.body.description !=
undefined ? req.body.description : ticket.description,
            ticket.ticketPriority = req.body.ticketPriority !=
undefined ? req.body.ticketPriority : ticket.ticketPriority,
            ticket.status = req.body.status != undefined ?
req.body.status : ticket.status
```

# API – Update the Ticket

```
        var updatedTicket = await ticket.save();

res.status(200).send(objectConvertor.ticketResponse(updatedTicket
));
    } else {
        console.log("Ticket was being updated by someone who has
not created the ticket");
        res.status(401).send({
            message: "Ticket can be updated only by the customer
who created it"
        })
    }

};
```

# API – Update the Ticket

## Request

PUT localhost:8080/crm/api/v1/tickets/62279fb23ea23c35256f295a

Params Authorization Headers (10) Body Pre-request Script Tests Settings

none form-data x-www-form-urlencoded raw binary GraphQL JSON

```
1 {
2     "title": "Not Able to update",
3     "ticketPriority": 4,
4     "description": "Update functionality is not working for my device",
5     "status": "CLOSED"
6 
7 }
```

# API – Update the Ticket

## Response

Body Cookies Headers (7) Test Results Status: 200 OK Time: 19 ms Size: 526 B

Pretty Raw Preview Visualize JSON


```
{
    "title": "Not Able to update",
    "ticketPriority": 4,
    "description": "Update functionality is not working for my device",
    "status": "CLOSED",
    "reporter": "shivani11",
    "assignee": "utkarsh11",
    "id": "62279fb23ea23c35256f295a",
    "createdAt": "2022-03-08T18:25:54.987Z",
    "updatedAt": "2022-03-08T18:25:54.987Z"
}
```

# API – Fetch all the tickets

Our operation is to fetch all the tickets created by the user.

We need to focus on the points below -

- We will use find() function of mongoDB
- We will fetch userId and status from the request and use these parameters to fetch the tickets
- If we are not passing status in the request, then it will fetch all the tickets based on the userId only

# API – Fetch all the tickets

```
/**
 * Get the list of all the tickets created by me
 */
exports.getAllTickets = async (req, res) => {
    const queryObj = {
        reporter: req.userId
    }
    if (req.query.status != undefined) {
        queryObj.status = req.query.status;
    }
    const tickets = await Ticket.find(queryObj);
res.status(200).send(objectConvertor.ticketListResponse(tickets))
;
}
```

# API – Fetch all the tickets

## Request - Headers

userId will be fetched from token

CRM / fetch All Tickets

Save

GET localhost:8080/crm/api/v1/tickets

Params Authorization Headers (8) Body Pre-request Script Tests Settings

Headers 6 hidden

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

# API – Fetch all the tickets

## Request - Body

GET localhost:8080/crm/api/v1/tickets

Params Authorization Headers (8) Body Pre-request Script Tests Settings

none form-data x-www-form-urlencoded raw binary GraphQL

This request does not have a body

# API – Fetch all the tickets

## Response

Body Cookies Headers (7) Test Results Status: 200 OK Time: 20 ms Size: 528 B

Pretty Raw Preview Visualize JSON


```
[
    {
        "title": "Not Able to update",
        "ticketPriority": 4,
        "description": "Update functionality is not working for my device",
        "status": "CLOSED",
        "reporter": "shivani11",
        "assignee": "utkarsh11",
        "id": "62279fb23ea23c35256f295a",
        "createdAt": "2022-03-08T18:25:54.987Z",
        "updatedAt": "2022-03-08T18:25:54.987Z"
    }
]
```

# API – Fetch the ticket based on the ticketId

Our operation is to fetch the ticket based on the ticketId given in the request.

We need to focus on the points below -

- We will use findOne() function of mongoDB
- We will fetch userId from the request and use it to fetch the ticket

```
/**
 * Get the ticket based on the ticketId
 */
exports.getOneTicket = async (req, res) => {
    const ticket = await Ticket.findOne({
        _id: req.params.id
    })
    res.status(200).send(objectConvertor.ticketResponse(ticket));
}
```

# API – Fetch the ticket based on the ticketId

## Request

CRM / Fetch the ticket

Sav

GET localhost:8080/crm/api/v1/tickets/62279fb23ea23c35256f295a

Params Authorization Headers (8) Body Pre-request Script Tests Settings

none form-data x-www-form-urlencoded raw binary GraphQL

This request does not have a body

# API – Fetch the ticket based on the ticketId

## Response

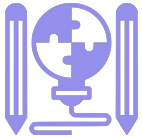
Body Cookies Headers (7) Test Results Status: 200 OK Time: 14 ms Size: 526 B

Pretty Raw Preview Visualize JSON


```
{
    "title": "Not Able to update",
    "ticketPriority": 4,
    "description": "Update functionality is not working for my device",
    "status": "CLOSED",
    "reporter": "shivani11",
    "assignee": "utkarsh11",
    "id": "62279fb23ea23c35256f295a",
    "createdAt": "2022-03-08T18:25:54.987Z",
    "updatedAt": "2022-03-08T18:25:54.987Z"
}
```

# Practice Code

- Run a flow where the Customer is trying to create the ticket and the Engineer is unavailable to be assigned to the ticket.
- Return the message which states that “Engineer not available”
- Run a flow where the Customer is trying to update the status of the ticket which is in CLOSED status.
- Return the message which states that “Ticket status is CLOSED. No update is required further”



# MCQs

1. **What will be the value of the parameter if we don't pass it in our API?**
   A. NA
   B. null
   C. undefined
   D. None

# MCQs

1. **What will be the value of the parameter if we don't pass it in our API?**
   A. NA
   B. null
   C. undefined
   D. None

**Answer: C**

# MCQs

2. **Which function is used to fetch a single document present in the schema in MongoDB?**
   A. find
   B. findOne
   C. findAll
   D. None

# MCQs

2. **Which function is used to fetch a single document present in the schema in MongoDB?**
   A. find
   B. findOne
   C. findAll
   D. None

**Answer: B**

# MCQs

3. **Which type of user can update the ticket?**
   A. User who created the ticket
   B. User who can view the ticket
   C. Any user
   D. None

# MCQs

3. **Which type of user can update the ticket?**
   A. User who created the ticket
   B. User who can view the ticket
   C. Any user
   D. None

**Answer: A**

# MCQs

4. **Which type of parameter is present in the below URL?**

**GET /crm/api/v1/tickets?status=OPEN**

A. query parameter
B. path parameter
C. A and B
D. None

# MCQs

**4. Which type of parameter is present in the below URL?**

**GET /crm/api/v1/tickets?status=OPEN**

A. query parameter

B. path parameter

C. A and B

D. None

**Answer: A**

# MCQs

5. **What will be the status of the ticket when created by the user?**
   A. OPEN
   B. CLOSED
   C. IN_PROGRESS
   D. BLOCKED

# MCQs

**5. What will be the status of the ticket when created by the user?**

A. OPEN

B. CLOSED

C. IN_PROGRESS

D. BLOCKED

**Answer: A**

# Thank You!