

Adding Middlewares for validation

Relevel
by Unacademy



Agenda

- Creation of Custom middleware to validate the request body for user registration
- Creation of Custom middleware to validate the request body for movie resource
- Creation of Custom middleware to validate the request body for theatre resource
- Updating existing routes with middleware validations.

Creation of Custom middleware to validate the request body for user registration

For the Movie Booking App, we will create a middleware to validate the request body of the user for registration/signup. Any valid user should be able to register itself so we will keep a check on the properties such as name, userid and make sure the user type is valid as well i.e. it is either CLIENT, CUSTOMER or ADMIN.

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

[Code Link] -

https://github.com/Vishwa07dev/mba_backend/blob/session5/middlewares/verifyUserReqBody.js

We will create middlewares for creation/registration of a user first and then for updating user details.

Let's name this middleware as `verifyUserReqBody.js`

1. Create user validation middleware for creation/registration of a user

- This middleware will have multiple validations for user registration that is discussed below, we will name it as

```
validateUserRequestBody = async (req, res, next) => {}
```

Let's name this middleware as verifyUserReqBody.js

2. Validating the userName

- Here, we will check if the user has provided its name or not
- If the user's name is empty or the field is not itself given in the request then we will send a message saying

“Failed! Username is not provided !”

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

Let's name this middleware as verifyUserReqBody.js

3. Validating the userId

- Here, we will check if the user has provided userId or not
- If the userId is not given in the request then we will send a message saying

“Failed! UserId is not provided !”

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

Let's name this middleware as verifyUserReqBody.js

4. Validating if user already exists with same userId

- Here, we will check if there is a user present with the same userId that another new user is trying to sign up with
- If the userId already exists then we will send a message saying

"Failed! Userid already exists!"

```
const user = await User.findOne({ userId: req.body.userId });  
if (user != null) {  
  console.log("Inside this");  
  res.status(400).send({  
    message: "Failed! Userid already exists!"  
  });  
  return;  
}
```

Let's name this middleware as verifyUserReqBody.js

5. Validating the email Id

- Here, we will check if a user has provided a valid email id
- We fetch the email id from request body and pass it as parameters and call `isValidEmail(req.body.email)` method which if is a falsy value we will send a message saying

"Failed! Email is not valid!"

```
//Validating the email Id
if (!isValidEmail(req.body.email)) {
  res.status(400).send({
    message: "Failed! Email is not valid!"
  });
  return;
}
```


Let's name this middleware as verifyUserReqBody.js

Below is the isValidEmail() method which tries to match the email string with the regular expression.

```
const isValidEmail = (email) => {  
  return String(email)  
    .toLowerCase()  
    .match(  

```

```
/^((([^<>() [\]\.\.,;:\s@"]+(\.[^<>() [\]\.\.,;:\s@"]+)*|("[.+"])|@([\d]{1,3}\.[\d]{1,3}\.[\d]{1,3}\.[\d]{1,3}\.|\)|([a-zA-Z\d-0-9]+\.)+[a-zA-Z]{2,}))$/  
    );  
};
```

Let's name this middleware as verifyUserReqBody.js

6. Validating if user already exists with same email id

- Here, we will check if there is a user present with the same email id that another new user is trying to sign up with
- If the email id already exists then we will send a message saying

"Failed! Email already exists!"

```
const email = await User.findOne({ email: req.body.email });  
if (email !== null) {  
  res.status(400).send({  
    message: "Failed! Email already exists!"  
  });  
  return;  
}
```

Let's name this middleware as `verifyUserReqBody.js`

7. Validating the `userType`

- Here, we do not want the `userType` to strictly be `ADMIN` but we will check if the `userType` is either `CLIENT`, `CUSTOMER` or `ADMIN`
- If the `userType` is anything other than `CLIENT`, `CUSTOMER` or `ADMIN` then we will send a message saying
"UserType provided is invalid. Possible values `CUSTOMER` | `CLIENT` | `ADMIN` "
- We have imported the constant values that have `userTypes`

```
const constants = require("../utils/constants");  
userTypes: {  
  customer: 'CUSTOMER',  
  client: 'CLIENT',  
  admin: 'ADMIN'  
},
```

Let's name this middleware as verifyUserReqBody.js

```
//Validating the user type
const userType = req.body.userType;
const userTypes = [constants.userTypes.customer,
constants.userTypes.client, constants.userTypes.admin]
if (userType && !userTypes.includes(userType)) {
    res.status(400).send({
        message: "UserType provided is invalid. Possible values
CUSTOMER | CLIENT | ADMIN "
    });
    return;
}
```

- Once the User has passed all the validation checks it will get registered.

Updating existing auth.routes.js for validating User registration:

Let's make changes to the required routes with the middleware validations -

[Code Link] -

https://github.com/Vishwa07dev/mba_backend/blob/session5/routes/auth.routes.js

- We will now update the existing auth.routes.js and add the verifyUserReqBody.validateUserRequestBody middleware validations as below.

```
app.post("/mba/api/v1/auth/signup",  
[verifyUserReqBody.validateUserRequestBody], authController.signup);
```

Requests/Response for various scenarios:

Let's see the valid request and response for user registration first and then the different validation failures.

Valid Signup Request:



Requests/Response for various scenarios:

Valid Signup Response:



The screenshot displays the 'Body' tab of a web browser's developer tools. The response is a JSON object representing a successful user signup. The status is 201 Created, with a response time of 117 ms and a size of 431 B. The JSON data is as follows:

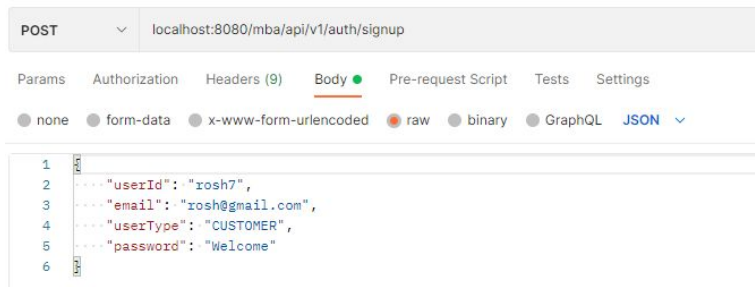
```
1 {
2   "name": "Roshan Kumar",
3   "userId": "rosh7",
4   "email": "rosh@gmail.com",
5   "userTypes": "CUSTOMER",
6   "userStatus": "APPROVED",
7   "createdAt": "2022-04-11T14:32:06.542Z",
8   "updatedAt": "2022-04-11T14:32:06.542Z"
9 }
```

Requests/Response for various scenarios:

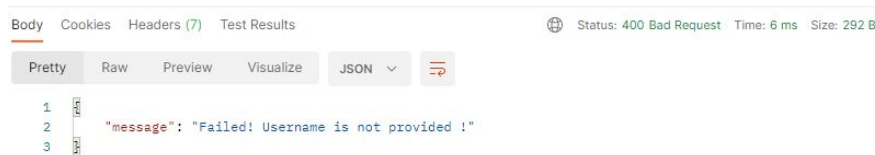
Validation Failures:

1. Validating the userName

Request:



Response:

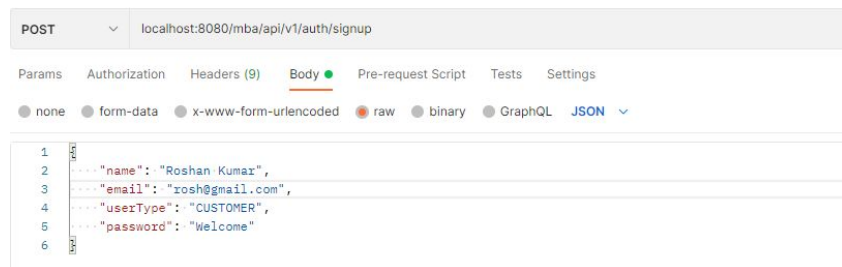


Requests/Response for various scenarios:

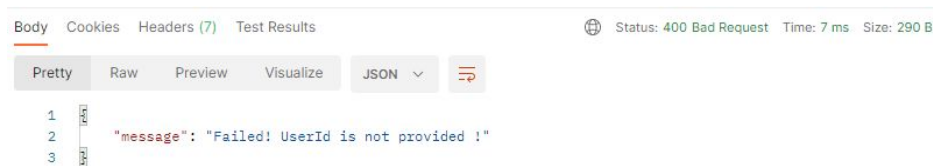
Validation Failures:

2. Validating the userId

Request:



Response:



Requests/Response for various scenarios:

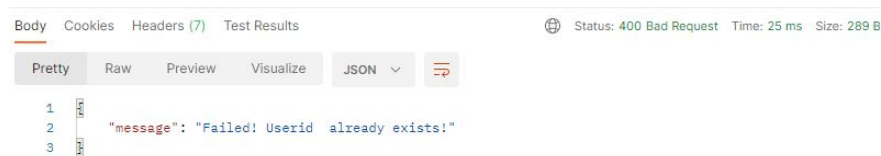
Validation Failures:

3. Validating if user already exists with same userId

Request:



Response:

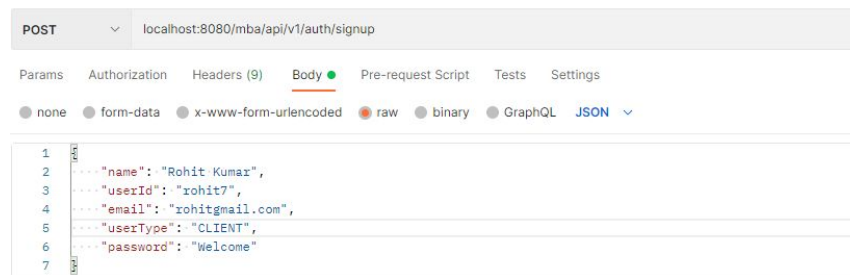


Requests/Response for various scenarios:

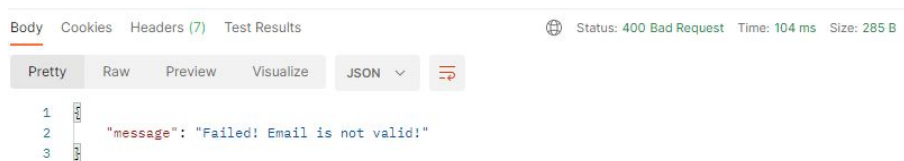
Validation Failures:

4. Validating the email Id

Request:



Response:

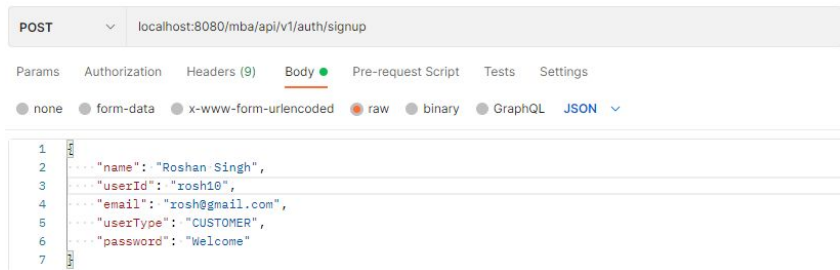


Requests/Response for various scenarios:

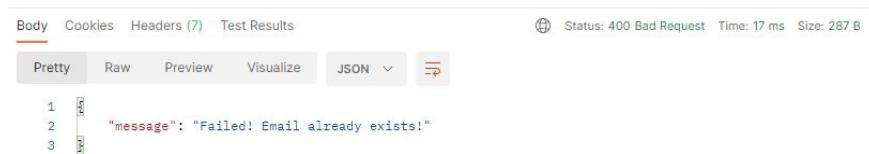
Validation Failures:

5. Validating if user already exists with same email id

Request:



Response:



Requests/Response for various scenarios:

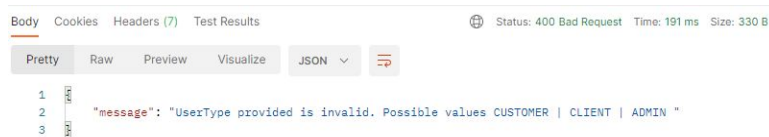
Validation Failures:

6. Validating the userType

Request:



Response:



Requests/Response for various scenarios:

As we have created validations for user registration we will now add validations for updating user details in our `verifyUserReqBody.js`

1. Create user validation middleware for updating user details

- This middleware will have validations for updating user status and type that is discussed below, we will name it as

```
validateUserStatusAndUserType = async (req, res, next) => {}
```

Requests/Response for various scenarios:

2. Validating the userType

- It is the same validation for userType that we have done while user registration
- We are not allowing the user to update the type of user as anything other than CLIENT, CUSTOMER or ADMIN, when the user tries to do so we will send a message saying

"UserType provided is invalid. Possible values CUSTOMER | CLIENT | ADMIN "

```
//Validating the user type
const userType = req.body.userType;
const userTypes = [constants.userTypes.customer,
constants.userTypes.client, constants.userTypes.admin]
if (userType && !userTypes.includes(userType)) {
    res.status(400).send({
message: "UserType provided is invalid. Possible values CUSTOMER | CLIENT
| ADMIN "
    });
    return;
}
```

Requests/Response for various scenarios:

3. Validating the userStatus

- Here, we make sure that the userStatus is either PENDING, APPROVED or REJECTED
- If the userStatus is anything other than PENDING, APPROVED or REJECTED then we will send a message saying

"UserStatus provided is invalid. Possible values PENDING | APPROVED | REJECTED "

- We have imported the constant values that have userStatus

```
const constants = require("../utils/constants");  
userStatus: {  
  pending: 'PENDING',  
  approved: 'APPROVED',  
  rejected: 'REJECTED'  
}
```


Requests/Response for various scenarios:

```
//validating the userStatus
const userStatus = req.body.userStatus;
const userSatuses = [constants.userStatus.pending, constants.userStatus.approved,
constants.userStatus.rejected]
if (userStatus && !userSatuses.includes(userStatus)) {
  res.status(400).send({
    message: "UserStatus provided is invalid. Possible values PENDING | APPROVED | REJECTED "
  });
  return;
}
```

Updating existing user.routes.js for validating updates to User Status and Type:

Let's make changes to the required routes with the middleware validations -

[Code Link] -

https://github.com/Vishwa07dev/mba_backend/blob/session5/routes/user.routes.js

- We will now update the existing user.routes.js and add the verifyUserReqBody.validateUserStatusAndUserType middleware validations as below.

```
app.put("/crm/api/v1/users/:userId", [authJwt.verifyToken,  
authJwt.isAdmin, verifyUserReqBody.validateUserStatusAndUserType],  
UserController.updateUser);  
}
```

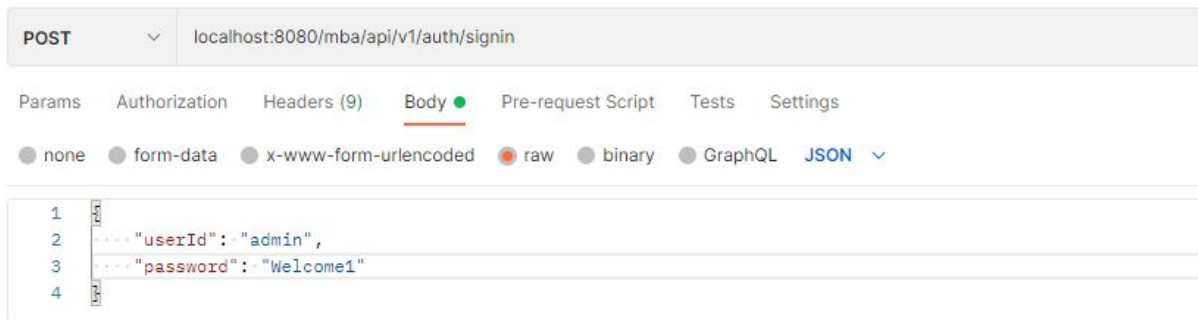
Requests/Response for various scenarios:

Let's see the valid request and response for updating user details first and then the validation failures.

First we need to fetch the x-access-token for admin access to update any field of user

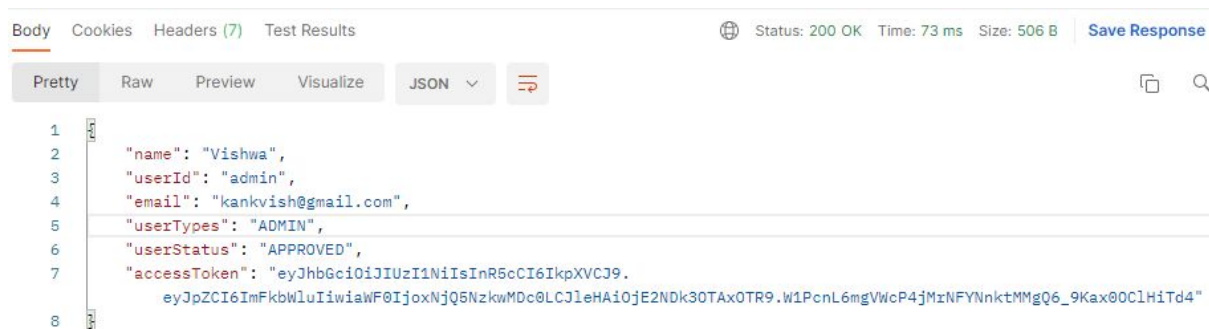
Sign in as admin

Admin Sign in request:



Requests/Response for various scenarios:

Admin Sign in response:



The screenshot shows the 'Body' tab of a web browser's developer tools. The response is a JSON object with the following fields: 'name', 'userId', 'email', 'userTypes', 'userStatus', and 'accessToken'. The 'accessToken' field contains a long alphanumeric string.

```
1 {
2   "name": "Vishwa",
3   "userId": "admin",
4   "email": "kankvish@gmail.com",
5   "userTypes": "ADMIN",
6   "userStatus": "APPROVED",
7   "accessToken": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZiZCI6ImFkbWluIiwiaWF0IjoxNjQ5NzkwMDc0LCJleHAiOjE2NDk3OTAxOTR9.W1PcnL6mgVwcP4jMzNFYNnktMMgQ6_9Kax00ClHiTd4"
8 }
```

Requests/Response for various scenarios:

Valid update userType and userStatus Request:

Request Header:

PUT localhost:8080/crm/api/v1/users/rosh7

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

Headers 8 hidden

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

Request Body:

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

Save

PUT localhost:8080/crm/api/v1/users/rosh7

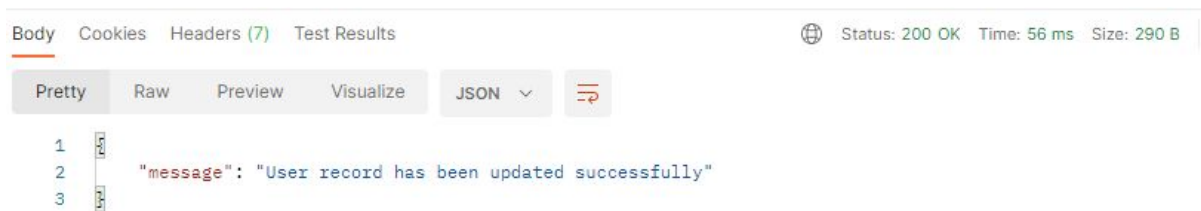
Params Authorization Headers (10) Body Pre-request Script Tests Settings

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

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

Requests/Response for various scenarios:

Valid Signup Response:



Requests/Response for various scenarios:

Validation Failures:

1. Validating the userType

Request:

PUT localhost:8080/crm/api/v1/users/rosh7

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

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

```
1 {
2   "name": "Roshan Kumaz",
3   "userType": "SELLER",
4   "userStatus": "APPROVED"
5 }
```

Response:

Body Cookies Headers (7) Test Results Status: 400 Bad Request Time: 12 ms Size: 330 B

Pretty Raw Preview Visualize JSON

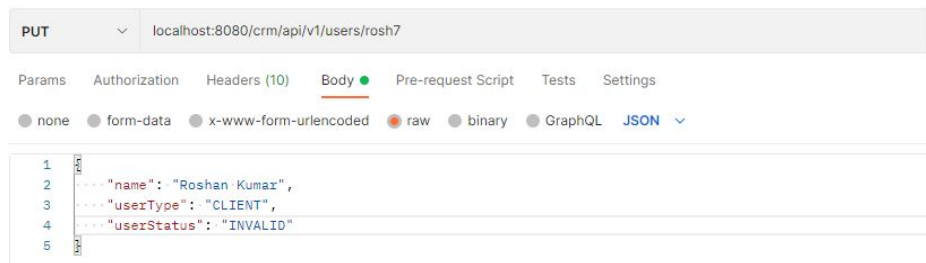
```
1 {
2   "message": "UserType provided is invalid. Possible values CUSTOMER | CLIENT | ADMIN "
3 }
```

Requests/Response for various scenarios:

Validation Failures:

2. Validating the userStatus

Request:



Response:



Creation of Custom middleware to validate the request body for movie resource

We will now create a middleware to validate the request body for movie resources. Any movie with valid details should be registered. So we will keep a check on the properties such as movie name, release date, director name and movie release status.

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

[Code Link] -

https://github.com/Vishwa07dev/mba_backend/blob/session5/middlewares/verifyMovieReqBody.js

Creation of Custom middleware to validate the request body for movie resource

Let's name this middleware as `verifyMovieReqBody.js`

1. Create validation middleware for movie request body

- This middleware will have multiple validations for movie request that is discussed below, we will name it as

```
validateMovieRequestBody = async (req, res, next) => {}
```

Creation of Custom middleware to validate the request body for movie resource

2. Validating the movie name

- Here, we will check if the user has provided its movie name or not
- If the movie name is empty or the field is not itself given in the request then we will send a message saying

"Failed! Movie name is not provided !"

```
//Validate the movie name
if (!req.body.name) {
    return res.status(400).send({
        message: "Failed! Movie name is not provided !"
    });
}
```

Creation of Custom middleware to validate the request body for movie resource

3. Validating the movie releaseStatus

- Here, we will check if the user has provided movie releaseStatus or not
- If the releaseStatus is not given in the request then we will send a message saying

"Failed! Movie release status is not provided !"

```
//validate the movie status
    if (!req.body.releaseStatus) {
        return res.status(400).send({
            message: "Failed! Movie release status is not provided !"
        });
    }
```

Creation of Custom middleware to validate the request body for movie resource

4. Validating if correct value of movie releaseStatus is provided

- Here, we do not want the value of releaseStatus to be anything other than UNRELEASED, RELEASED or BLOCKED
- If the releaseStatus is anything other than UNRELEASED, RELEASED or BLOCKED then we will send a message saying

"Movie release status provided is invalid. Possible values UNRELEASED | RELEASED | BLOCKED "

- We have imported the constant values that have releaseStatus

Creation of Custom middleware to validate the request body for movie resource

```
const constants = require("../utils/constants");
```

```
releaseStatus: {  
  unreleased: 'UNRELEASED',  
  released: 'RELEASED',  
  blocked: 'BLOCKED'  
},
```

Creation of Custom middleware to validate the request body for movie resource

```
//Checking for the correct value of status
const releaseStatus = req.body.releaseStatus;
const releaseStatusTypes = [constants.releaseStatus.unreleased,
constants.releaseStatus.released, constants.releaseStatus.blocked];
if (!releaseStatusTypes.includes(releaseStatus)) {
  return res.status(400).send({
    message: "Movie release status provided is invalid. Possible
values UNRELEASED | RELEASED | BLOCKED "
  });
}
```

Creation of Custom middleware to validate the request body for movie resource

5. Validating the movie releaseDate

- Here, we will check if the movie releaseDate is provided or not
- If the releaseDate is not given in the request then we will send a message saying

"Failed! Movie release date is not provided !"

```
//validate the release date
if (!req.body.releaseDate) {
  return res.status(400).send({
    message: "Failed! Movie release date is not provided !"
  });
}
```


Creation of Custom middleware to validate the request body for movie resource

6. Validating the director

- Here, we will check if a user has provided the director of the movie or not
- If the director is not given in the request then we will send a message saying

"Failed! Movie director is not provided !"

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

- Once all the validation checks are passed, the movie will get registered.

Updating existing movie.routes.js for validating movie registration and updating of movie details:

Let's make changes to the required routes with the middleware validations -

[Code Link] -

https://github.com/Vishwa07dev/mba_backend/blob/session5/routes/movie.routes.js

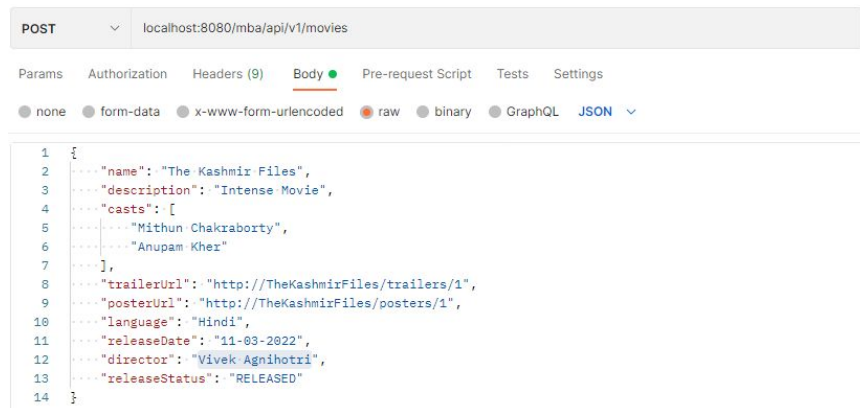
- We will now update the existing movie.routes.js and add the `verifyMovieReqBody.validateMovieRequestBody` middleware validations as below.

```
app.post("/mba/api/v1/movies/", [verifyMovieReqBody.  
validateMovieRequestBody], movieController.createMovie);  
app.put("/mba/api/v1/movies/:id", [verifyMovieReqBody.  
validateMovieRequestBody], movieController.updateMovie);
```

Requests/Response for various scenarios:

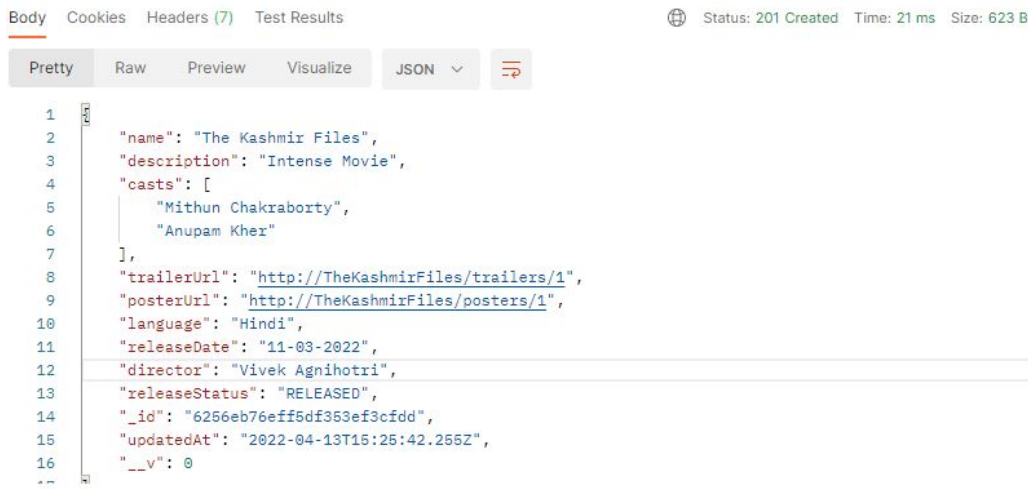
Let's see the valid request and response for movie registration and updation first and then the different validation failures.

Valid Movie creation/registration Request:



Requests/Response for various scenarios:

Valid Movie creation/registration Response:

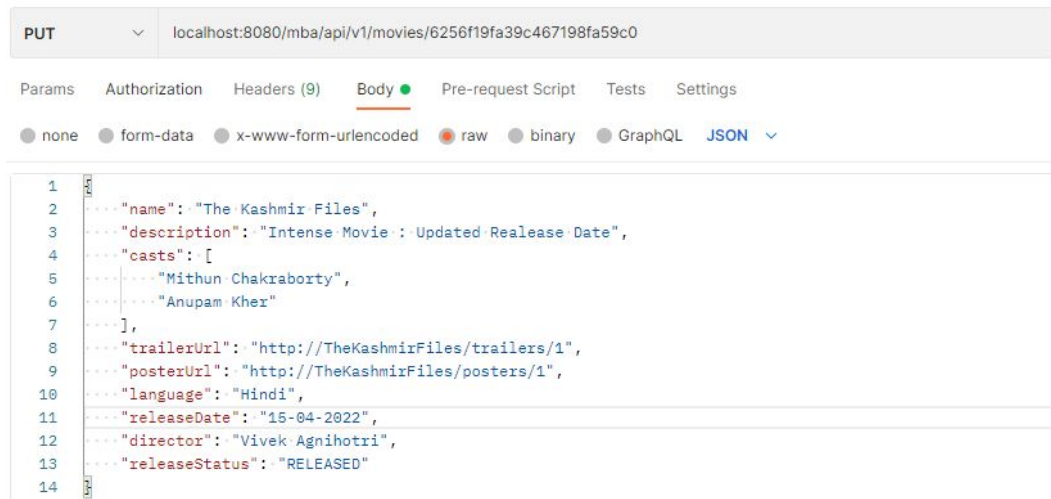


The screenshot displays the 'Body' tab of a web browser's developer tools. The response is a JSON object representing a movie registration. The status bar at the top right indicates 'Status: 201 Created', 'Time: 21 ms', and 'Size: 623 B'. The JSON is formatted in 'Pretty' mode. The response includes fields for movie name, description, cast members, trailer URL, poster URL, language, release date, director, release status, a unique ID, an updated timestamp, and a version number.

```
1 {
2   "name": "The Kashmir Files",
3   "description": "Intense Movie",
4   "casts": [
5     "Mithun Chakraborty",
6     "Anupam Kher"
7   ],
8   "trailerUrl": "http://TheKashmirFiles/trailers/1",
9   "posterUrl": "http://TheKashmirFiles/posters/1",
10  "language": "Hindi",
11  "releaseDate": "11-03-2022",
12  "director": "Vivek Agnihotri",
13  "releaseStatus": "RELEASED",
14  "_id": "6256eb76eff6df353ef3cfdd",
15  "updatedAt": "2022-04-13T15:25:42.255Z",
16  "__v": 0
}
```

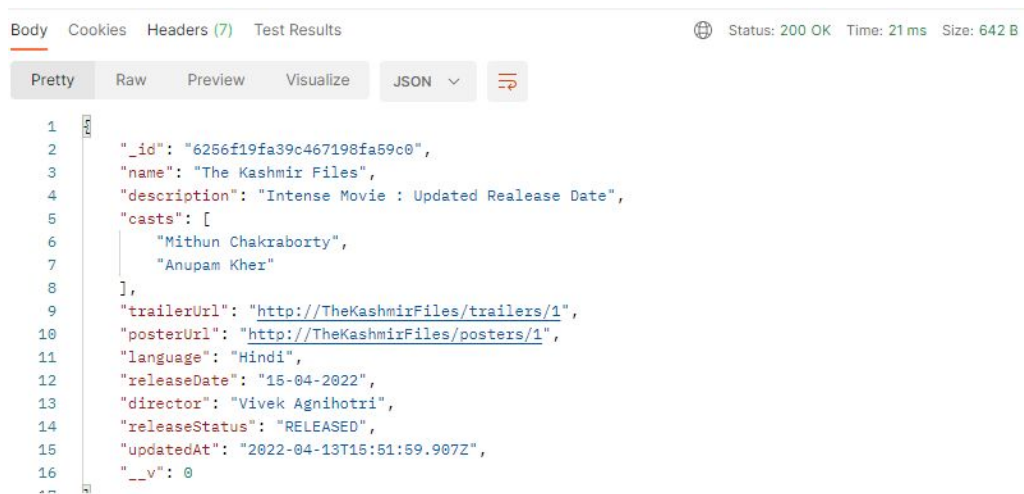
Requests/Response for various scenarios:

Valid Movie updation Request:



Requests/Response for various scenarios:

Valid Movie updation Response:



The screenshot shows a REST client interface with a response body tab selected. The response is a JSON object representing a movie update. The status is 200 OK, the time taken is 21 ms, and the size is 642 B. The JSON data is as follows:

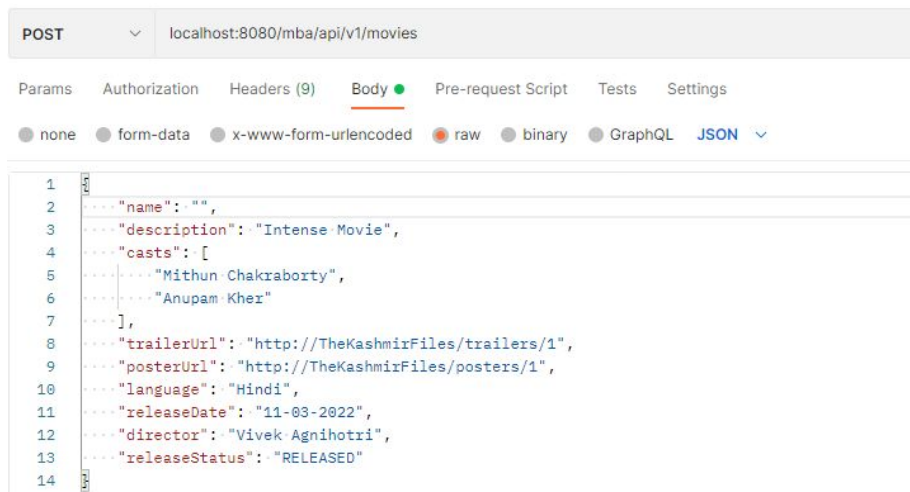
```
1 {
2   "_id": "6256f19fa39c467198fa59c0",
3   "name": "The Kashmir Files",
4   "description": "Intense Movie : Updated Realease Date",
5   "casts": [
6     "Mithun Chakraborty",
7     "Anupam Kher"
8   ],
9   "trailerUrl": "http://TheKashmirFiles/trailers/1",
10  "posterUrl": "http://TheKashmirFiles/posters/1",
11  "language": "Hindi",
12  "releaseDate": "15-04-2022",
13  "director": "Vivek Agnihotri",
14  "releaseStatus": "RELEASED",
15  "updatedAt": "2022-04-13T15:51:59.907Z",
16  "__v": 0
17 }
```

Requests/Response for various scenarios:

Validation Failures:

1. Validating the movie name

Request:



```
POST localhost:8080/mba/api/v1/movies

Params Authorization Headers (9) Body Pre-request Script Tests Settings
none form-data x-www-form-urlencoded raw binary GraphQL JSON

1  {
2    "name": "",
3    "description": "Intense Movie",
4    "casts": [
5      "Mithun Chakraborty",
6      "Anupam Kher"
7    ],
8    "trailerUrl": "http://TheKashmirFiles/trailers/1",
9    "posterUrl": "http://TheKashmirFiles/posters/1",
10   "language": "Hindi",
11   "releaseDate": "11-03-2022",
12   "director": "Vivek Agnihotri",
13   "releaseStatus": "RELEASED"
14 }
```

Requests/Response for various scenarios:

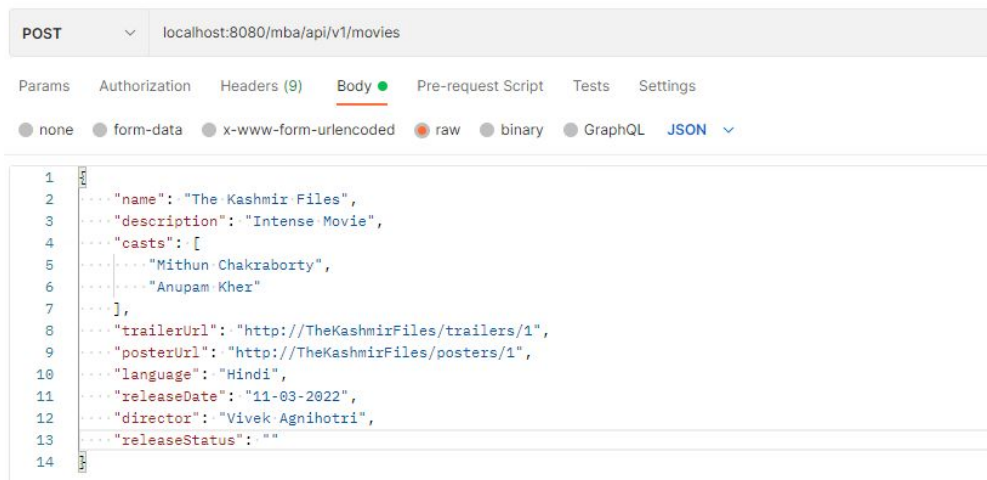
Response:



Requests/Response for various scenarios:

2. Validating the movie releaseStatus

Request:



Requests/Response for various scenarios:

Response:



Requests/Response for various scenarios:

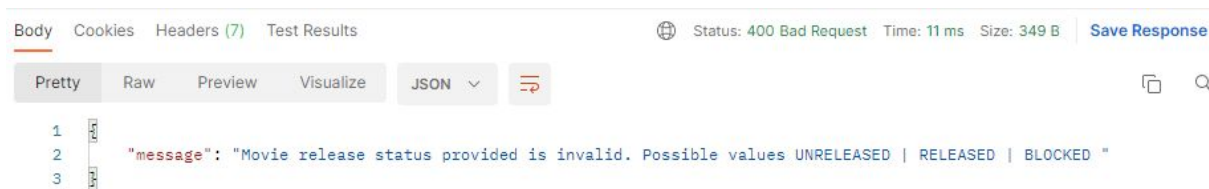
3. Validating if correct value of movie releaseStatus is provided

Request:



Requests/Response for various scenarios:

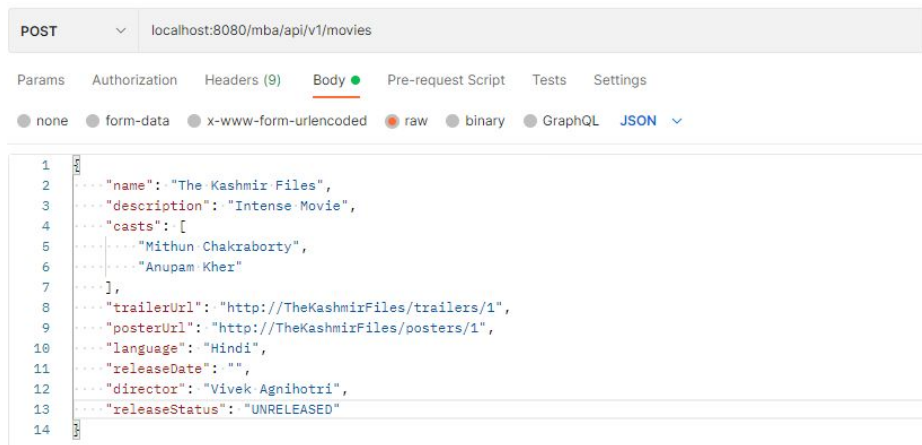
Response:



Requests/Response for various scenarios:

4. Validating the movie releaseDate

Request:



Requests/Response for various scenarios:

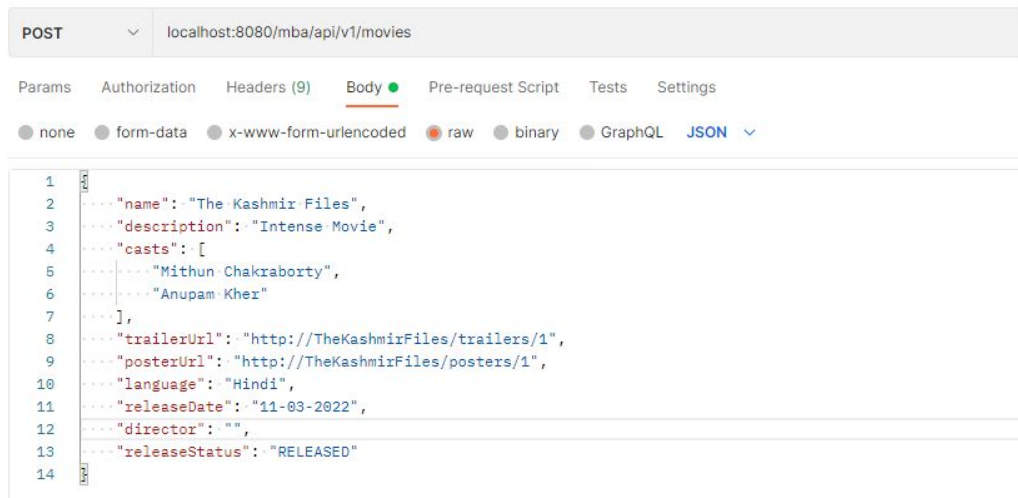
Response:



Requests/Response for various scenarios:

5. Validating the director

Request:



Requests/Response for various scenarios:

Response:



Creation of Custom middleware to validate the request body for theatre resource

We will now create a middleware to validate the request body for theatre resources. Any theatre with valid details should be registered. So we will keep a check on the properties such as theatre name, city, pincode and description.

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

[Code Link] -

https://github.com/Vishwa07dev/mba_backend/blob/session5/middlewares/verifyTheatreReqBody.js

Creation of Custom middleware to validate the request body for theatre resource

Let's name this middleware as verifyTheatreReqBody.js

1. Create validation middleware for theatre request body

- This middleware will have multiple validations for theatre request that is discussed below, we will name it as

```
validateTheatreRequestBody = async (req, res, next) => {}
```

Creation of Custom middleware to validate the request body for theatre resource

2. Validating the theatre name

- Here, we will check if the user has provided the theatre name or not
- If the theatre name is empty or the field is not itself given in the request then we will send a message saying

"Failed! Theatre name is not provided !"

```
//Validate the theatre name
if (!req.body.name) {
    return res.status(400).send({
        message: "Failed! Theatre name is not provided !"
    });
}
```

Creation of Custom middleware to validate the request body for theatre resource

3. Validating the theatre description

- Here, we will check if the user has provided theatre description or not
- If the description is not given in the request then we will send a message saying

"Failed! Theatre description is not provided !"

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

Creation of Custom middleware to validate the request body for theatre resource

4. Validating the theatre city

- Here, we will check if the theatre city is provided or not
- If the city is not given in the request then we will send a message saying

"Failed! Theatre city is not provided !"

```
//Validate the theatre city
if(!req.body.city){
    return res.status(400).send({
        message: "Failed! Theatre city is not provided !"
    });
}
```

Creation of Custom middleware to validate the request body for theatre resource

5. Validating the theatre pincode

- Here, we will check if a user has provided the pincode of the theatre where it is located
- If the pincode is not given in the request then we will send a message saying

"Failed! Theatre location pincode is not provided !"

```
//Validate the theatre pincode
if(!req.body.pinCode){
  return res.status(400).send({
    message: "Failed! Theatre location pincode is not provided !"
  });
}
```

Creation of Custom middleware to validate the request body for theatre resource

6. Validating if there exists same theatre at same location

- Here, we will check if there is a theatre present with the same name at the same location based on pincode
- If the theatre already exists then we will send a message saying

"Failed! Same theatre in same location already exists !"

Creation of Custom middleware to validate the request body for theatre resource

```
/**
 * Validate same theatre at the same location is not created
 */
const theatre = await Theatre.findOne({name: req.body.name , pinCode
: req.body.pinCode});
if(theatre!=null){
    return res.status(400).send({
    message: "Failed! Same theatre in same location already exists !"
    });
}
```

- Once all the validation checks are passed, the theatre will get registered/created.

Updating existing theatre.routes.js for validating theatre registration and updation of theatre details:

Let's make changes to the required routes with the middleware validations -

[Code Link] -

https://github.com/Vishwa07dev/mba_backend/blob/session5/routes/theatre.routes.js

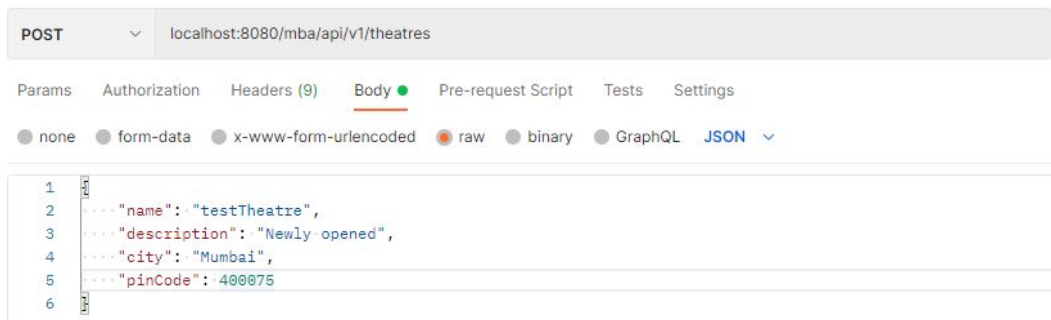
- We will now update the existing theatre.routes.js and add the verifyTheatreReqBody.validateTheatreRequestBody middleware validations as below.

```
app.post("/mba/api/v1/theatres", [verifyTheatreReqBody.validateTheatreRequestBody], theatreController.createTheatre);  
app.put("/mba/api/v1/theatres/:id", [verifyTheatreReqBody.validateTheatreRequestBody], theatreController.updateTheatre);
```

Requests/Response for various scenarios:

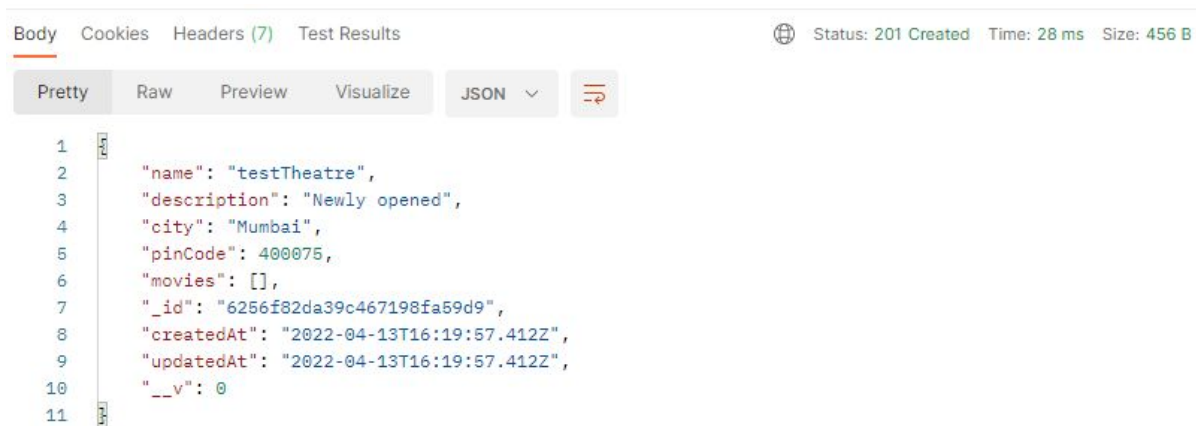
Let's see the valid request and response for theatre registration and updation first and then the different validation failures.

Valid Theatre creation/registration Request:



Requests/Response for various scenarios:

Valid Theatre creation/registration Response:



The screenshot displays a REST client interface with the 'Body' tab selected. The response is a JSON object representing a newly created theatre. The status bar indicates a 201 Created status, a response time of 28 ms, and a size of 456 B. The JSON body contains the following fields:

```
1  {
2    "name": "testTheatre",
3    "description": "Newly opened",
4    "city": "Mumbai",
5    "pinCode": 400075,
6    "movies": [],
7    "_id": "6256f82da39c467198fa59d9",
8    "createdAt": "2022-04-13T16:19:57.412Z",
9    "updatedAt": "2022-04-13T16:19:57.412Z",
10   "__v": 0
11 }
```

Requests/Response for various scenarios:

Valid Theatre updation Request:

PUT localhost:8080/mba/api/v1/theatres/6256f82da39c467198fa59d9

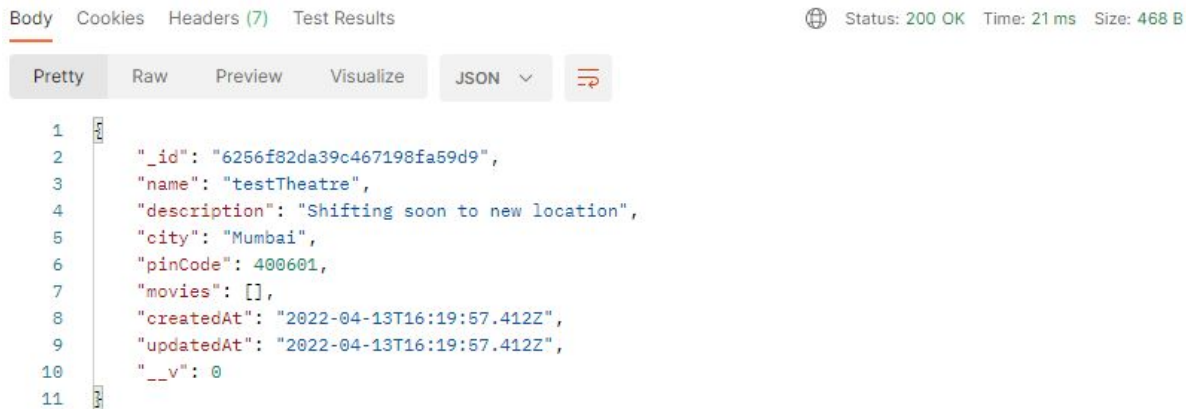
Params Authorization Headers (9) **Body** Pre-request Script Tests Settings

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

```
1 {
2   "name": "testTheatre",
3   "description": "Shifting soon to new location",
4   "city": "Mumbai",
5   "pinCode": 400601
6 }
```

Requests/Response for various scenarios:

Valid Theatre updation Response:



The screenshot displays a REST client interface with the 'Body' tab selected. The response is a JSON object representing a theatre update. The status is 200 OK, the time taken is 21 ms, and the size is 468 B. The JSON is formatted in 'Pretty' mode.

```
1  {
2    "_id": "6256f82da39c467198fa59d9",
3    "name": "testTheatre",
4    "description": "Shifting soon to new location",
5    "city": "Mumbai",
6    "pinCode": 400601,
7    "movies": [],
8    "createdAt": "2022-04-13T16:19:57.412Z",
9    "updatedAt": "2022-04-13T16:19:57.412Z",
10   "__v": 0
11 }
```

Requests/Response for various scenarios:

Validation Failures:

1. Validating the theatre name

Request:

The screenshot shows a REST client interface with the following details:

- Method:** POST
- URL:** localhost:8080/mba/api/v1/theatres
- Body Tab:** Selected, showing a JSON payload:

```
1 {  
2   "name": "",  
3   "description": "Newly opened",  
4   "city": "Mumbai",  
5   "pinCode": 400075  
6 }
```
- Body Type:** JSON (selected from a dropdown menu that also includes none, form-data, x-www-form-urlencoded, raw, binary, and GraphQL).

Requests/Response for various scenarios:

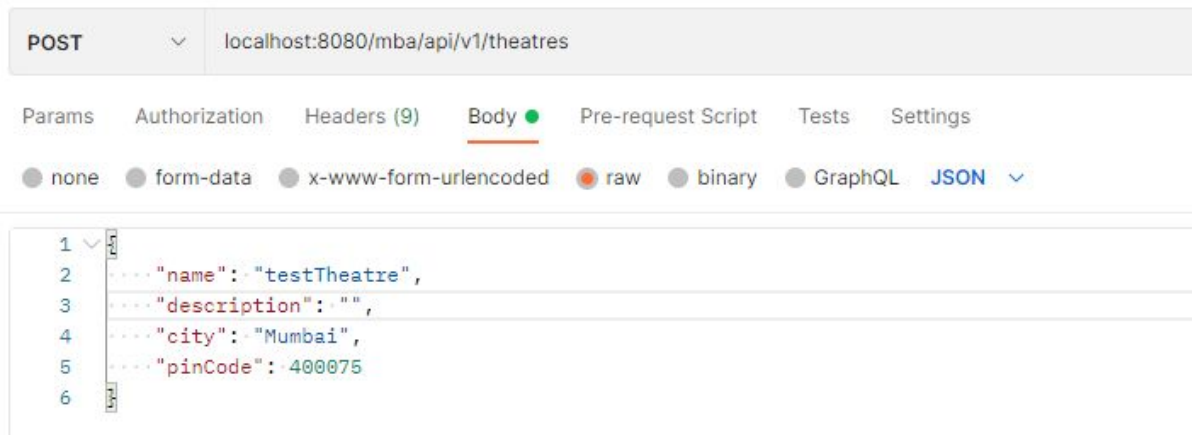
Response:



Requests/Response for various scenarios:

2. Validating the theatre description

Request:



Requests/Response for various scenarios:

Response:



Requests/Response for various scenarios:

3. Validating the theatre city

Request:

The screenshot shows a REST client interface with the following details:

- Method:** POST
- URL:** localhost:8080/mba/api/v1/theatres
- Body:** JSON
- Body Content:**

```
1 {
2   "name": "testTheatre",
3   "description": "Newly opened",
4   "city": "",
5   "pinCode": 400075
6 }
```

Requests/Response for various scenarios:

Response:



Requests/Response for various scenarios:

4. Validating the theatre pincode

Request:

The screenshot shows a REST client interface with the following details:

- Method:** POST
- URL:** localhost:8080/mba/api/v1/theatres
- Body Tab:** Selected, showing a JSON payload:

```
1 {  
2   "name": "testTheatre",  
3   "description": "Newly opened",  
4   "city": "Mumbai"  
5 }
```
- Format:** JSON

Requests/Response for various scenarios:

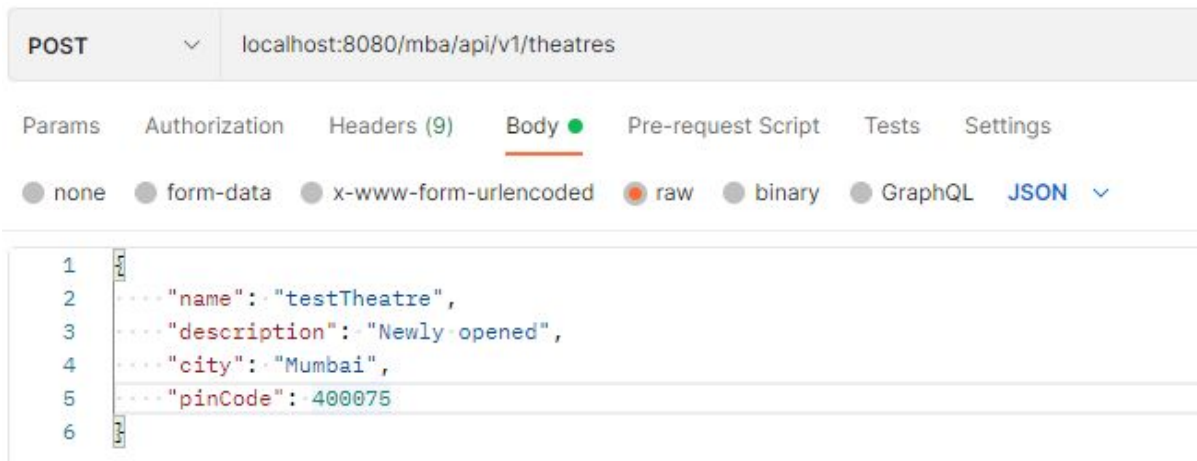
Response:



Requests/Response for various scenarios:

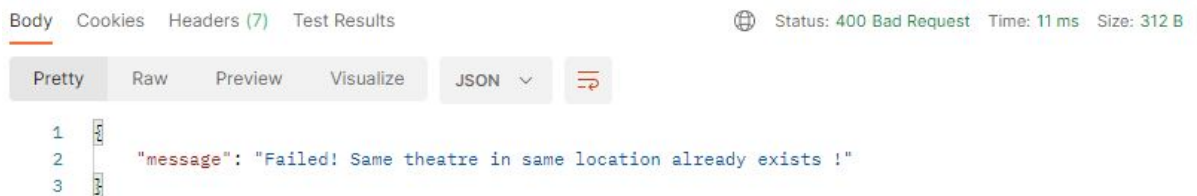
5. Validating if there exists same theatre at same location

Request:



Requests/Response for various scenarios:

Response:



Practice Code

1. Write an API that can delete a theatre based on the theatre name as a query parameter. If the theatre doesn't exist then it will throw an error message saying that "Theatre doesn't exist with the given name". If a theatre exists, then delete the theatre using its name
2. Make this deleteTheatre API secure by allowing only Admin to access it.

MCQs

1. What is the correct representation of a middleware in Express.js?
 - A. `function(req){ }`
 - B. `method(req){ }`
 - C. `function(req,res,next){ }`
 - D. `method(req,res,next){ }`

MCQs

1. What is the correct representation of a middleware in Express.js?

- A. `function(req){ }`
- B. `method(req){ }`
- C. `function(req,res,next){ }`
- D. `method(req,res,next){ }`

Answer: C

MCQs

2. Which of the following middleware parses cookies bound to the client request object?
- A. cookie
 - B. cookie-parser
 - C. cookies
 - D. None of the above

MCQs

2. Which of the following middleware parses cookies bound to the client request object?
- A. cookie
 - B. cookie-parser
 - C. cookies
 - D. None of the above

Answer: B

MCQs

3. Captured values of Route parameters are populated in which of the following?
- A. req.data
 - B. app.locals
 - C. req.params
 - D. All of the above

MCQs

3. Captured values of Route parameters are populated in which of the following?

- A. req.data
- B. app.locals
- C. req.params
- D. All of the above

Answer: C

MCQs

4. Identify the correct API call we make in Postman for fetching all the movies?

- A. GET -> localhost:8080/mba/api/v1/movies/id:
- B. PUT -> localhost:8080/mba/api/v1/movies/id:
- C. POST - > localhost:8080/mba/api/v1/movies
- D. GET - > localhost:8080/mba/api/v1/movies

MCQs

4. Identify the correct API call we make in Postman for fetching all the movies?

- A. GET -> localhost:8080/mba/api/v1/movies/id:
- B. PUT -> localhost:8080/mba/api/v1/movies/id:
- C. POST - > localhost:8080/mba/api/v1/movies
- D. GET - > localhost:8080/mba/api/v1/movies

Answer: D

MCQs

5. What is used in the `isValidEmail` method to check if the email is valid?
- A. String matching
 - B. Character matching
 - C. Regular expression for String matching
 - D. All of the above

MCQs

5. What is used in the `isValidEmail` method to check if the email is valid?
- A. String matching
 - B. Character matching
 - C. Regular expression for String matching
 - D. All of the above

Answer: C

Thank You!