

Adding the movie into the table webapi

The screenshot shows two screenshots of the Swagger UI interface for a 'CreateWebapi' API.

Top Screenshot (Definition View):

- Header:** Select a definition **CreateWebapi v1**
- Title:** CreateWebapi 1.0 OAS3
- Endpoint:** POST /api/AddMovie
- Parameters:** No parameters
- Request body:** application/json
- Body Content:**

```
{
  "movieId": 0,
  "movieName": "Leo",
  "movieType": "Commercial/Action",
  "movieLanguage": "Tamil/Telugu",
  "movieDuration": "2.58 Hour"
}
```

Bottom Screenshot (Execution View):

- Header:** localhost:5224/swagger/index.html
- Buttons:** Execute, Clear
- Responses:**
 - Curl:**

```
curl -X 'POST' \
  'http://localhost:5224/api/AddMovie' \
  -H 'accept: */*' \
  -H 'Content-Type: application/json' \
  -d '{
    "movieId": 0,
    "movieName": "Leo",
    "movieType": "Commercial/Action",
    "movieLanguage": "Tamil/Telugu",
    "movieDuration": "2.58 Hour"
}'
```
 - Request URL:** <http://localhost:5224/api/AddMovie>
 - Server response:**
 - Code:** 200
Details: Response headers:

```
access-control-allow-origin: http://localhost:5224
content-type: application/json
date: Sun, 26 Dec 2023 03:55:03 GMT
server: Kestrel
vary: Origin
```
 - Responses:**
 - Code:** 200
Description: Success

Read the movie details from table web api

localhost:5059/swagger/index.html

Select a definition FetchWebapi v1

FetchWebapi 1.0 OAS3

FetchMovie

GET /api/FetchMovie

Parameters

No parameters

Responses

Curl

```
curl -X 'GET' \
'http://localhost:5059/api/FetchMovie' \
-H 'accept: text/plain'
```

Request URL

```
http://localhost:5059/api/FetchMovie
```

Code Details

200 Response body

```
[ { "movieId": 4, "movieName": "Hello", "movieType": "Drama/Hriller", "movieLanguage": "Hindi/Tamil", "movieDuration": "1.50 Hour" } ]
```

Download

Response headers

```
Content-Type: application/json; charset=utf-8
Date: Tue, 26 Dec 2023 05:52:21 GMT
Server: Kestrel
Transfer-Encoding: chunked
```

Responses

Code	Description	Links
200	Success	No links

Media type

text/plain

Controls Accept header.

Example Value | Schema

```
[ { "movieId": 4, "movieName": "string", "movieType": "string" } ]
```

Fetch individual data by id

The screenshot shows a browser window with multiple tabs open, all titled "Swagger". The active tab is "localhost:5059/swagger/index.html". The main content area displays a JSON schema for a response array:

```
[  
  {  
    "movieId": 0,  
    "movieName": "string",  
    "movieType": "string",  
    "movieLanguage": "string",  
    "movieDurations": "string"  
  }  
]
```

Below this, a "GET /api/FetchMovie/{id}" section is shown. It includes a "Parameters" table with one entry:

Name	Description
id <small>required</small>	integer(\$int32) (path) 4

Buttons for "Execute" and "Clear" are present. A "Responses" section follows, containing a "Curl" command and a "Request URL" field.

The screenshot shows a browser window with multiple tabs open, all titled "Swagger". The active tab is "localhost:5059/swagger/index.html". The main content area displays a "Curl" command and a "Request URL" field.

```
curl -X 'GET' \  
  'http://localhost:5059/api/FetchMovie/4' \  
  -H 'accept: text/plain'
```

Below this, a "Server response" section is shown. It includes a "Code" table with one entry (200) and a "Details" table with one entry (Response body). The Response body contains the following JSON:

```
{  
  "movieId": 4,  
  "movieName": "Hello 2",  
  "movieType": "Drama/Thriller",  
  "movieLanguage": "Hindi/English",  
  "movieDurations": "1.50 Hour"  
}
```

Buttons for "Download" and "Copy" are present. A "Responses" section follows, containing a "Code" table with one entry (200) and a "Description" table with one entry (SUCCESS). The "Media type" dropdown is set to "text/plain".

Fetch total no of movie in table

text/plain

Controls Accept header.

Example Value | Schema

```
{ "movieId": 6, "movieName": "string", "movieType": "string", "movieLanguage": "string", "movieDurations": "string" }
```

GET /api/TotalMovie

Parameters

No parameters

Execute Clear

Responses

Curl

```
curl -X 'GET' \
'http://localhost:5059/api/TotalMovie' \
-H 'accept: text/plain'
```

Request URL

```
http://localhost:5059/api/TotalMovie
```

Server response

Code Details

6

200 Response body

6

Response headers

```
content-type: application/json; charset=utf-8
date: Tue, 26 Dec 2023 06:45:19 GMT
server: Kestrel
transfer-encoding: chunked
```

Responses

Code Description Links

200 Success No links

Media type

text/plain

Controls Accept header.

Example Value | Schema

Update the data in webapi

The screenshot shows the Swagger UI interface for the `UpdateWebapi` API. The top navigation bar includes tabs for `Batch`, `N`, `Swagger`, `Swagger`, `Swagger`, `Swagger`, `Add`, `Swagger`, `Mate`, `react`, `Until`, `React`, and a plus sign. The main title is `UpdateWebapi 1.0 OAS3`. The URL is `http://localhost:5273/swagger/v1/swagger.json`. A dropdown menu titled `Select a definition` shows `UpdateWebapi v1`. Below the title, it says `UpdateMovie`. The method is `PUT` and the endpoint is `/api/UpdateMovie`. The parameters section indicates "No parameters". The request body is set to `application/json`. The JSON payload shown is:

```
{  
    "movieId": 4,  
    "movieName": "Kiren",  
    "movieType": "Action",  
    "movieLanguage": "Tamil",  
    "movieDuration": "2.00 Hour"  
}
```

The screenshot shows the execution results for the `UpdateMovie` endpoint. At the top, there is a blue button labeled `Execute` and a `Clear` button. The `Responses` section contains a `Curl` command and a `Request URL`. The `Request URL` is `http://localhost:5273/api/UpdateMovie`. The `Server response` section shows a `Code` of `200` and a `Description` of `Success`. The `Response headers` are listed as follows:

```
access-control-allow-origin: http://localhost:5273  
content-length: 0  
date: Tue, 26 Dec 2023 06:48:09 GMT  
server: Kestrel  
vary: Origin
```

Delete the particular data by id

DeleteWebapi 1.0 OAS3
http://localhost:5225/swagger/v1/swagger.json

DeleteMovie

DELETE /api/DeleteMovie/{id}

Parameters

Name	Description
id * required	integer(\$int32) 2 (path)

Responses

Curl

```
curl -X 'DELETE' \
'http://localhost:5225/api/DeleteMovie/2' \
-H "Accept: */*
```

Request URL

```
http://localhost:5225/api/DeleteMovie/2
```

Server response

Code **Details**

200

Response headers

```
access-control-allow-origin: http://localhost:5225
content-length: 0
date: Tue, 26 Dec 2023 04:12:18 GMT
server: Kestrel
vary: Origin
```

Responses

Code **Description**

200 Success

Links

No links

Login And Register webapi in separate database

Register also check whether the email is already exists or not

The screenshot shows the Swagger UI interface for a 'User' endpoint. The top navigation bar has tabs for 'Select a definition' and 'LoginWebapi v1'. Below the navigation, the title 'LoginWebapi 1.0 OAS3' is displayed along with the URL 'http://localhost:5166/swagger/v1/swagger.json'. The main content area shows a 'POST /api/Register' operation. It includes sections for 'Parameters' (which is empty) and 'Request body'. The request body example is a JSON object:

```
{
  "userEmail": "Kiren@gmail.com",
  "userPassword": "kiren123",
  "userName": "Kiren Pal"
}
```

The screenshot shows the results of executing the 'User' registration endpoint. The top bar shows the URL 'localhost:5166/swagger/index.html'. The main content area has a 'Responses' section. Under the 'Curl' tab, there is a command-line example:

```
curl -X 'POST' \
  'http://localhost:5166/api/Register' \
  -H 'Accept: */*' \
  -H 'Content-Type: application/json' \
  -d '{
    "userEmail": "Kiren@gmail.com",
    "userPassword": "kiren123",
    "userName": "Kiren Pal"
}'
```

Under the 'Request URL' tab, the URL 'http://localhost:5166/api/Register' is shown. The 'Server response' section displays the status code 200 and the response body:

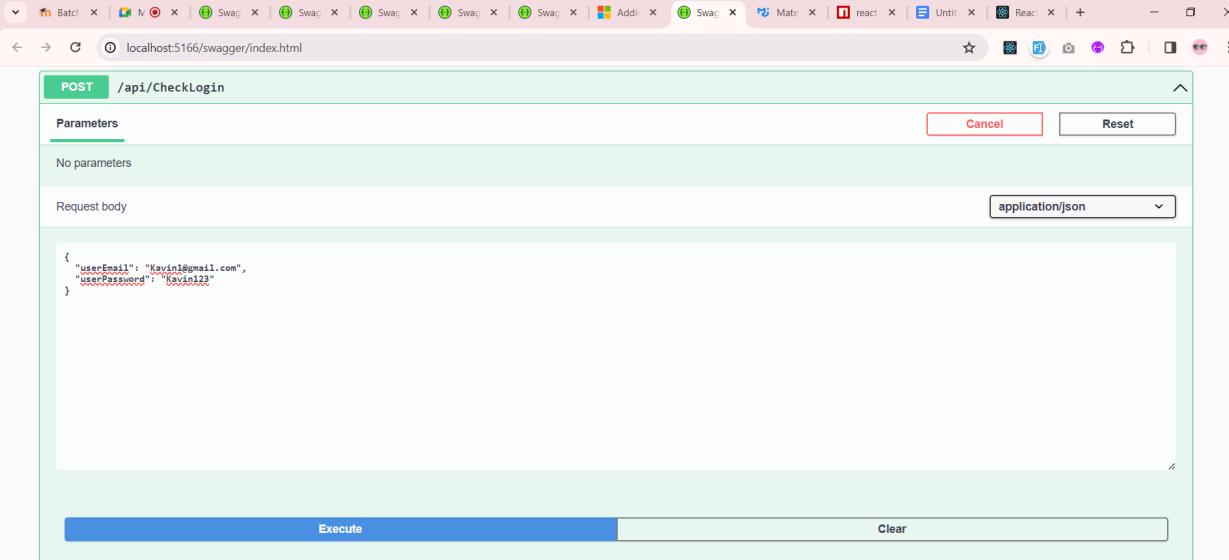
```
{"Exists":false}
```

It also shows the response headers:

```
content-type: text/plain; charset=utf-8
date: Tue, 26 Dec 2023 06:58:15 GMT
server: Kestrel
transfer-encoding: chunked
```

Login

In this check whether the account exists or not and email and password right or wrong
If all correct then it return username



The screenshot shows the Postman interface with the following details:

- Method:** POST
- URL:** /api/CheckLogin
- Parameters:** No parameters
- Request body:** application/json
- Body Content:**

```
{
  "userEmail": "Kavini@gmail.com",
  "userPassword": "Kavini123"
}
```

- Buttons:** Execute, Clear
- Responses:**
- Curl:**

```
curl -X 'POST' \
  'http://localhost:5166/api/CheckLogin' \
  -H 'accept: */*' \
  -H 'Content-Type: application/json' \
  -d {
    "userEmail": "Kavini@gmail.com",
    "userPassword": "Kavini123"
  }'
```

- Request URL:** http://localhost:5166/api/CheckLogin
- Server response:**

Code	Details
200	Response body: <pre>{"account":true,"emailstatus":true,"passwordstatus":true,"username":"Kavin Subburaj"}</pre> Download
	Response headers: <pre>content-type: text/plain; charset=utf-8 date: Tue, 26 Dec 2023 06:52:28 GMT server: Kestrel transfer-encoding: chunked</pre>

- Responses:**

Code	Description	Links
200	Success	No links

Ocelot Webapi Gateway checked in postman
 Checking the fetchwebapi in ocelot using postman

The screenshot shows the Postman interface with a successful API call. The URL is `http://localhost:5245/apiGateway/FetchMovie`. The response body is:

```
1 [  
2 {  
3   "movieId": 1,  
4   "movieName": "Manik Basha",  
5   "movieType": "Commercial/Drama/Action",  
6   "movieLanguage": "Tamil/Telugu/Malayalam/Hindi",  
7   "movieDurations": "2.54 Hour"  
8 }  
9 ]
```

Checking the fetchwebapi for particular data by id in ocelot using postman

The screenshot shows the Postman interface with a successful API call. The URL is `http://localhost:5245/apiGateway/FetchMovie/1`. The response body is:

```
1 {  
2   "movieId": 1,  
3   "movieName": "Manik Basha",  
4   "movieType": "Commercial/Drama/Action",  
5   "movieLanguage": "Tamil/Telugu/Malayalam/Hindi",  
6   "movieDurations": "2.54 Hour"  
7 }
```

Checking thecreatewebsapi in ocelot using postman

My Workspace

POST http://localhost:5245/apiGateway/AddMovie

Body (JSON)

```
1 {  
2   "movieId": 0,  
3   "movieName": "Leo",  
4   "movieType": "Commercial/Drama/Action",  
5   "movieLanguage": "Tamil/Telugu/Malayalam/Hindi",  
6   "movieDurations": "3.04 Hour"  
7 }
```

Headers (3)

Key	Value
Content-Length	0
Date	Tue, 26 Dec 2023 04:36:46 GMT
Server	Kestrel

Body Cookies Headers (3) Test Results

Status: 200 OK Time: 1352 ms Size: 92 B Save as example

Checking the updatewebapi in ocelot using postman

My Workspace

PUT http://localhost:5245/apiGateway/UpdateMovie

Body (JSON)

```
1 {  
2   "movieId": 3,  
3   "movieName": "Leo 2",  
4   "movieType": "Commercial/Drama/Action",  
5   "movieLanguage": "Tamil/Telugu/Malayalam/Hindi",  
6   "movieDurations": "3.04 Hour"  
7 }
```

Headers (3)

Key	Value
Content-Length	0
Date	Tue, 26 Dec 2023 04:37:36 GMT
Server	Kestrel

Body Cookies Headers (3) Test Results

Status: 200 OK Time: 927 ms Size: 92 B Save as example

Checking the deletewebapi in ocelot using postman

My Workspace

Overview

DELETE http://localhost:5245/apiGateway/DeleteMovie/3

Params Authorization Headers (7) Body Pre-request Script Tests Settings

Query Params

Key	Value	Description
Key	Value	Description

Body Cookies Headers (3) Test Results

Key	Value
Content-Length	0
Date	Tue, 26 Dec 2023 04:38:05 GMT
Server	Kestrel

Postbot Runner Start Proxy Cookies Trash

Checking the loginapi for checklogin in ocelot using postman

My Workspace

Overview

POST http://localhost:5245/apiGateway/CheckLogin

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

Body

```
1 {  
2   "userEmail": "Kavin1@gmail.com",  
3   "userPassword": "Kavin123"  
4 }
```

Body Cookies Headers (4) Test Results

Pretty	Raw	Preview	Visualize
1 {"account":true,"emailstatus":true,"passwordstatus":true,"usernamer":"Kavin Subburaj"}			

Postbot Runner Start Proxy Cookies Trash

Checking the loginapi for register in ocelot using postman

The screenshot shows the Postman application interface. On the left, there's a sidebar with 'My Workspace' containing 'Collections' (API testing basics), 'Environments', and 'History'. The main area displays a POST request to `http://localhost:5245/apiGateway/Register`. The 'Body' tab is selected, showing the following JSON payload:

```
1 {
2   "userEmail": "Kavin12@gmail.com",
3   "userPassword": "Kavin123",
4   "userName": "Kavin Subburaj"
5 }
```

Below the request, the response status is shown as `Status: 200 OK Time: 32 ms Size: 150 B`. The response body is displayed as:

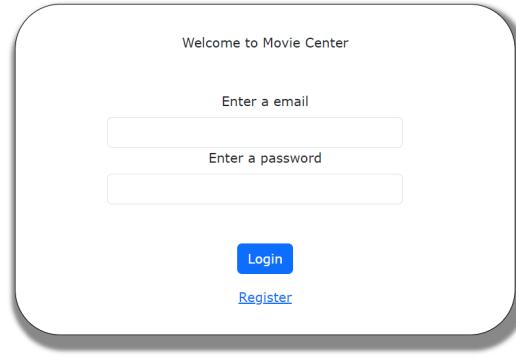
```
1 {"Exists":false}
```

At the bottom, the Windows taskbar is visible with various icons and the date/time: 12:26 26-12-2023.

In React

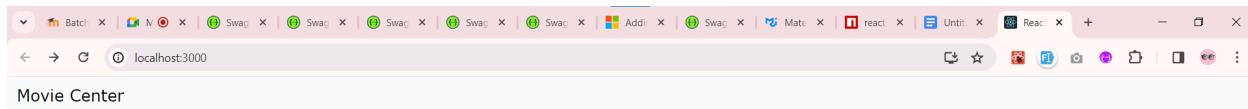
Login page with client side and server side validations

The screenshot shows a browser window with multiple tabs open, all titled 'Swag'. The active tab has the URL `localhost:3000` and displays a login form for 'Movie Center'. The form consists of two input fields: 'Enter a email' and 'Enter a password', followed by a 'Login' button and a 'Register' link.

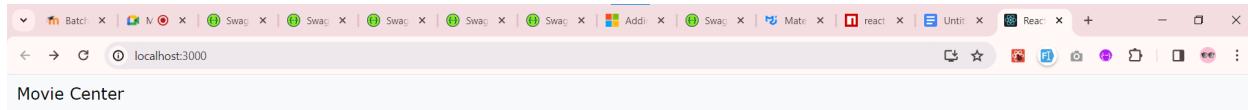


This screenshot shows the 'Movie Center' login page again, but now with successful validation. The 'Email' and 'Password' fields are no longer highlighted in red, and the 'Login' button is now active and blue. The 'Register' link remains visible below it.

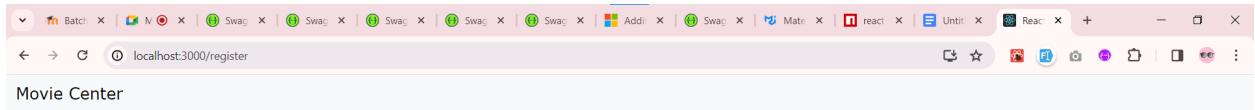
Client side validations



Server side validations



Register with client side validations and server side validations



Welcome to Movie Center

Enter a name

Enter a email

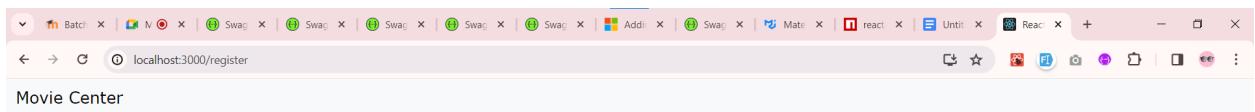
Enter a password

[Register](#)

[Login](#)



client side validations



Welcome to Movie Center

Please enter a username
Enter a name

Please enter a email
Enter a email

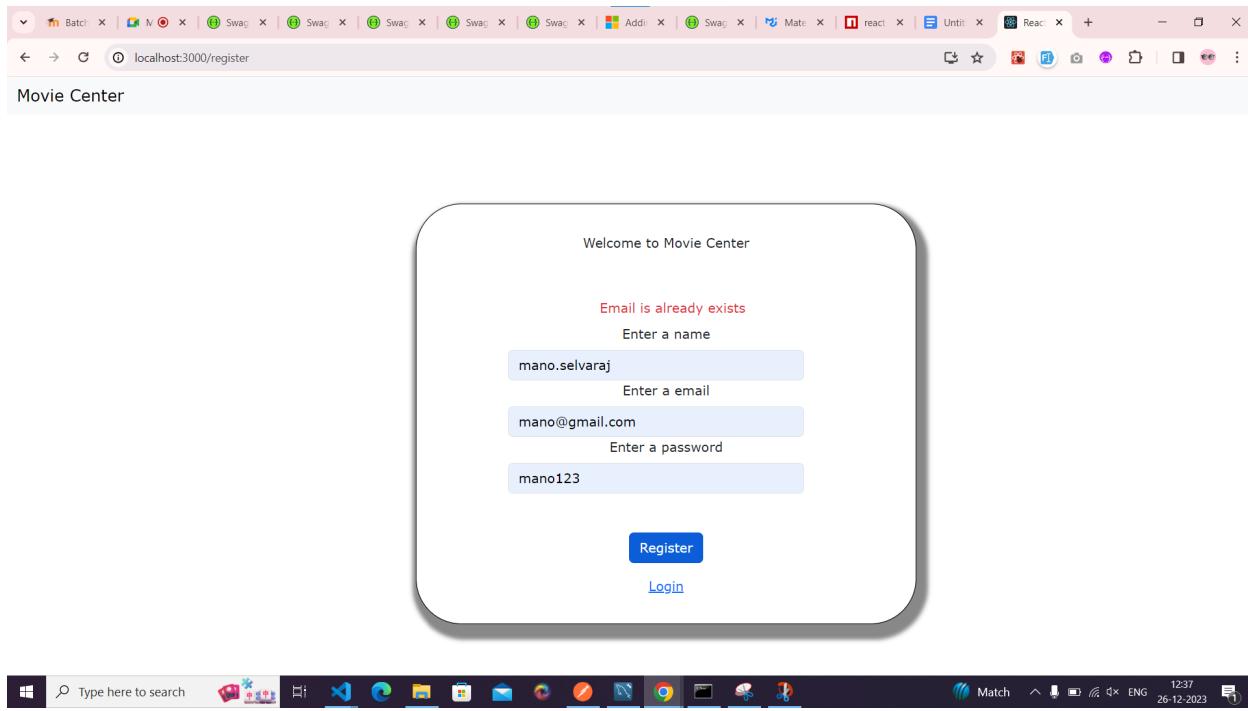
Please enter a password
Enter a password

[Register](#)

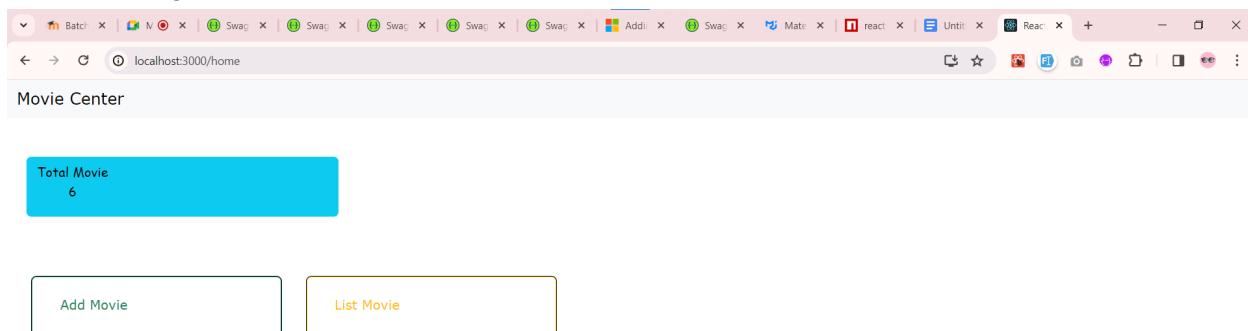
[Login](#)



Server side validations



Home Page Simple Dashboard With checking total number of movie available in database

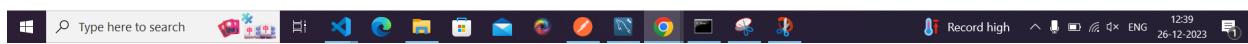


List of movie

A screenshot of a web browser window titled "Movie Center". The URL in the address bar is "localhost:3000/list". The page displays a table of movies with columns: #, Movie Name, Movie Language, Movie Type, Movie Durations, and Action. The table contains three rows:

#	Movie Name	Movie Language	Movie Type	Movie Durations	Action
4	Kiren	Tamil	Action	2.00 Hour	<button>Delete</button> <button>Update</button>
6	Basha	Tamil	Action	2.50 Hour	<button>Delete</button> <button>Update</button>
7	Hello	Hindi	Action	3.00 Hour	<button>Delete</button> <button>Update</button>

An "Add Movie" button is located at the top right of the table area.



Add Movie

A screenshot of a web browser window titled "Movie Center". The URL in the address bar is "localhost:3000/create". The page displays a form for adding a new movie:

Movie Name:

Movie Type:

Movie Language:

Movie Durations:

An "Add Movie" button is located at the bottom right of the form.



Added item listed in list

A screenshot of a web browser window titled "Movie Center". The URL bar shows "localhost:3000/list". The main content area displays a table of movies:

ID	Title	Language	Genre	Duration	Action	Action
10	Kaka muttai	Tamil	Drama	2.15 Hour	Delete	Update
11	Seven Pound	English	Sentimental Drama	2.05 Hour	Delete	Update
12	Hope	Korean	Drama	2.01 Hour	Delete	Update

An "Add Movie" button is located at the top right of the table.



Delete with dialog box

A screenshot of a web browser window titled "Movie Center". The URL bar shows "localhost:3000/list". A modal dialog box is centered over the movie list, asking for confirmation to delete a movie:

Confirm Delete

Make sure you want to delete?

The background movie list is partially visible:

ID	Title	Language	Genre	Duration	Action	Action
10	Kaka muttai	Tamil	Drama	2.15 Hour	Delete	Update
11	Seven Pound	English	Sentimental Drama	2.05 Hour	Delete	Update
12	Hope	Korean	Drama	2.01 Hour	Delete	Update

An "Add Movie" button is located at the top right of the table.

Update the value

Screenshot of a web browser window showing a movie creation form. The URL is localhost:3000/create/4.

The form fields are:

- Movie Name: Kiren
- Movie Type: Action
- Movie Language: Tamil
- Movie Durations: 2.00 Hour

An "Update" button is located at the bottom right of the form.

