Authenticate User Ticket Booking

Relevel
by Unacademy

Agenda

- Set up data model for booking and transaction
- Authenticated APIs for allowing authenticated customers to perform booking
- Ability to cancel the booking
- Ability to complete payment within a given time frame

Data Models

- Booking Resource
- threatreId: It is ObjectID and reference from Threatre schema which is required.
- movieId: It is ObjectID and reference from Movie schema which is required.
- userId: It is ObjectID and reference from User schema which is required.
- Timing: It is a string which is required.
- Status: It is a string which is required with default as IN_PROGRESS.
- createdAt: It is a date type and immutable.
- updatedAt: It is a date type.
- noOfSeats: It is a number type and required.
- totalCost: It is a number type.

Booking Resource Application - CRUD Operations

[Code Link] -
https://github.com/Vishwa07dev/mba_backend/blob/session7/routes/booking.routes.js

- getAllBookings - "/mba/api/v1/bookings"
- getBookingOnId - "/mba/api/v1/bookings/:id"
- createBooking - "/mba/api/v1/bookings"
- updateBooking - "/mba/api/v1/bookings/:id"

```
const bookingController = require("../controllers/booking.controller");
const { authJwt, verifyBookingReqBody } = require("../middlewares");

module.exports = function (app) {
   app.get("/mba/api/v1/bookings", [authJwt.verifyToken], bookingController.getAllBookings);
   app.get("/mba/api/v1/bookings/:id", [authJwt.verifyToken], bookingController.getBookingOnId);
   app.post("/mba/api/v1/bookings", [authJwt.verifyToken,verifyBookingReqBody.validateBookingRequestBody],
bookingController.createBooking);
   app.put("/mba/api/v1/bookings/:id", [authJwt.verifyToken], bookingController.updateBooking);
}
```

Authentication for booking

There we have two types of authentication:

1. **VerifyToken:** We have already discussed this authJWT in middlewares. We will use verifyToken in all the Api's.
2. **validateBookingRequestBody:** https://github.com/Vishwa07dev/mba_backend/blob/session7/middlewares/verifyBookingReqBody.js

Here we are validating:

1. Validate the theater id is passed
2. Validate the theater id is valid
3. Validate if the theater exists
4. Validate the movie id is passed
5. Validate the movie id is valid
6. Validate if the movie id is present inside the theater
7. Validate if the timings is present
8. Validate if the no of seats is provided

Authentication for booking

```
const constants = require("../utils/constants");
var ObjectId = require('mongoose').Types.ObjectId;
const Theatre = require("../models/theatre.model");
validateBookingRequestBody = async (req, res, next) => {
   if (!req.body.theatreId) {
       return res.status(400).send({
           message: "Failed! theatreId is not provided !"
       });

   }
   if (!ObjectId.isValid(req.body.theatreId)) {
       return res.status(400).send({
           message: "Failed! theatreId is not valid format !"
       });
   }
   if (!req.body.movieId) {
       return res.status(400).send({
           message: "Failed! movieId is not provided !"
       });

   }
   if (!ObjectId.isValid(req.body.movieId)) {
       return res.status(400).send({
           message: "Failed! movieId is not valid format !"
       });
   }
   console.log(req.body.theatreId);
   const theatre = await Theatre.findOne({_id : req.body.theatreId});
    if(theatre == null){
          return res.status(400).send({
              message: "Failed! Theatre passed doesn't exist !"
          });
      }
      console.log(theatre);
      if(!theatre.movies.includes(req.body.movieId)){
          return res.status(400).send({
              message: "Failed! movieId passed is not available inside the theatre
!"
          });
      }
       if (!req.body.timing) {
          return res.status(400).send({
              message: "Failed! timing is not provided !"
          });

      }
       if (!req.body.noOfSeats) {
          return res.status(400).send({
              message: "Failed! number of seats is not provided !"
          });

      }
      next();

}
const verifyBookingReqBody = {
   validateBookingRequestBody : validateBookingRequestBody
};
module.exports = verifyBookingReqBody;
```

Create a New Booking

In this API, we are creating a Booking.
[Code Link] - https://github.com/Vishwa07dev/mba_backend/blob/session7/controllers/booking.controller.js

1. We are fetching the req body data and create the object which will be used to store data in db
2. We are creating the booking entry in the db and waiting for the promise using await.
3. On successful creation of booking we are responding back with success code 201

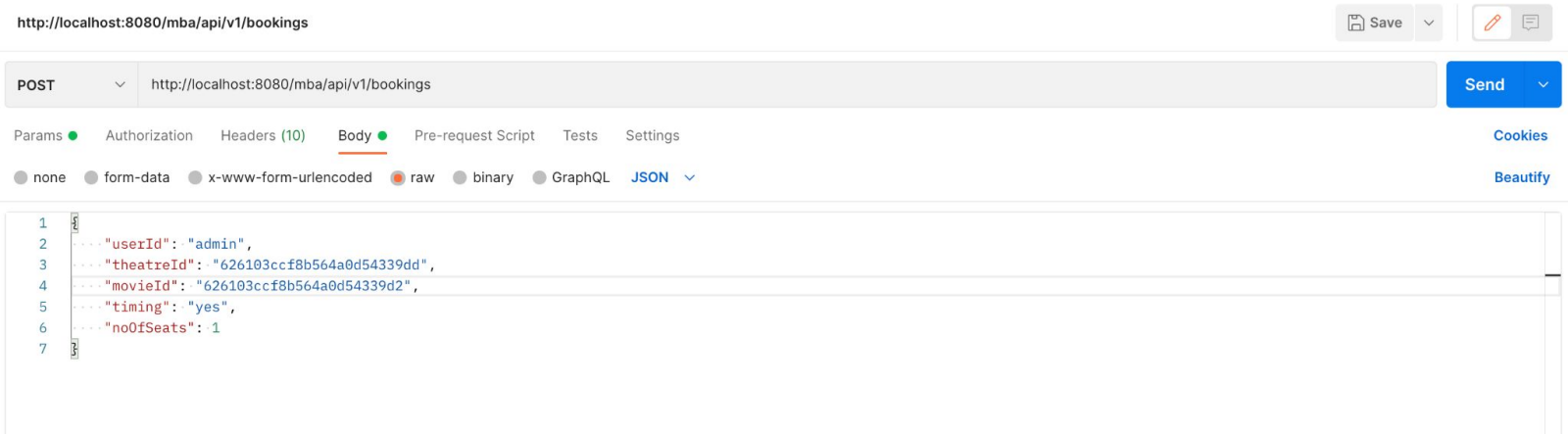
Here, we will use the booking schema of our database and create a booking by giving all the details as an object to create the function of the booking.
We will give below details to create a booking -

- userId
- threatreId
- movieId
- timing
- no of Seats

```
exports.createBooking = async (req, res) => {

   const user = await User.findOne({
       userId: req.userId
   })
   var bookingObject = {
       theatreId: req.body.theatreId,
       movieId: req.body.movieId,
       userId: user._id,
       timing: req.body.timing,
       noOfSeats: req.body.noOfSeats,
       totalCost: (req.body.noOfSeats *
constants.ticketPrice)
   }
```

```
   try {
       const booking = await
Booking.create(bookingObject);
       res.status(201).send(booking);
   } catch (err) {
       console.log(err);
       res.status(500).send({
           message: "Internal error while creating
the booking"
       })
   }


}
```

Request


http://localhost:8080/mba/api/v1/bookings
Save
POST
http://localhost:8080/mba/api/v1/bookings
Send
Params
Authorization
Headers (10)
Body
Pre-request Script
Tests
Settings
Cookies
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
JSON
Beautify
{
"userId": "admin",
"theatreId": "626103ccf8b564a0d54339dd",
"movieId": "626103ccf8b564a0d54339d2",
"timing": "yes",
"noOfSeats": 1
}

Response


Body
Cookies
Headers (7)
Test Results
Status: 201 Created
Time: 57 ms
Size: 542 B
Save Response
Pretty
Raw
Preview
Visualize
JSON
{
"theatreId": "626103ccf8b564a0d54339dd",
"movieId": "626103ccf8b564a0d54339d2",
"userId": "626774c90cbdd060a46869f3",
"timing": "yes",
"status": "IN_PROGRESS",
"noOfSeats": 1,
"totalCost": 150,
"_id": "626774cc0cbdd060a46869f6",
"createdAt": "2022-04-26T04:27:56.107Z",
"updatedAt": "2022-04-26T04:27:56.107Z",
"__v": 0
}

Update the booking based on the booking id

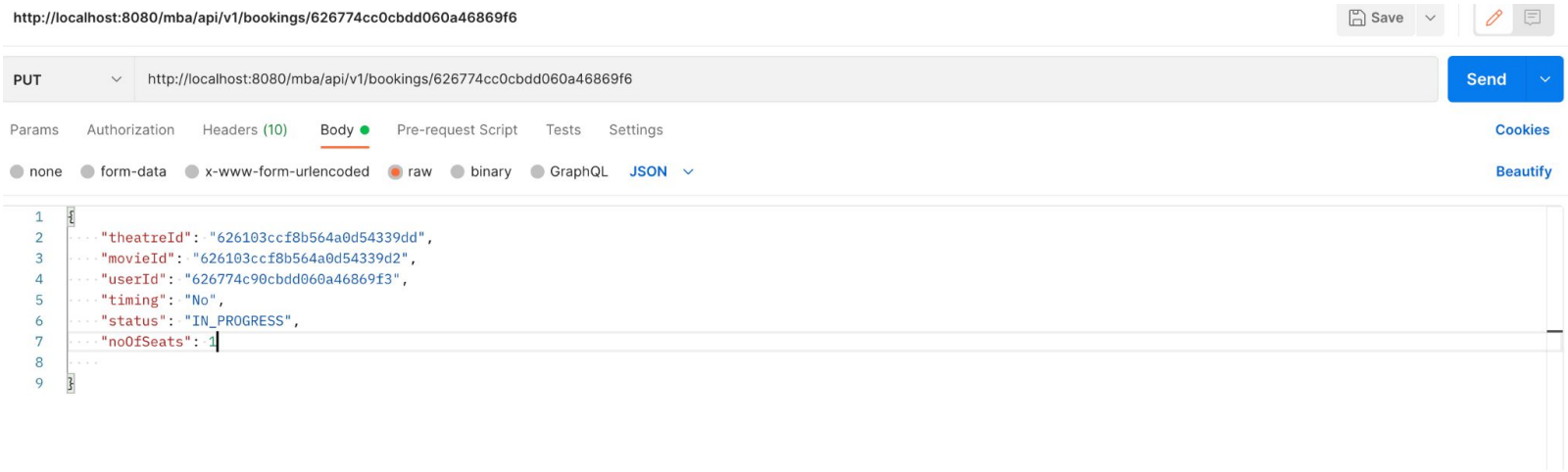
In this API, we will update a booking based on the booking id

[Code Link] -
https://github.com/Vishwa07dev/mba_backend/blob/session7/controllers/booking.controller.js

- Here, first, we will fetch the booking based on the booking id given in the request. We will use the findOne function of the booking schema to fetch the booking.
- If the booking does not exist, we will send an error message saying that “booking being updated doesn’t exist”
- If the booking exists, we will proceed with updating the booking
- We will fetch all the details of the booking from the request one by one.
- Once done, we will use the save function of the booking schema to save the updated booking in the database.

```
exports.updateBooking = async (req, res) => {

    const booking = await Booking.findOne({
        _id: req.params.id
    })
    booking.theatreId = req.body.theatreId != undefined ? req.body.theatreId : booking.theatreId;
    booking.movieId = req.body.movieId != undefined ? req.body.movieId : booking.movieId;
    booking.userId = req.body.userId != undefined ? req.body.userId : booking.userId;
    booking.timing = req.body.timing != undefined ? req.body.timing : booking.timing;
    booking.noOfSeats = req.body.noOfSeats != undefined ? req.body.noOfSeats : booking.noOfSeats;
    booking.totalCost = booking.noOfSeats * constants.ticketPrice;
    booking.status = req.body.status != undefined ? req.body.status : booking.status;

    try {
        const updatedBooking = await booking.save();
        res.status(201).send(updatedBooking);
    } catch (err) {
        console.log(err);
        res.status(500).send({
            message: "Internal error while updating the booking"
        })
    }

}
```

Request


http://localhost:8080/mba/api/v1/bookings/626774cc0cbdd060a46869f6
Save
PUT
http://localhost:8080/mba/api/v1/bookings/626774cc0cbdd060a46869f6
Send
Params
Authorization
Headers (10)
Body
Pre-request Script
Tests
Settings
Cookies
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
JSON
Beautify
{
"theatreId": "626103ccf8b564a0d54339dd",
"movieId": "626103ccf8b564a0d54339d2",
"userId": "626774c90cbdd060a46869f3",
"timing": "No",
"status": "IN_PROGRESS",
"noOfSeats": 1
}

Response


Body
Cookies
Headers (7)
Test Results
Status: 201 Created
Time: 23 ms
Size: 541 B
Save Response
Pretty
Raw
Preview
Visualize
JSON
{
"_id": "626774cc0cbdd060a46869f6",
"theatreId": "626103ccf8b564a0d54339dd",
"movieId": "626103ccf8b564a0d54339d2",
"userId": "626774c90cbdd060a46869f3",
"timing": "No",
"status": "IN_PROGRESS",
"noOfSeats": 1,
"totalCost": 150,
"createdAt": "2022-04-26T04:27:56.107Z",
"updatedAt": "2022-04-26T04:27:56.107Z",
"__v": 0
}

Get the Booking

In this API, we will get a list of the bookings.

Behaviour for Customer and Admin roles:
Customer: Only able to get his bookings
Admin: All the bookings

[Code Link] -
https://github.com/Vishwa07dev/mba_backend/blob/session7/controllers/booking.controller.js

Here, we will use the find function of the booking schema present in our database.
We also have used a query object here for the user Id.

```
exports.getAllBookings = async (req, res) => {

    const user = await User.findOne({
        userId: req.userId
    })

    const queryObj = {};
    if (user.userType == constants.userTypes.admin) {
    } else {
        queryObj._id = user._id
    }

    const bookings = await Booking.find(queryObj);

    res.status(200).send(bookings);

}
```

Request – Get All the booking

Response


Body
Cookies
Headers (7)
Test Results
Status: 200 OK
Time: 17 ms
Size: 538 B
Save Response
Pretty
Raw
Preview
Visualize
JSON
[
{
"_id": "626774cc0cbdd060a46869f6",
"theatreId": "626103ccf8b564a0d54339dd",
"movieId": "626103ccf8b564a0d54339d2",
"userId": "626774c90cbdd060a46869f3",
"timing": "No",
"status": "IN_PROGRESS",
"noOfSeats": 1,
"totalCost": 150,
"createdAt": "2022-04-26T04:27:56.107Z",
"updatedAt": "2022-04-26T04:27:56.107Z",
"__v": 0
}
]

Get the booking based on the booking id

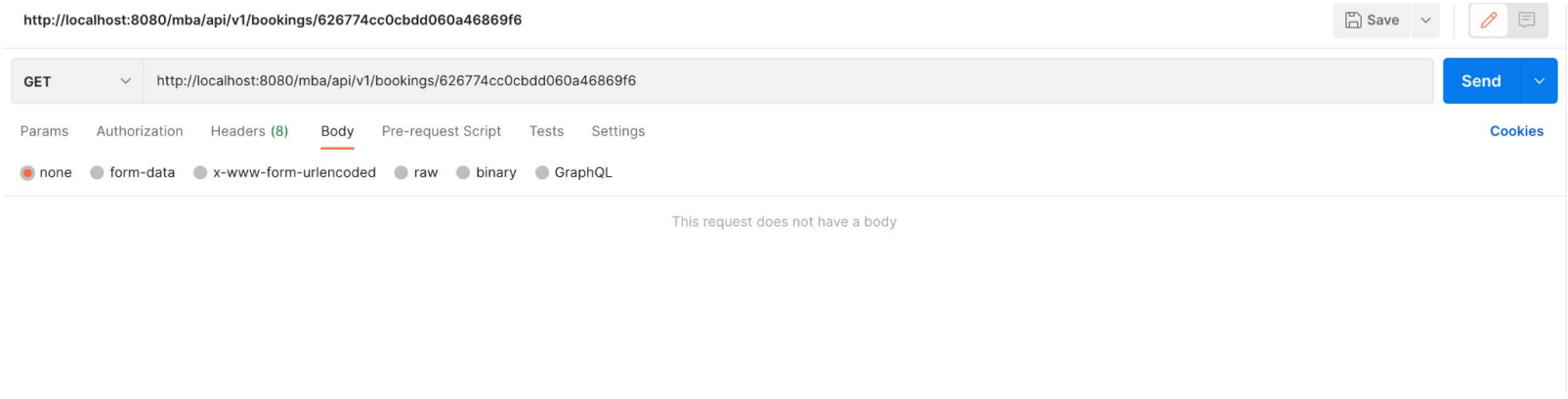
In this API, we will get a booking based on the booking id

[Code Link] - https://github.com/Vishwa07dev/mba_backend/blob/session7/controllers/booking.controller.js

Here, we will use the findOne function of the booking schema present in our database.
We will pass booking id as a parameter to the findOne function of the booking schema. This will give us the booking having booking id equals to the passed one

```
exports.getBookingOnId = async (req, res) => {
    try {
        const bookings = await Booking.findOne({ _id: req.params.id });
        res.status(200).send(bookings);
    } catch (err) {
        console.log(err.message);
        res.status(500).send({
            message: "Internal error while searching for the booking"
        })
    }
}
```

Request


http://localhost:8080/mba/api/v1/bookings/626774cc0cbdd060a46869f6
Save
GET
http://localhost:8080/mba/api/v1/bookings/626774cc0cbdd060a46869f6
Send
Params
Authorization
Headers (8)
Body
Pre-request Script
Tests
Settings
Cookies
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
This request does not have a body

Response


Body
Cookies
Headers (7)
Test Results
Status: 200 OK
Time: 18 ms
Size: 536 B
Save Response
Pretty
Raw
Preview
Visualize
JSON
{
"_id": "626774cc0cbdd060a46869f6",
"theatreId": "626103ccf8b564a0d54339dd",
"movieId": "626103ccf8b564a0d54339d2",
"userId": "626774c90cbdd060a46869f3",
"timing": "No",
"status": "IN_PROGRESS",
"noOfSeats": 1,
"totalCost": 150,
"createdAt": "2022-04-26T04:27:56.107Z",
"updatedAt": "2022-04-26T04:27:56.107Z",
"__v": 0
}

Data Models

- **Payment Resource**
- bookingId: It is ObjectID and reference from Booking schema which is required.
- Amount: It is a number type and required.
- status:It is a string which is required with default as FAILED.
- createdAt: It is a date type and immutable.
- updatedAt: It is a date type.

Payment Resource Application - CRUD Operations

[Code Link] - https://github.com/Vishwa07dev/mba_backend/blob/session7/routes/payment.routes.js

- getAllPayments - "/mba/api/v1/payments"
- getPaymentOnId - "/mba/api/v1/payments/:id"
- createPayment - "/mba/api/v1/payments"

```
const paymentController = require("../controllers/payment.controller");
const { authJwt , verifyPaymentReqBody} = require("../middlewares");
module.exports = function (app) {
   app.get("/mba/api/v1/payments", [authJwt.verifyToken], paymentController.getAllPayments);
   app.get("/mba/api/v1/payments/:id", [authJwt.verifyToken], paymentController.getPaymentOnId);
   app.post("/mba/api/v1/payments", [authJwt.verifyToken,
verifyPaymentReqBody.validatePaymentRequestBody], paymentController.createPayment);

}
```

Authentication for payment

There we have two types of authentication:

1. **VerifyToken:** We have already discussed this authJWT in middlewares. We will use verifyToken in all the Api's.
2. **validatePaymentRequestBody:** https://github.com/Vishwa07dev/mba_backend/blob/session7/middlewares/verifyPaymentReqBody.js

Here we are validating:

1. Validate the theater id is passed
2. Validate the booking id is valid
3. Validate if the booking exists
4. Check for the amount

Authentication for payments request body

```
const constants = require("../utils/constants");
var ObjectId = require('mongoose').Types.ObjectId;
const Booking = require("../models/booking.model");


validatePaymentRequestBody = async (req, res, next) => {

   //Validate the booking id is passed
   if (!req.body.bookingId) {
       return res.status(400).send({
           message: "Failed! bookingId is not provided !"
       });

   }
   //Validate the booking id is valid
   if (!ObjectId.isValid(req.body.bookingId)) {
       return res.status(400).send({
           message: "Failed! bookingId is not valid format
!"
       });

   }
```

```
 //Validte if the booking exists
   const booking = await Booking.findOne({_id
:req.body.bookingId });
   if(booking==null){
       return res.status(400).send({
           message: "Failed! Booking Id passed doesn't
exist !"
       });
   }
    /**
    * Check for the amount
    */
    if (req.body.amount < booking.totalCost) {
       return res.status(400).send({
           message: "Can't do the payment as the payment
amount is less than the booking amount"
       })
   }
   next();

}
const verifyPaymentReqBody = {
   validatePaymentRequestBody : validatePaymentRequestBody
};

module.exports = verifyPaymentReqBody;
```

Create a New Payment

In this API, we are creating a Payment.
[Code Link] - https://github.com/Vishwa07dev/mba_backend/blob/session7/controllers/payment.controller.js

1. We will fetch the bookingId from request and get the booking document form Db related to the bookingId.
2. Then check if the booking time is greater than 5 minutes or not? If yes, then we will updated booking status to “expired” and return the message "Can't do the payment as the booking is delayed and expired"
3. We are fetching the req body data and create the object which will be used to store data in db
4. We are creating the payment entry in the db and waiting for the promise using await.
5. On the successful payment, we need to go and update the status of the Booking and finally return with the status code 201

Here, we will use the payment schema of our database and create a payment by giving all the details as an object to create the function of the payment.
We will give below details to create a payment -

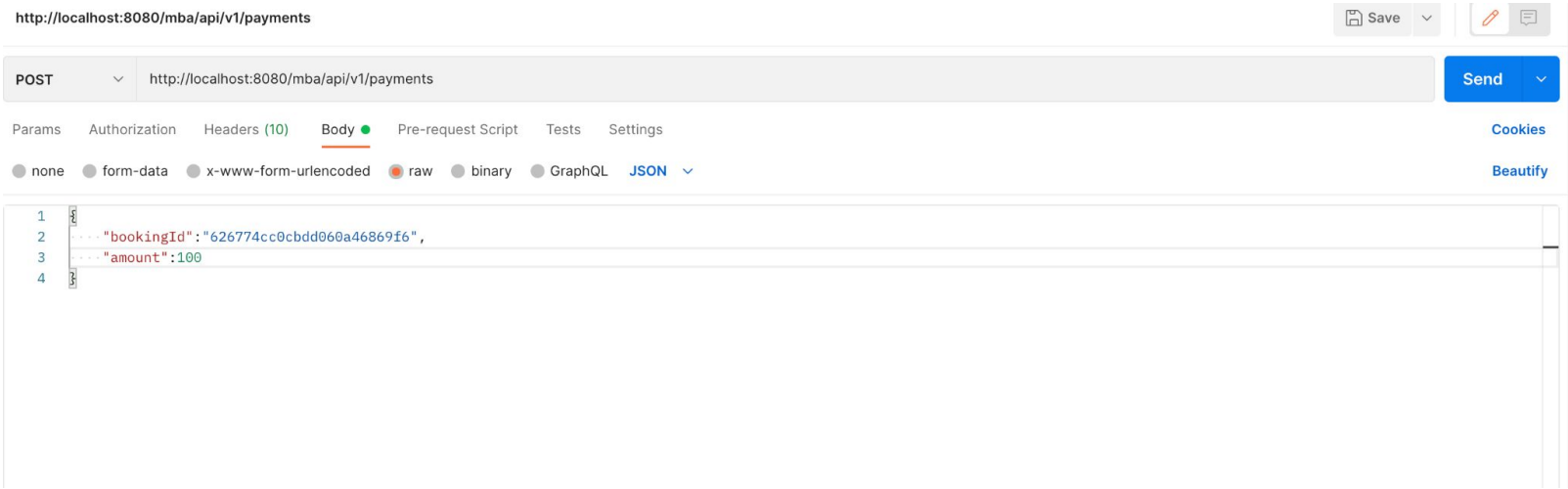
- bookingId
- ammount

Create a New Payment

```
exports.createPayment = async (req, res) => {

    const booking = await Booking.findOne({
        _id: req.body.bookingId
    });

    var bookingTime = booking.createdAt;
    var currentTime = Date.now();

    var minutes = Math.floor(((currentTime -
bookingTime) / 1000) / 60);

    if (minutes > 5) {
        booking.status =
constants.bookingStatus.expired;
        await booking.save();
        return res.status(200).send({
            message: "Can't do the payment as the
booking is delayed and expired"
        })
    }
```

```
 var paymentObject = {
        bookingId: req.body.bookingId,
        amount: req.body.amount,
        status: constants.paymentStatus.success,
    }
    try {
        const payment = await
Payment.create(paymentObject);
        /**
         * Update the booking status
         */
        booking.status =
constants.bookingStatus.completed;
        await booking.save();
        return res.status(201).send(payment);
    } catch (err) {
        console.log(err);
        res.status(500).send({
            message: "Internal error while creating the
booking"
        })
    }

}
```

Request


http://localhost:8080/mba/api/v1/payments
Save
POST
http://localhost:8080/mba/api/v1/payments
Send
Params
Authorization
Headers (10)
Body
Pre-request Script
Tests
Settings
Cookies
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
JSON
Beautify
{
"bookingId":"626774cc0cbdd060a46869f6",
"amount":100
}

Response due to delay

Response in 5 minute of booking

Get the Payment

In this API, we will get a list of the payments.
Behaviour for Customer and Admin roles:
Customer: Only able to get his payment
Admin: All the payments
[Code Link] - https://github.com/Vishwa07dev/mba_backend/blob/session7/controllers/payment.controller.j
Here, we will use the find function of the payment schema present in our database.

- We also have used a query object here for the user Id, bookingId.

```
exports.getAllPayments = async (req, res) => {
   const queryObj = {}
   if (user.userType == constants.userTypes.admin) {
   } else {
       const bookings = await Booking.find({
           _id: req.userId
       });
       const bookingIds = bookings.map(b => b._id);
       console.log(bookingIds);
       queryObj._id = { $in: bookingIds }
   };
   const payments = await Payment.find(queryObj);

   res.status(200).send(payments);
}
```

Request – Get All the payment

Response due to delay

Get the payment based on the payment id

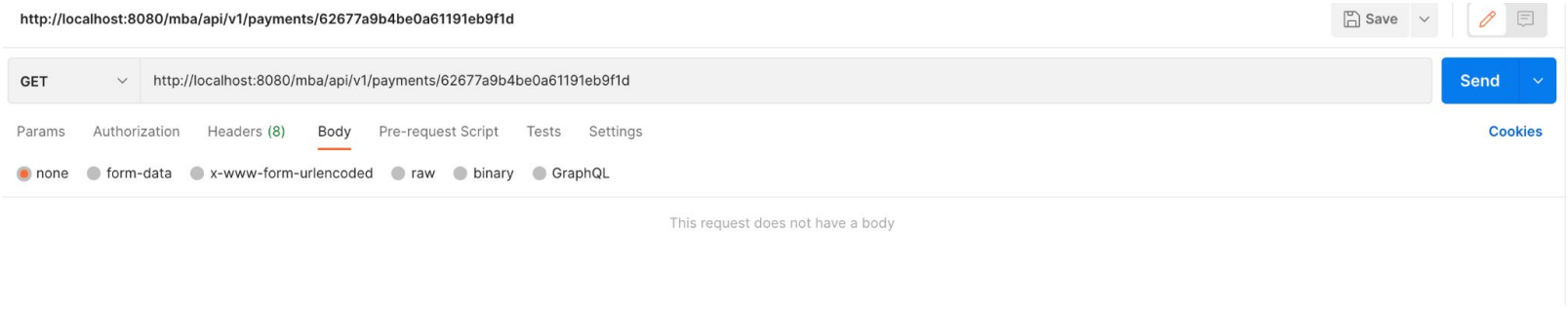
In this API, we will get a booking based on the booking id

[Code Link] - https://github.com/Vishwa07dev/mba_backend/blob/session7/controllers/payment.controller.js

Here, we will use the findOne function of the payment schema present in our database.
We will pass payment id as a parameter to the findOne function of the payment schema. This will give us the payment having payment id equals to the passed one

```
exports.getPaymentOnId = async (req, res) => {
    try {
        const payments = await Payment.findOne({ _id: req.params.id });
        res.status(200).send(payments);
    } catch (err) {
        console.log(err.message);
        res.status(500).send({
            message: "Internal error while searching for the payments"
        })
    }
}
```

Request


http://localhost:8080/mba/api/v1/payments/62677a9b4be0a61191eb9f1d
Save
GET
http://localhost:8080/mba/api/v1/payments/62677a9b4be0a61191eb9f1d
Send
Params
Authorization
Headers (8)
Body
Pre-request Script
Tests
Settings
Cookies
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
This request does not have a body

Response


Body
Cookies
Headers (7)
Test Results
Status: 200 OK
Time: 10 ms
Size: 427 B
Save Response
Pretty
Raw
Preview
Visualize
JSON
{
"_id": "62677a9b4be0a61191eb9f1d",
"bookingId": "626774cc0cbdd060a46869f6",
"amount": 100,
"status": "SUCCESS",
"createdAt": "2022-04-26T04:52:43.247Z",
"updatedAt": "2022-04-26T04:52:43.247Z",
"__v": 0
}

MCQ:

1. Which function is used to get current date time?

A. Date.now()

B. Date.current()

C. Both A and B

D. None

Answer: A

2. What key to add in the schema to set default value?

A. placeholder

B. default

C. Both A and B

D. None

Answer: B

3. What authentications are being used while create payment?

A. verifyToken

B. validatePaymentRequestBody

C. None

D. Both A and B

Answer: D

4. What authentications are being used while getting payment ?

A. verifyToken

B. validatePaymentRequestBody

C. None

D. Both A and B

Answer: A

5. Which method is used to create the database?

A. save

B. post

C. create

D. All of the above

Answer: C

Practice Code

- Write a discount.model.js with the schema:
  discountCode: String Required
  PercentageOfDiscount: Number Required
  Deadline: Date Required
- Write the logic to give discount while payment using discount code.

Thank You!