



ITCS212

Web Programming

Project: Web Application

Movie: NungNaDoo

Chatkawin	Phongpawarit	6388041
Chanisara	Kotrachai	6388087
Jittiwat	Sanit	6388101
Prapakorn	Sealim	6388145

Section 2 Group 13

Dr. Jidapa Kraisangka

Dr. Wudhichart Sawangphol

Table of content:

Overview	1
Na Diagram	2
User View	2
Admin View	2
Normal User Pages	3
Administrators Pages	5
Web Services	9
Search Function	9
For Admin	9
For Content	11
Web Services Interaction	14
Postman Testing	15

Overview:

Our website “NUNGNADOO” is a website about movies.

Phase 1: In phase 1, we create the navigation diagram to show the flow or transition between each page. Our website “NUNGNADOO” has a homepage that has a navigation bar that can switch to a member page, login page, and search page for the normal user.

Phase 2: In phase 2, the database needs to be implemented in DBMS, preferably MySQL. Therefore, we created the database to store the user information, which includes username, user ID, and password, as well as the content information, which includes content ID, content title, content pic, content description, and movie information, which includes movie ID, movie title, and movie picture. We have functions for normal users, the users can search and view products or services in the system, our search function supports the no criteria search and criteria search. We also use the RAPIDAPI API to achieve phase 2.

Phase 3: In phase 3, for the web application for administrators we created the admin dashboard that contains the navigation bar that can switch into the user management page and movie management page. On the user management page, the admin can see the list of users, and the admin can update, insert users into the list, and delete the users from the list. For the movie management page, the admin can see the list of movies, and the admin can update and insert movies into the list and delete the movie from the list.

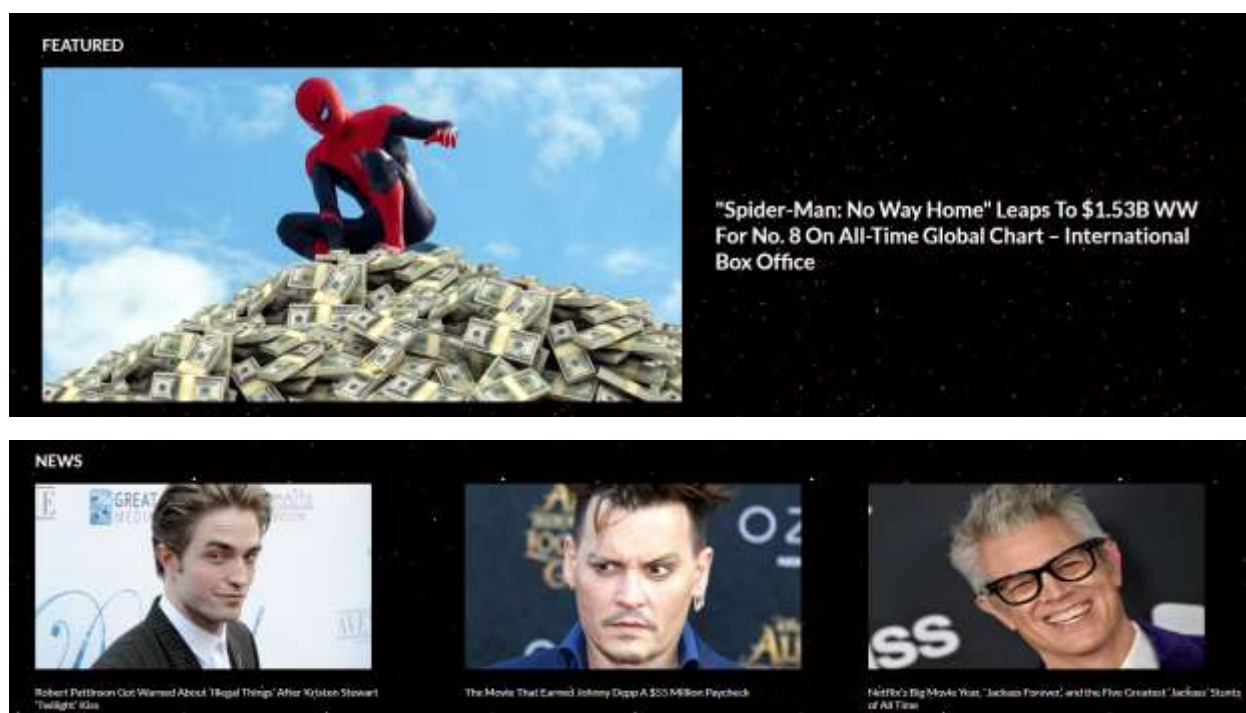


Normal User Pages:

Login Page: Users can login into the NUNGNADOO website from the login page.

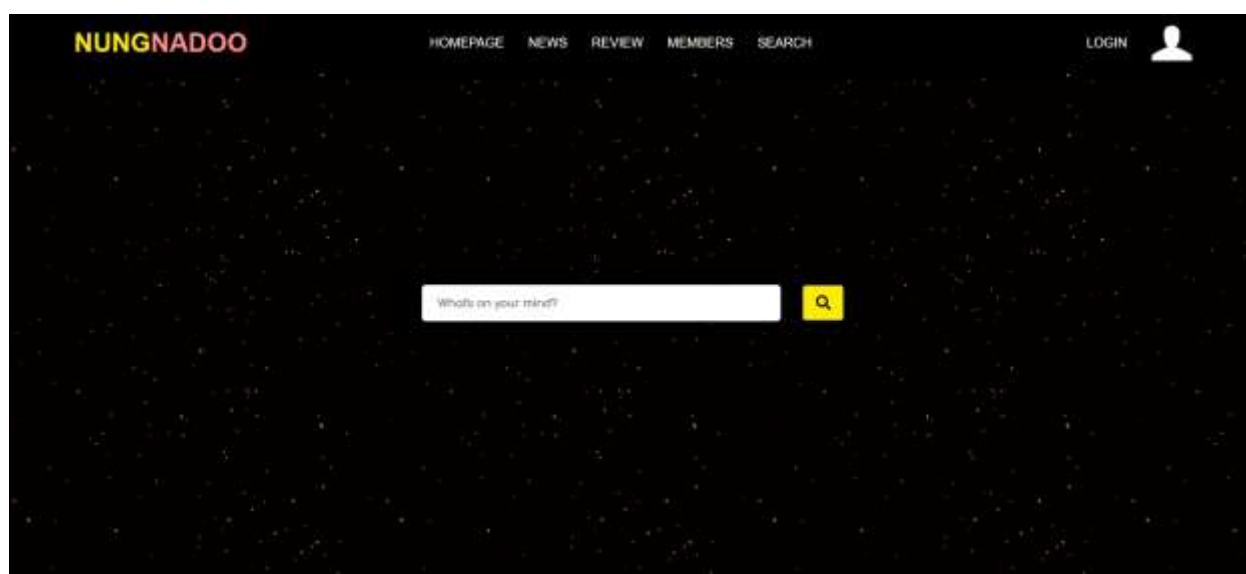


Homepage: Users can see the news, features, and reviews from this page.





Search: Users can search on the search page.

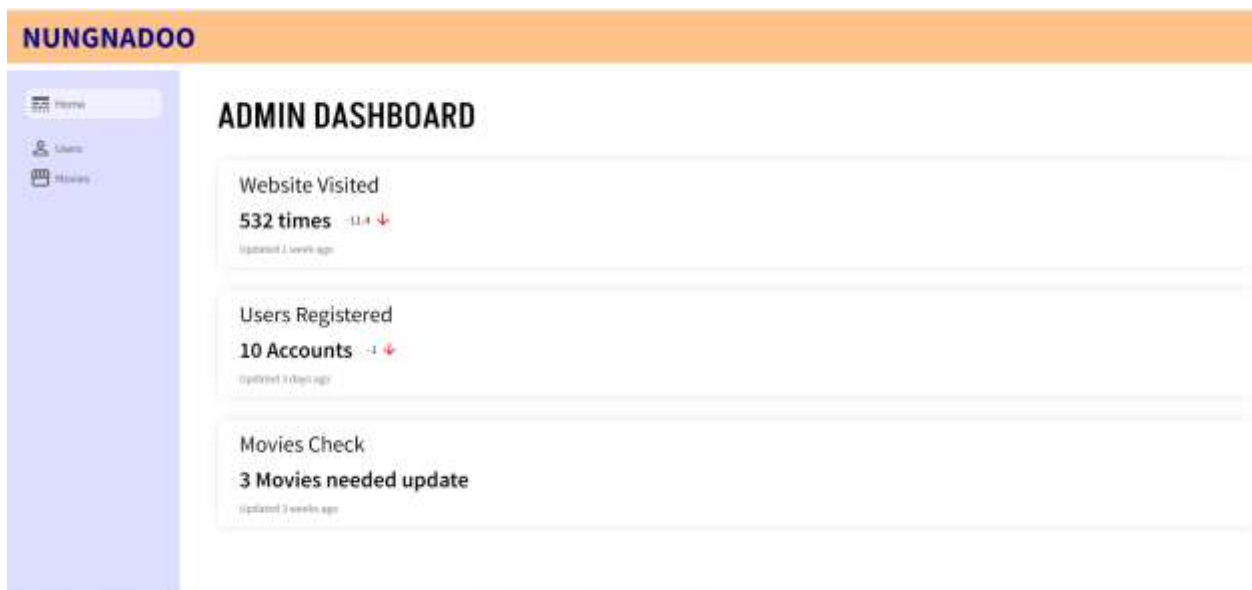


Search Result:



Administrators Pages:

Homepage of the admin dashboard.



User page for admin: Admin can edit the name or delete the user from this page. If the admin wants to delete the user, click the bin button,

ID	Username	Password	Action
1	Tiger	080601117	Edit Delete
2	Shivansh	yvy2744	Edit Delete
3	Bubbl	60112233	Edit Delete
4	Aswiny	09090123	Edit Delete
5	Ramkumar	boom087345678	Edit Delete
6	Adar	2545454	Edit Delete
7	Rudraj	09112376	Edit Delete
8	Shivansh	600000	Edit Delete

If the admin wants to edit and update the Id, name, and password of the user, click the Edit button. Edit user page will show.

Edit User

Edit

ID

Username

Password

Update

Create

If the admin wants to create a new user, click the create button. A new user page will show.

New User

ID

Username

Password

Create

If the admin wants to create a new movie, click the create button. This page will show.



The screenshot shows a web application interface for NUNGNADOO. At the top is an orange header bar with the text "NUNGNADOO" in white. Below the header is a light blue sidebar containing three menu items: "Home" (with a house icon), "Users" (with a person icon), and "Movies" (with a film strip icon). The "Movies" item is highlighted. The main content area is white and titled "New Movie" in bold. Below the title are three input fields: "ID", "Movie Title", and "Movie Picture". Each field has a small red error message below it: "Required" for ID, "Required" for Movie Title, and "Required" for Movie Picture. At the bottom of the form is a blue "Create" button.

NUNGNADOO

New Movie

ID

Required

Movie Title

Required

Movie Picture

Required

Create

Web Services:

Search Function

```
app.post('/result', function(request, response) {
  var inp = request.body.searchinput;
  connection.query('SELECT * FROM Movie WHERE movie_title LIKE "%${inp}%"', function(error, res, field) {
    if (error) throw error;
    console.log(res);
    response.render('result', {title: 'Result', result: res});
  });
});
```

When we searched with the input, it redirects the page to be resulted by selecting the data which includes the Movie title from our database and rendering as the result, {title: (movie name)}

For Admin:

Get User

```
////////////////////////////////////////ADMIN////////////////////////////////////////
// Get all user list
//PASS
router.get('/user', function (req, res){
  connection.query('SELECT * FROM user', function (error, results){
    if(error) throw error;
    return res.send({ error: false, data: results, message: 'User Lists' });
  });
});
```

When an admin wants to see all the users in the database, the admin can use the Postman application to get all the user lists in the database by using a GET request in the Postman application at localhost:3000/user. If there is an error then return the message “error: false”, If nothing is an error then display all the users in the database.

Insert User

```
//Insert user
//PASS
router.post('/user', function (req, res){
  console.log(req.body);
  let user = req.body.user;
  console.log(user);

  if (!user) {
    return res.status(400).send({error: true, message: 'Please provide User Information'});
  }

  connection.query("INSERT INTO user SET ? ", user, function (error, results){
    if(error) throw error;
    return res.send({ error: false, data: results.affectedRows, message: 'Inserted User' });
  });
});
```

When the admin wants to insert a new user into the database, the admin can use the Postman application to do it by using the POST request with the URL localhost:3000/user. If there is an error, return the message "error: false." If there is no error, display all of the users in the database.

Update User

```
//Update User info
//PASS
router.put('/user', function (req, res){
  console.log(req.body);
  let user = req.body.user;
  console.log(user);

  if (!user) {
    return res.status(400).send({error: true, message: 'Please provide User Information'});
  }

  connection.query("UPDATE user SET ? WHERE user_ID = ?", [user, user.user_ID], function (error, results){
    if(error) throw error;
    return res.send({ error: false, data: results.affectedRows, message: 'Updated User Information' });
  });
});
```

When the admin wants to update users on the database, the admin can use the Postman application to do it by using the PUT request with the URL localhost:3000/user. If there is an error, return the message "error: false." If there is no error, display the message that says, "Updated User Information".

Delete User

```
//Delete user from db
//PASS
router.delete('/user', function (req, res){
  console.log(req.body);
  let user = req.body.user.user_ID;
  console.log(user);

  if (!user) {
    return res.status(400).send({error: true, message: 'Please provide User ID'});
  }

  connection.query("DELETE FROM user WHERE id = ?", user, function (error, results){
    if(error) throw error;
    return res.send({ error: false, data: results.affectedRows, message: 'Deleted User' });
  });
});
```

When the admin wants to delete users on the database, the admin can use the Postman application to do it by using the DELETE request with the URL localhost:3000/user. If there is an error, return the message "error: false." If there is no error, display the message that says, "Delete User".

For Content:

Get Content

```
// Get all Content list
//PASS
router.get('/content', function (req, res){
  connection.query('SELECT * FROM Content', function (error, results){
    if(error) throw error;
    return res.send({ error: false, data: results, message: 'Content Lists' });
  });
});
```

When an admin wants to see all the content lists in the database, the admin can use the Postman application to get all the movie lists in the database by using a GET request in the Postman application at localhost:3000/content. If there is an error then return the message "error: false", If nothing is an error then display all the content list in the database.

Insert Content

```

//Insert content
//PASS
router.post('/content', function (req, res){
  console.log(req.body);
  let content = req.body.Content;
  console.log(content);

  if (!content) {
    return res.status(400).send({error: true, message: 'Please provide Content Information'});
  }

  connection.query("INSERT INTO Content SET ? ", content, function (error, results){
    if(error) throw error;
    return res.send({ error: false, data: results.affectedRows, message: 'Inserted Content' });
  });
});

```

When the admin wants to insert new content into the database, the admin can use the Postman application to do it by using the POST request with the URL localhost:3000/content. If there is an error, return the message "error: false." If there is no error, show the message "Inserted Content".

Update Content

```

//Update Content info
//PASS
router.put('/content', function (req, res){
  console.log(req.body);
  let content = req.body.Content;
  console.log(content);

  if (!content) {
    return res.status(400).send({error: true, message: 'Please provide Content Information'});
  }

  connection.query("UPDATE Content SET ? WHERE content_ID = ?", [content, content.content_ID], function (error, results){
    if(error) throw error;
    return res.send({ error: false, data: results.affectedRows, message: 'Updated Content Information' });
  });
});

```

When the admin wants to update content on the database, the admin can use the Postman application to do it by using the PUT request with the URL localhost:3000/content. If there is an error, return the message "error: false." If there is no error, display the message that says, "Updated content Information".

Delete Content

```
//Delete Content from db
//PASS
router.delete('/content', function (req, res){
  console.log(req.body);
  let content = req.body.Content.content_ID;
  console.log(content);

  if (!content) {
    return res.status(400).send({error: true, message: 'Please provide Content ID'});
  }

  connection.query("DELETE FROM Content WHERE content_ID = ?", content, function (error, results){
    if(error) throw error;
    return res.send({ error: false, data: results.affectedRows, message: 'Deleted Content' });
  });
});

router.delete('/movie', function (req, res){
  console.log(req.body);
  let movie = req.body.Movieid;
  console.log(movie);

  if (!movie) {
    return res.status(400).send({error: true, message: 'Please provide Movie ID'});
  }

  connection.query("DELETE FROM Movie WHERE id = ?", movie, function (error, results){
    if(error) throw error;
    return res.send({ error: false, data: results.affectedRows, message: 'Deleted Movie' });
  });
});
```

When the admin wants to delete content on the database, the admin can use the Postman application to do it by using the DELETE request with the URL localhost:3000/content. If there is an error, return the message "error: false." If there is no error, display the message that says "Deleted content".

Web Services Interaction

```

<script>
  Title = document.querySelector("#title");
  Poster = document.querySelector("#poster");
  Desc = document.querySelector("#desc");
  Year = document.querySelector("#year");
  const options = {
    method: 'GET',
    headers: {
      'X-RapidAPI-Host': 'movie-database-alternative.p.rapidapi.com',
      'X-RapidAPI-Key': 'ace4465481msh829161a72e401eap1d97d0jsn7882921b7562'
    }
  };

  fetch('https://movie-database-alternative.p.rapidapi.com/?r=json&i=tt0865554', options)
    .then(response => response.json())
    .then(response => {

      console.log(response)
      Poster.innerHTML = ``

      Title.innerHTML = `${response.Title}`;

      Desc.innerHTML = `${response.Plot}`;

      Year.innerHTML = `Release Date: ${response.Released}`;
    })
    .catch(err => {
      console.error(err);
    })
  );
</script>

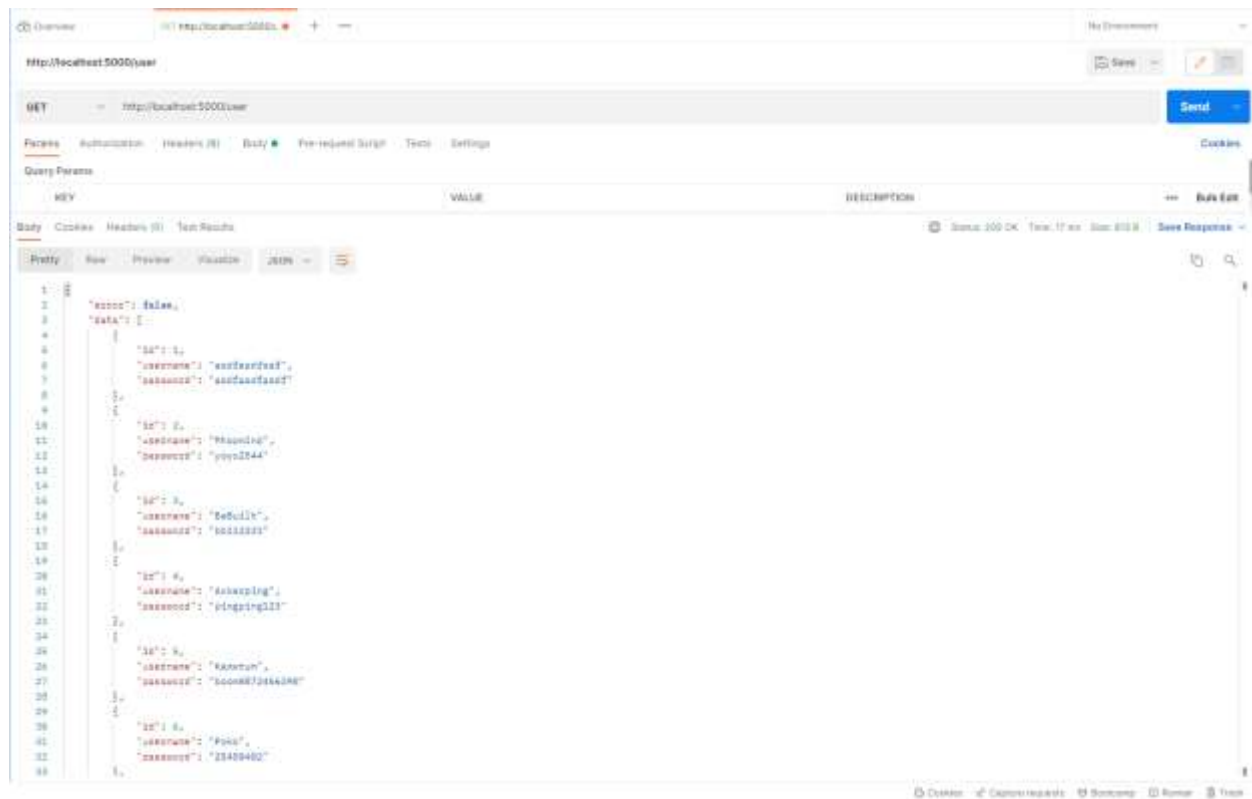
```

At first, we get the API key from rapidAPI.com then we use the provided script on the website to get movie information such as movie poster, title, movie description, release date, after we get all the information, we show all the provided information, add CSS then show the information on the result page after we searched by the keyword that user input.

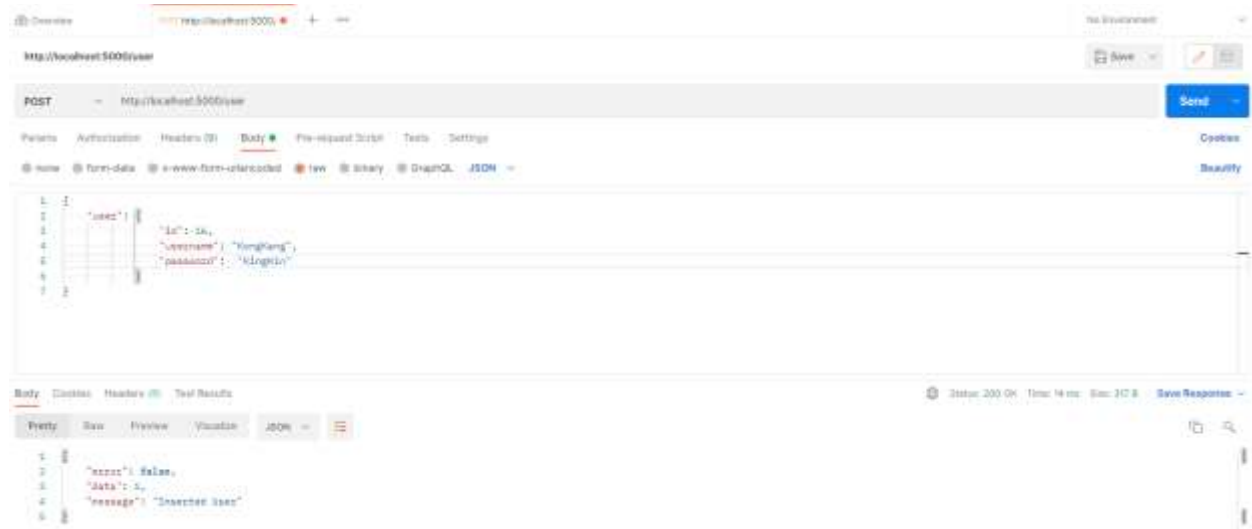
Postman Testing:

For user

Get user:



Insert User:



Update User:

The screenshot shows the Postman interface for a PUT request to `http://localhost:5000/user`. The request body is a JSON object:

```
{  "id": 2,  "username": "test",  "password": "testtest"}
```

The response status is 200 OK, and the response body is:

```
{  "access": false,  "data": 1,  "message": "updated user information"}
```

Delete User:

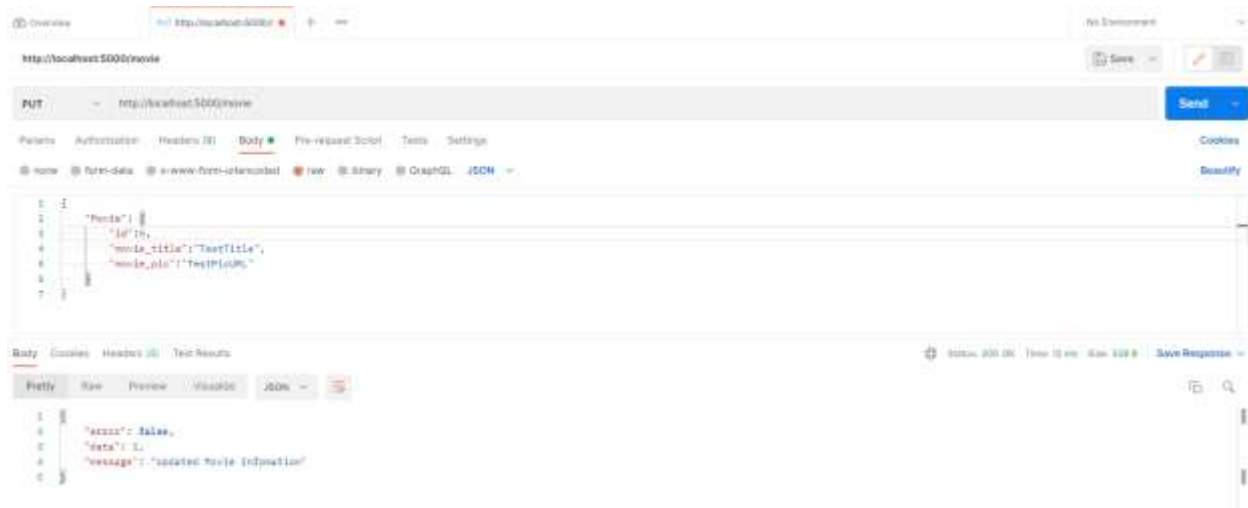
The screenshot shows the Postman interface for a DELETE request to `http://localhost:5000/user`. The request body is a JSON object:

```
{  "user_id": 2}
```

The response status is 200 OK, and the response body is:

```
{  "access": false,  "data": 1,  "message": "deleted user"}
```


Update Movie:



Overview http://localhost:5000/ Save Send

PUT http://localhost:5000/movie Send

Params Authorization Headers (3) **Body** Pre-request Script Tests Settings Cookies Beautify

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

```

1 {
2   "movie": {
3     "id": 1,
4     "movie_title": "TestTitle",
5     "movie_plot": "TestPlot"
6   }
7 }

```

Body Cookies Headers (3) Test Results Status: 200 OK Time: 10 ms Size: 128 B Save Response

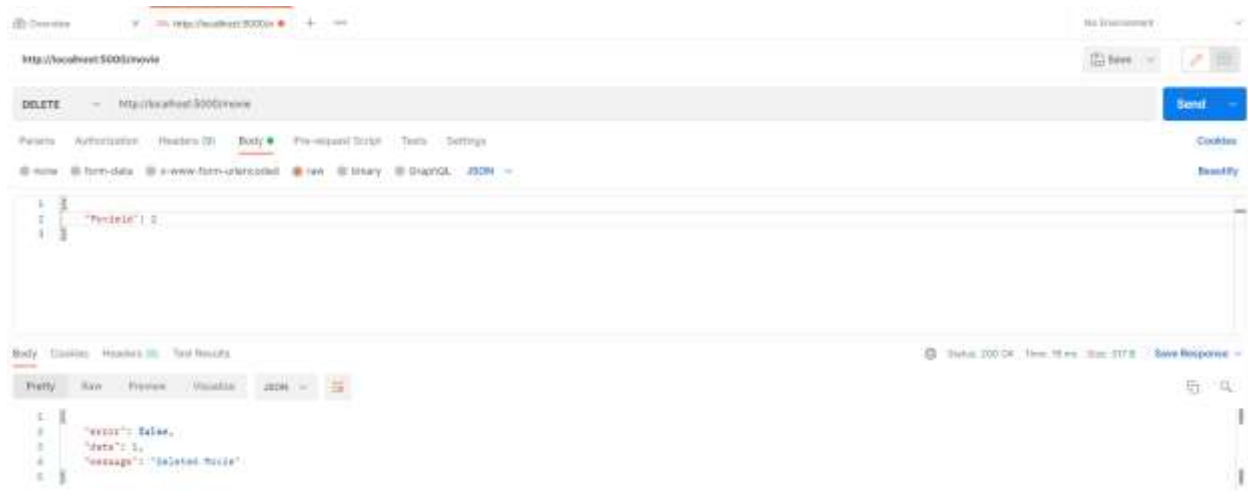
Pretty Raw Preview Variables JSON Save

```

1 {
2   "success": false,
3   "data": 1,
4   "message": "Updated Movie Information"
5 }

```

Delete Movie:



Overview http://localhost:5000/ Save Send

DELETE http://localhost:5000/movie Send

Params Authorization Headers (3) **Body** Pre-request Script Tests Settings Cookies Beautify

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

```

1 {
2   "id": 1
3 }

```

Body Cookies Headers (3) Test Results Status: 200 OK Time: 10 ms Size: 117 B Save Response

Pretty Raw Preview Variables JSON Save

```

1 {
2   "success": false,
3   "data": 1,
4   "message": "Deleted Movie"
5 }

```