PinkBox

PinkBox is e-commerce movie streaming service to allow its customers to stream any movies at a discounted price. The challenge, at present time, is that existing movie streaming providers, NetFlix, Hulu, AmazonPrime etc. require monthly subscription irrespective of how many movies one can watch. Each provides is contracted to host selected set of movies. MovieMart, virtual store, negotiated will all providers at a discounted prices for each view and provides a unified UI to select any movie from any of the streaming providers.

PinkBox is built using MERN tech stack. It has three projects, each to support frontend, backend and Inventory.

Frontend: It is built on using react.js and makes calls to endpoints exposed by the backend.

Backend: It has hosted on Express server which is connected to MongoDB hosted on google cloud. Express server exposes various endpoints for frontend to all to get data in json format from MongoDB

Inventory: This project is meant for loading movies to MongoDB database.

Backend Design:

MongoDB is chosen database to persist data. There are two schemas (model) to support this e-commerce application.

Schema 1: movies

To store all the movies

Schema 2: users

To store users and their shopping cart details

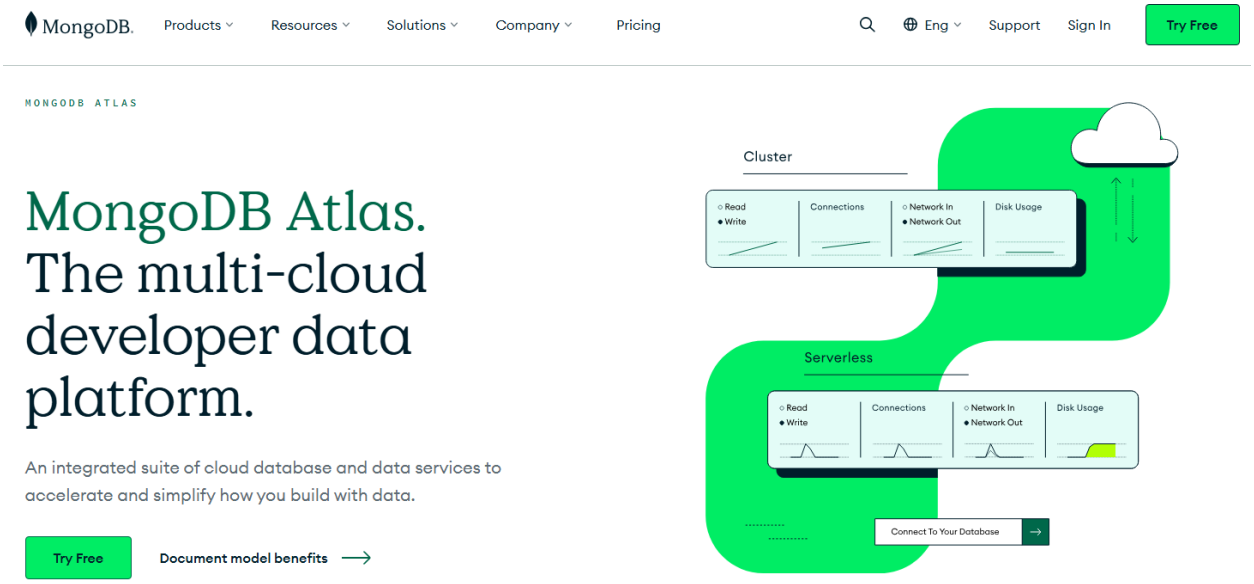
MongoDB hosting:
MongoDB Atlas is a chosen as it's a fully managed cloud database service provided by MongoDB. Having MongDB on the cloud enables everyone to access the database on their laptop and loading all the movies. Any changes require redo every time for everyone in the project team. Any changes made to the database or the data is a one time task and it is available to all the team mebers. It saves time and effort in deleivering the project.

Deploy MongoDB database on the cloud.

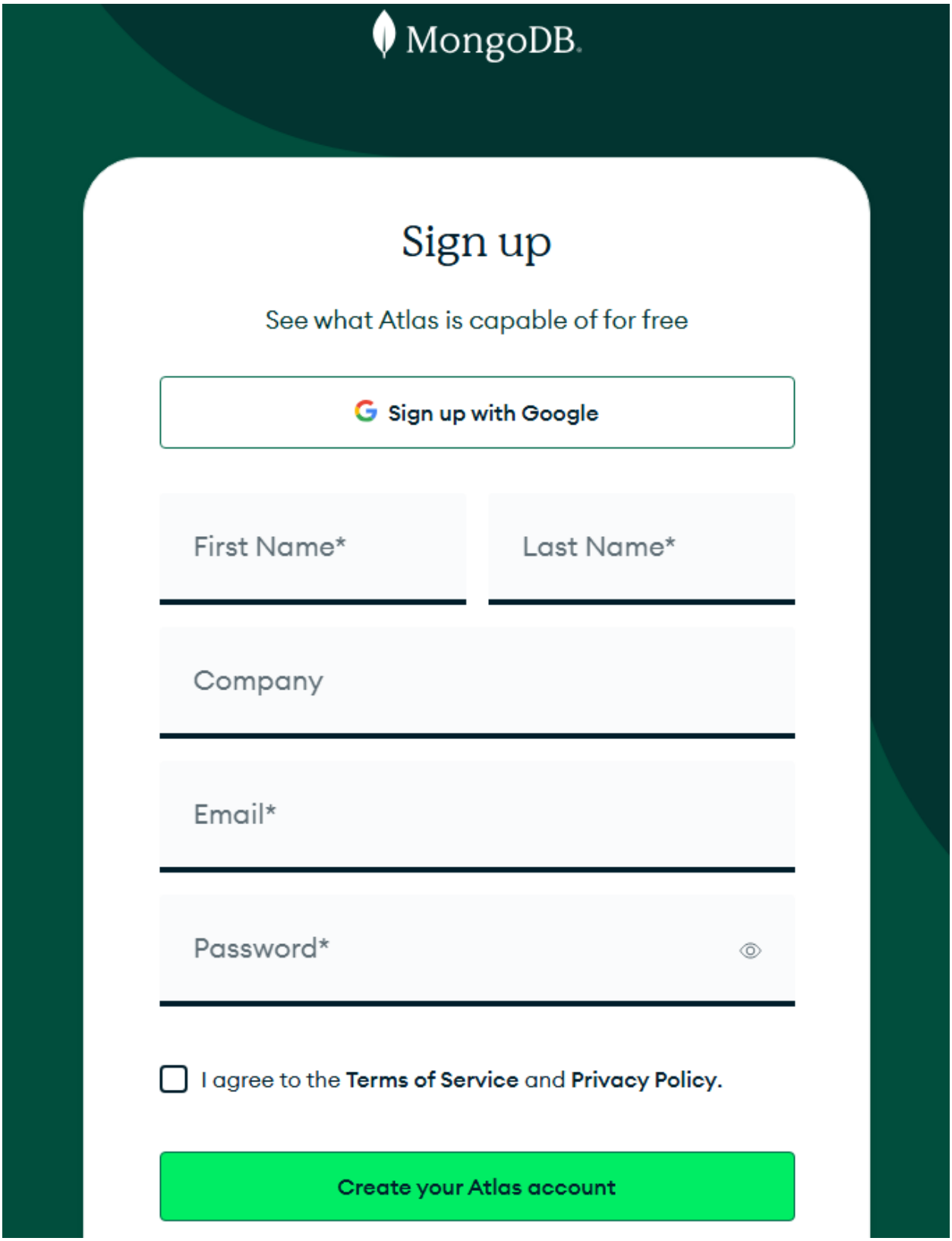
MongoDB Atlas on the cloud is a free service with limited space. Google is the chosen cloud provide and therefore to deploy the database on Google cloud, a gmail account is required.

Step 1: Deploy MongDB database

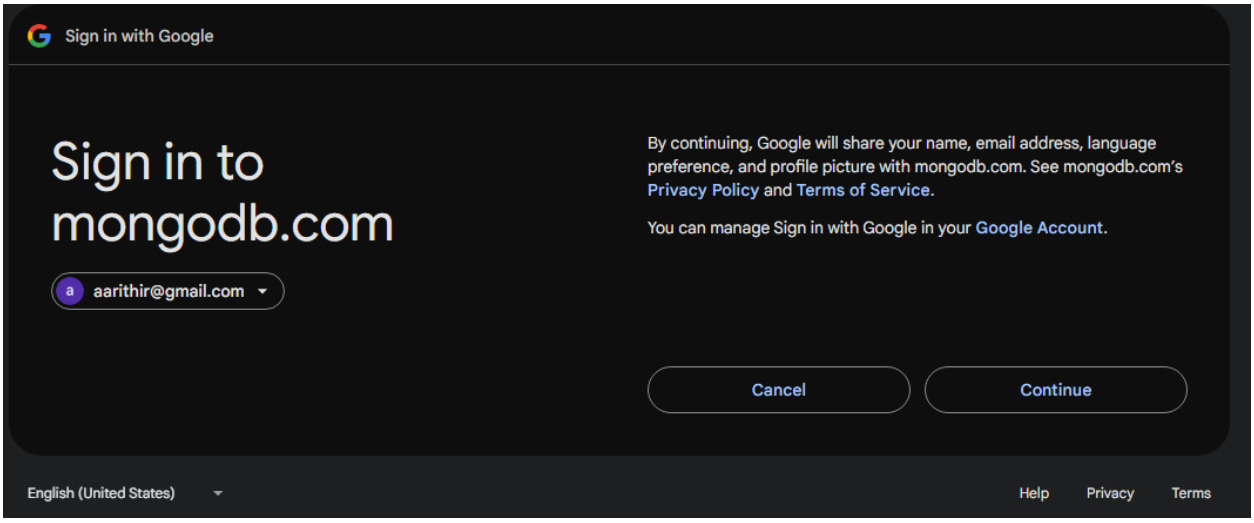
The instructions to deploy the MongoDB Atlas on the cloud is available at https://www.mongodb.com/atlas



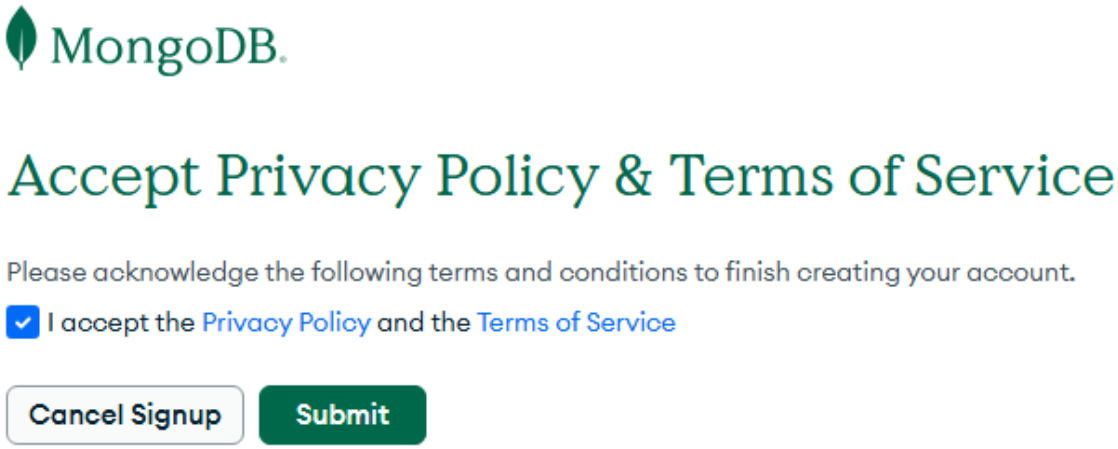
Click on “Try Free”



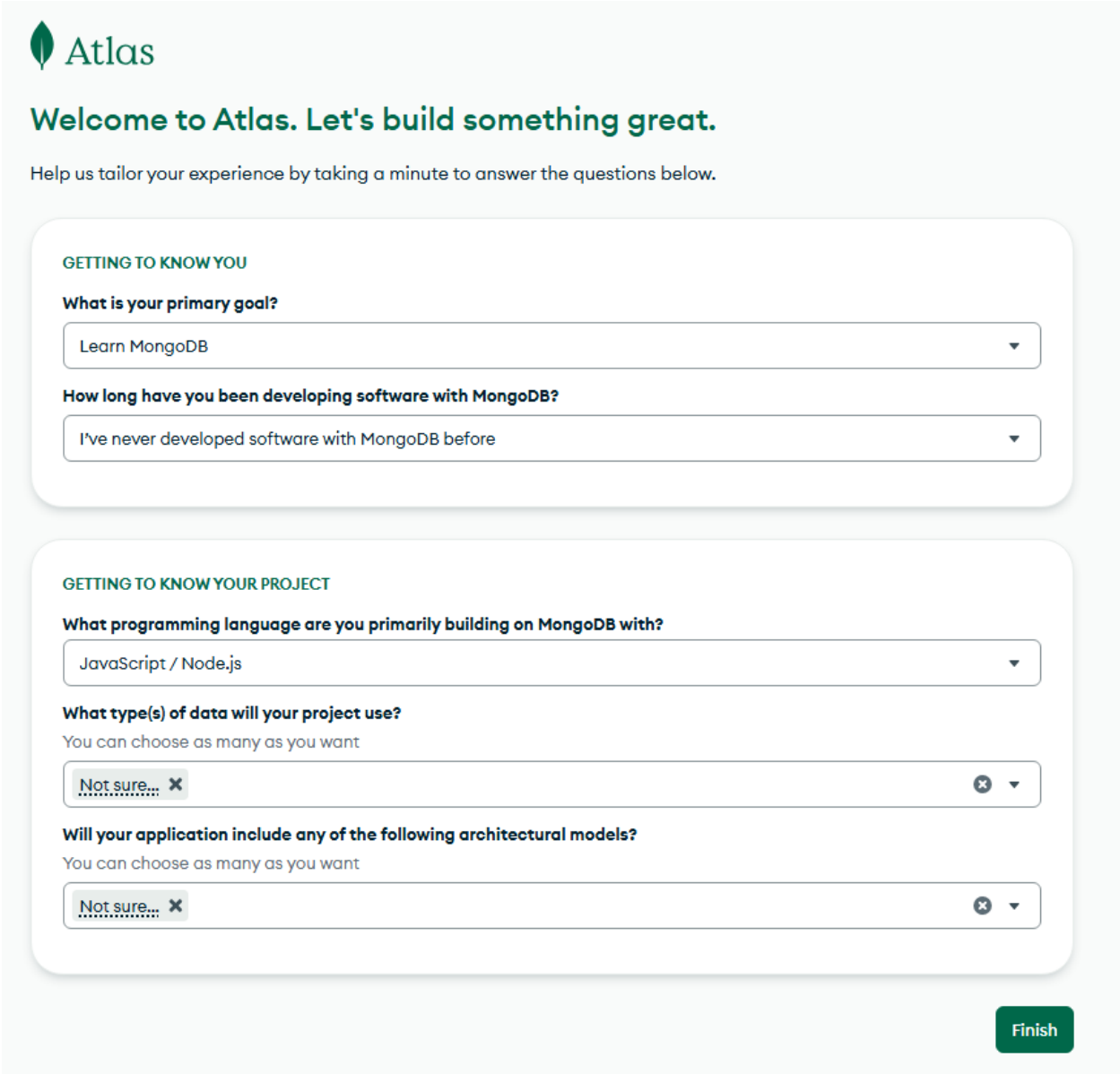
Click on “Sign up with Google”



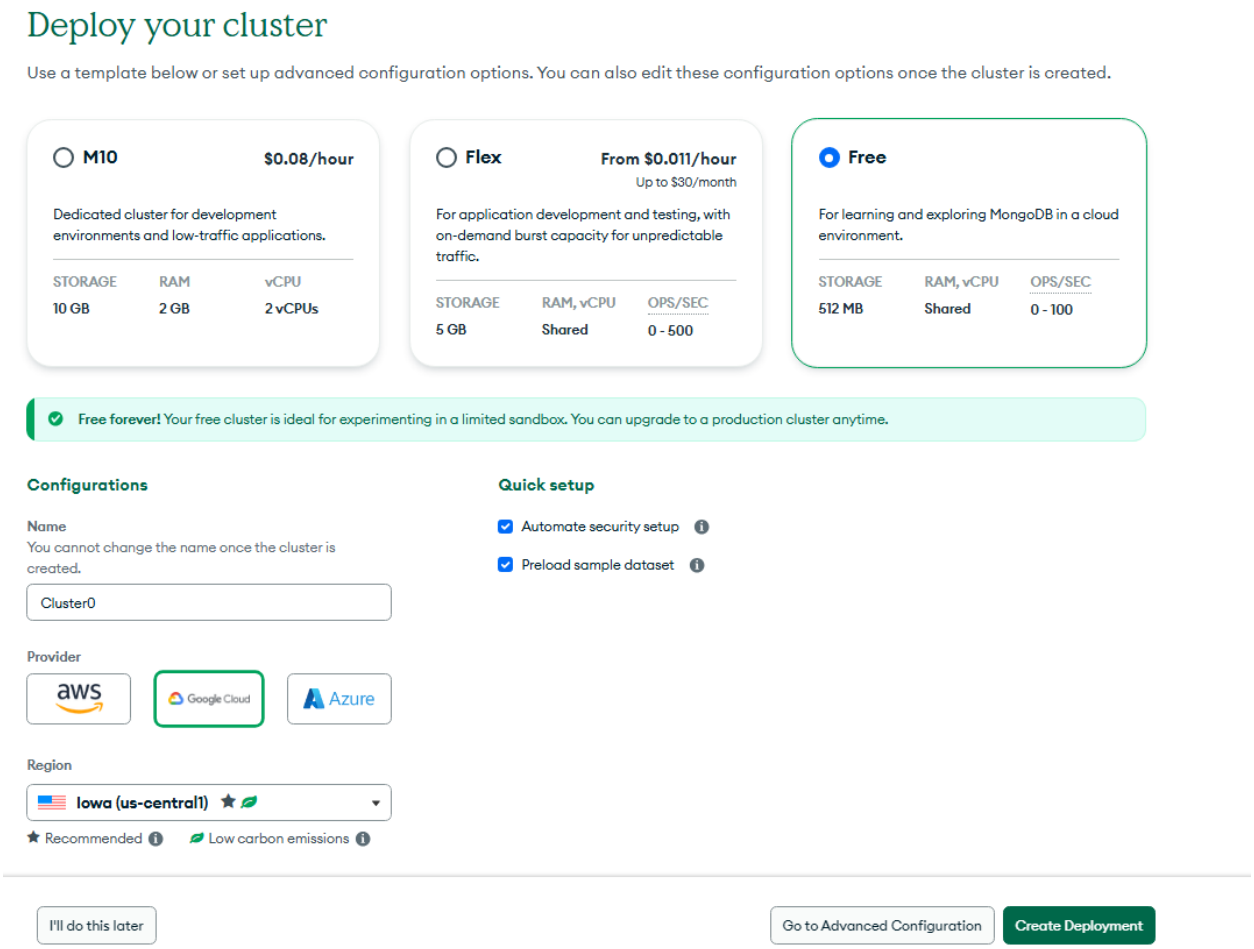
Click on “Continue”



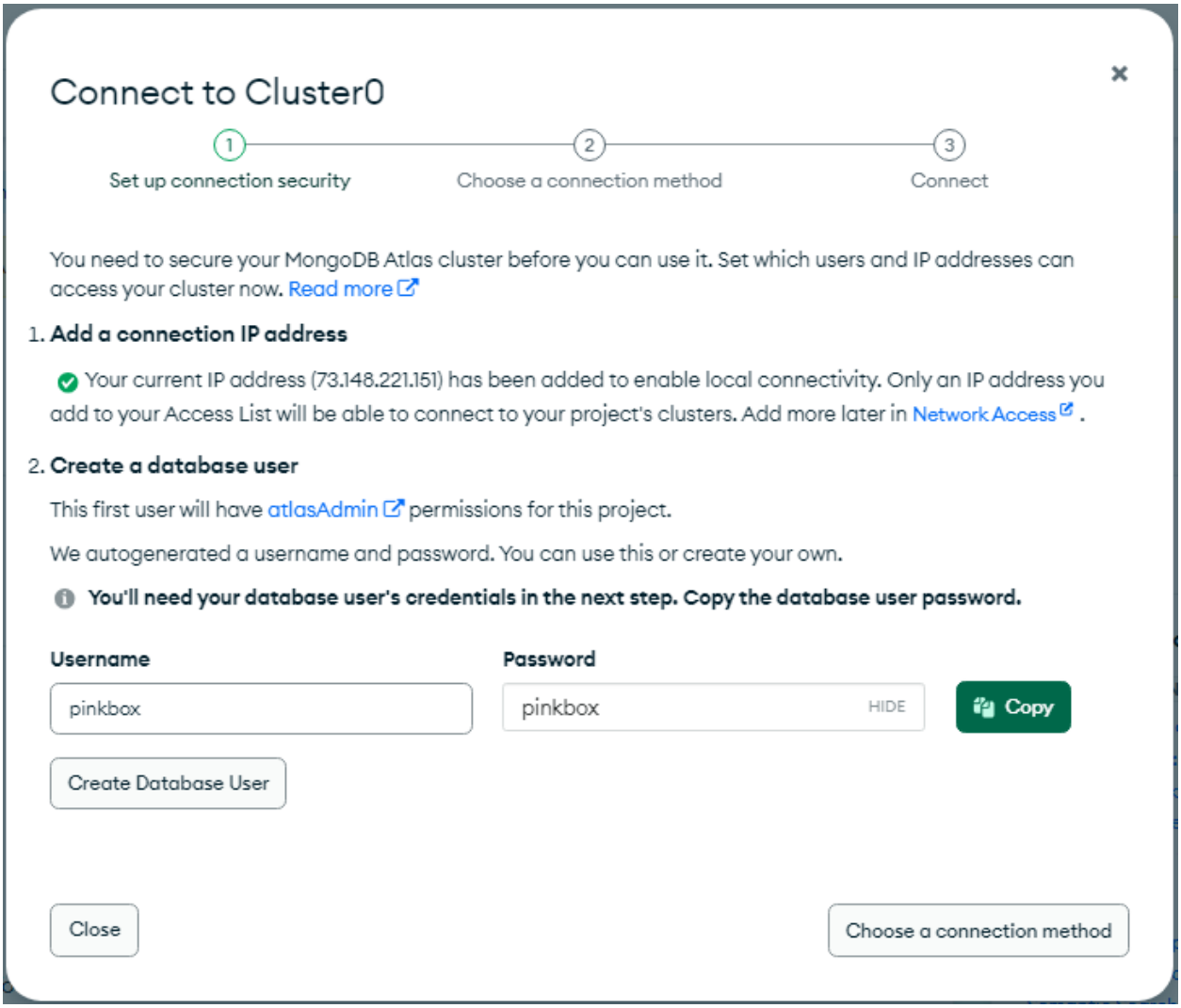
Accept and submit



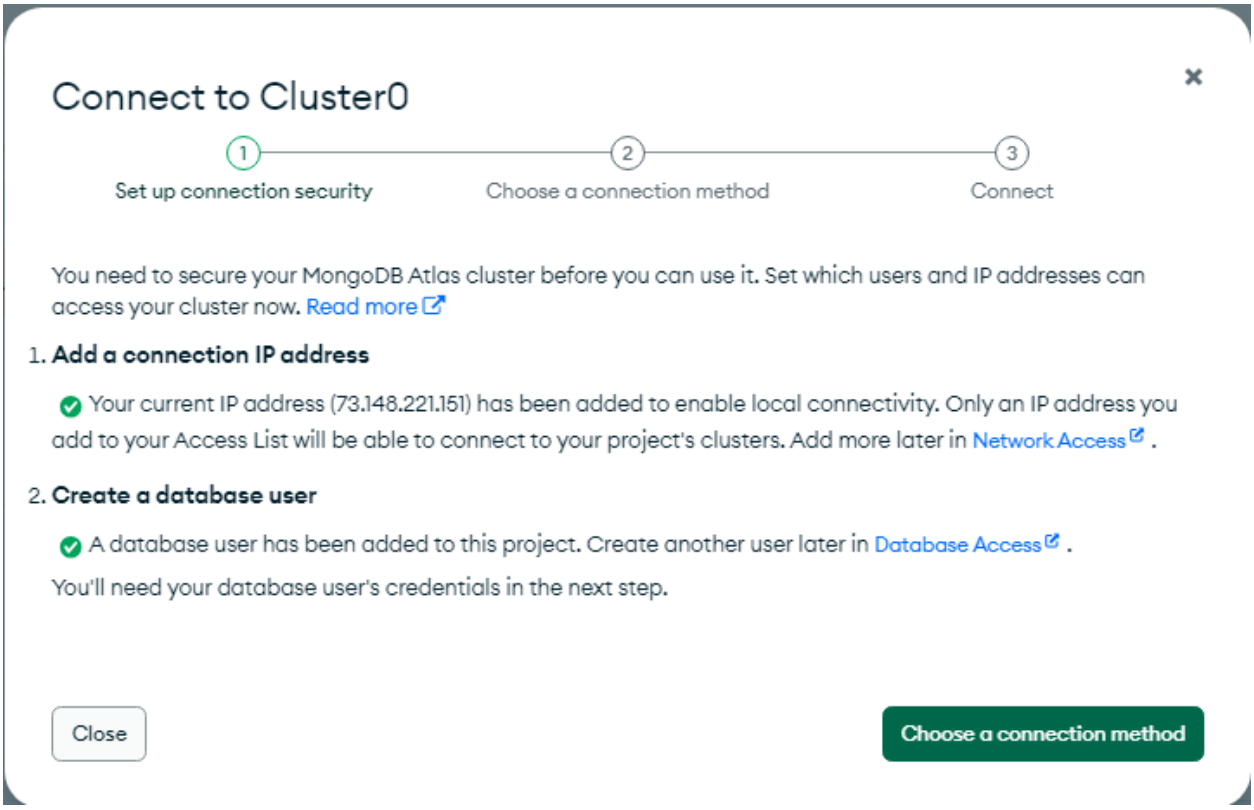
Answer the questions and “Finish”



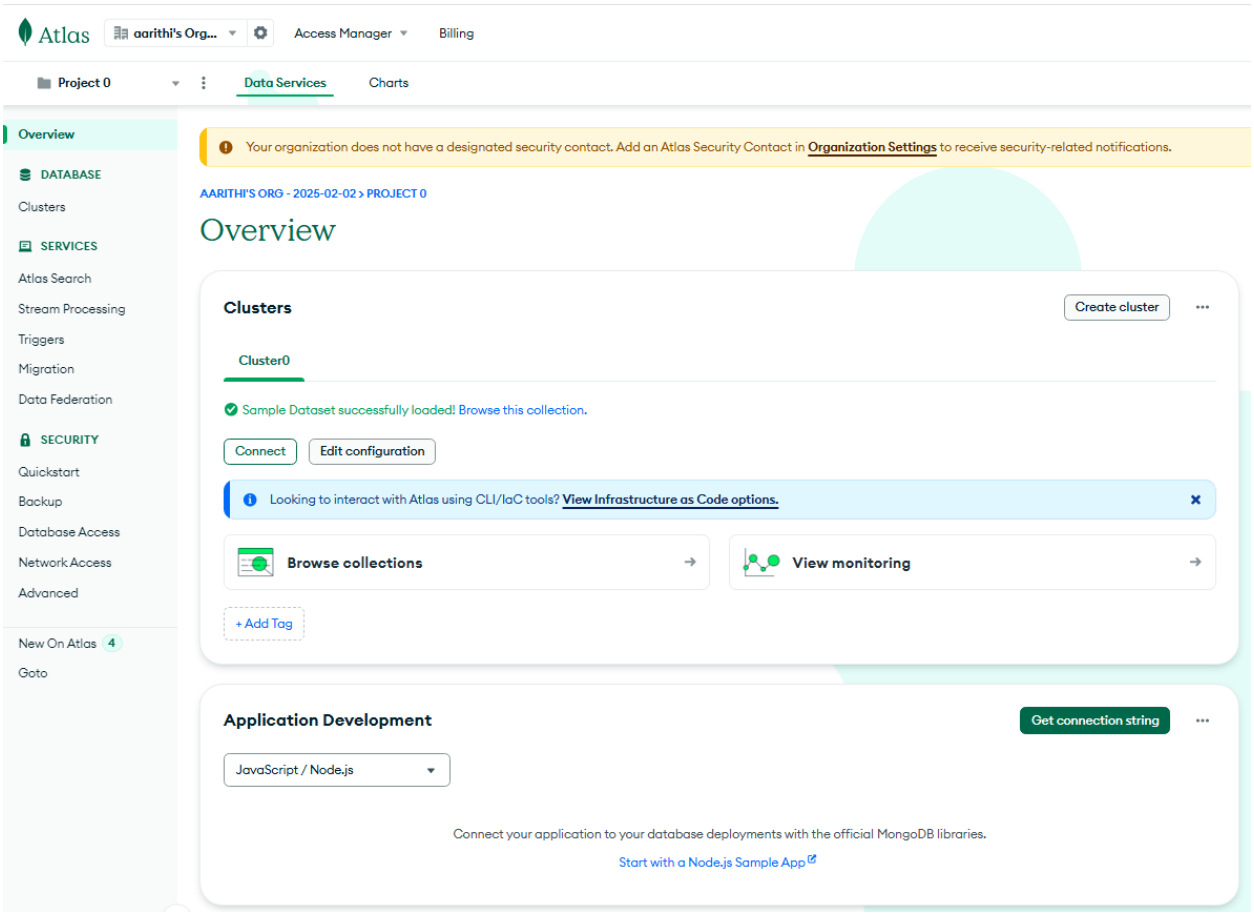
Select “Free” and “Google Cloud”
Select the Region and leave the rest to Default
click on “Create Deployment” and
Note that storage allowed for free is only 512MB.



Pick username (pinkbox) and password (pinkbox)
Click on “Create Database User”

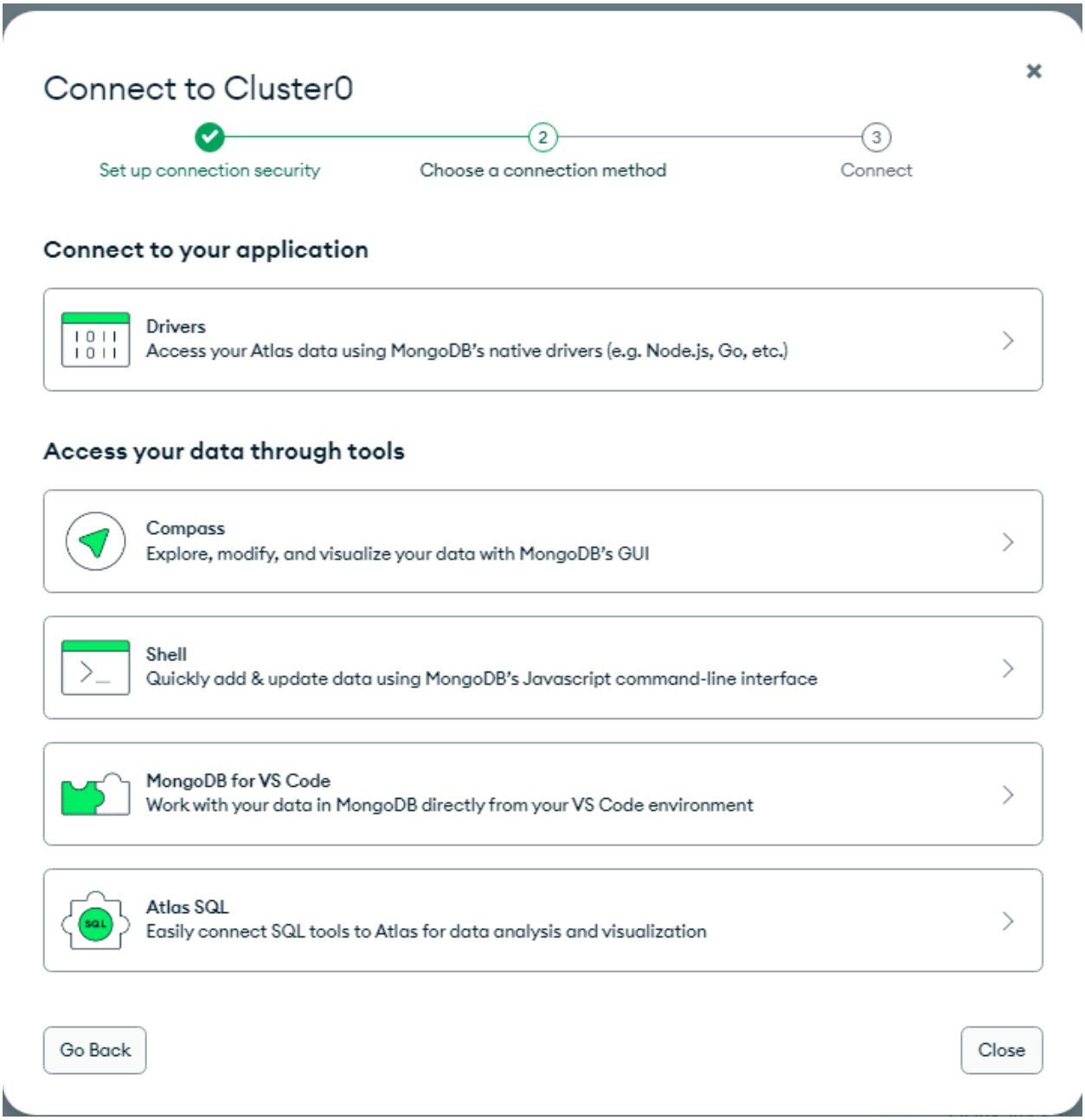


Click on “Close”

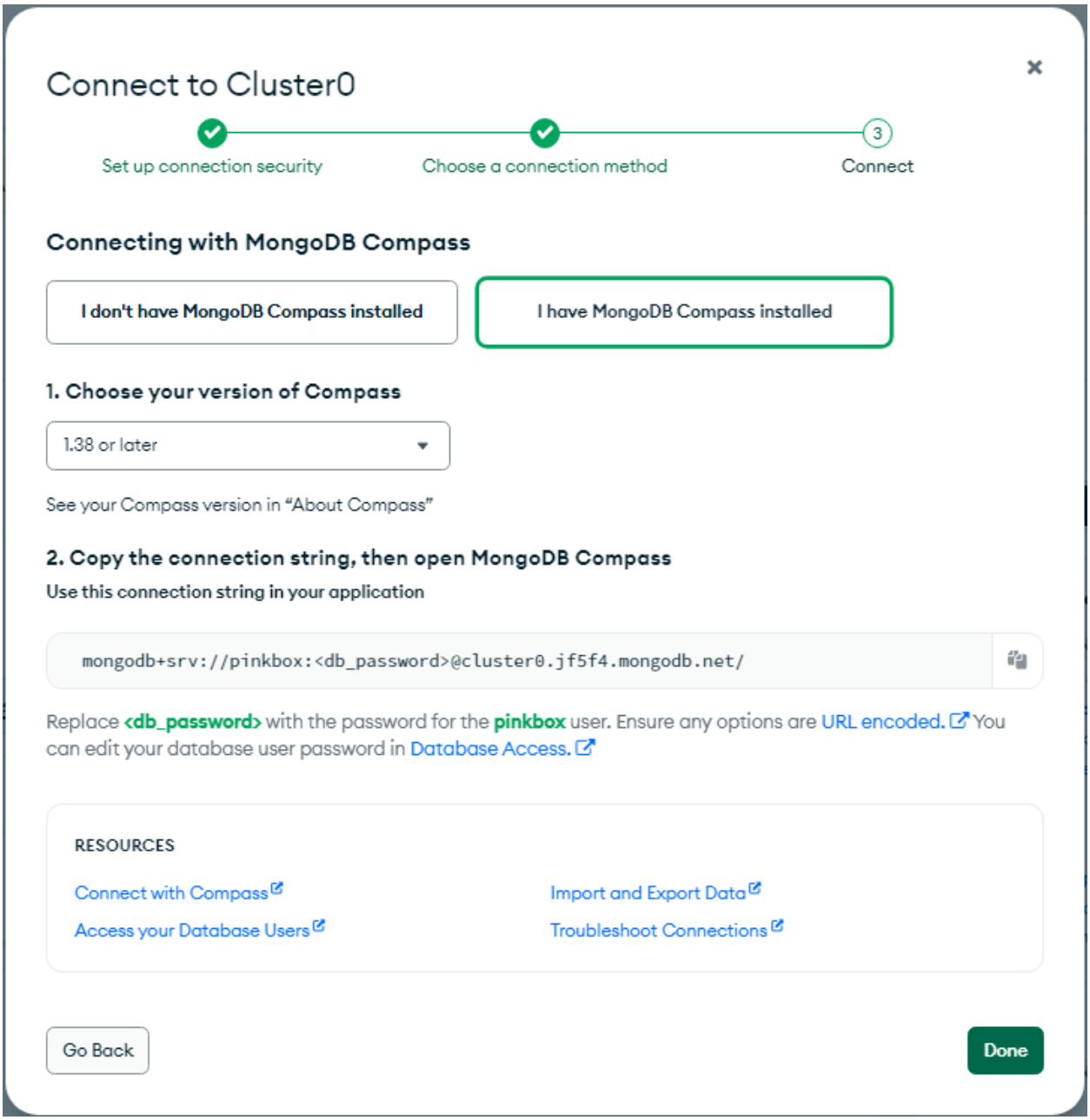


Step 2: To get the connection string to connect to the database just created,

Click “Connect”



Click on “Compass”



Copy the connection string:

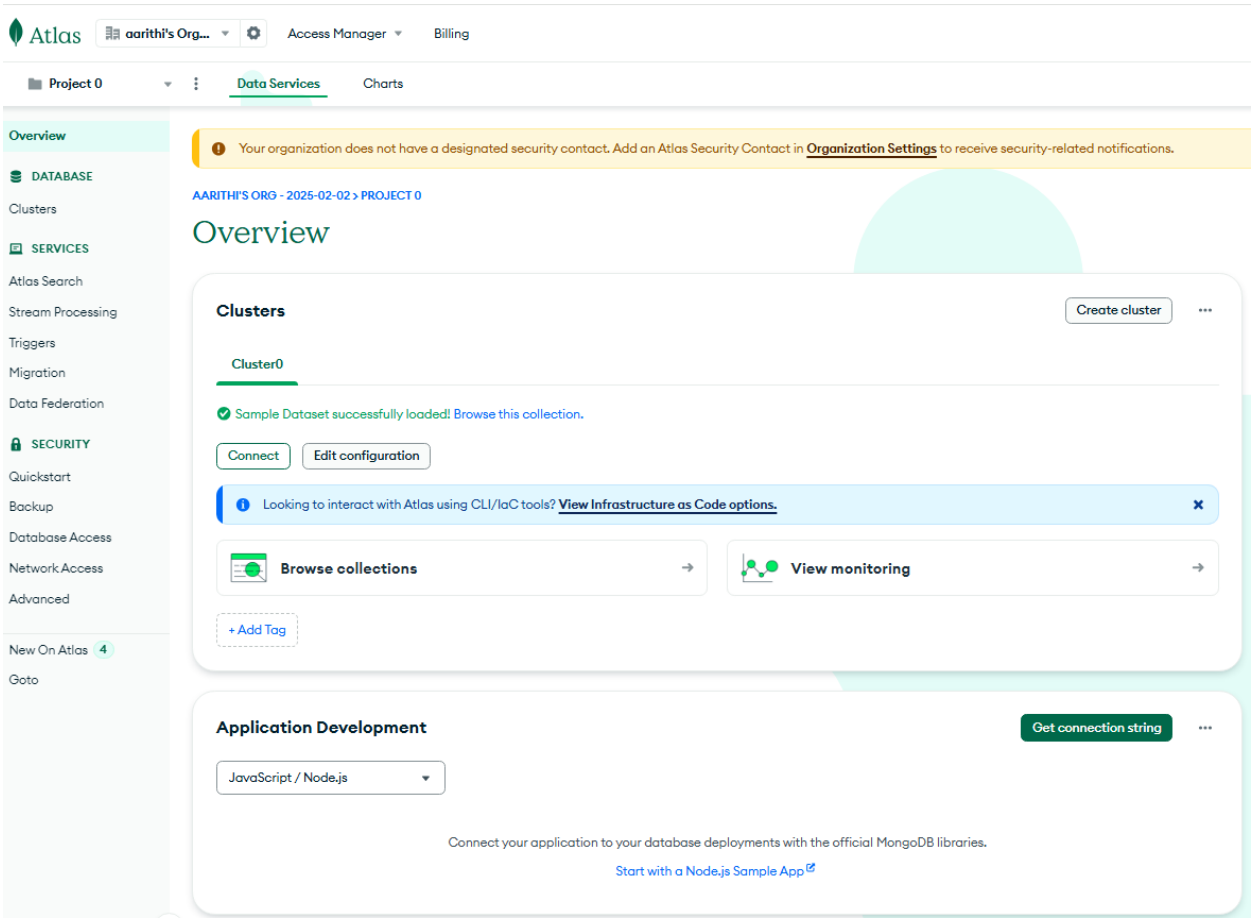
mongodb+srv://pinkbox:<db_password>@cluster0.jf5f4.mongodb.net/

This is the string to be used when connecting the MongoDb database in Express server.
Replace <d_password> with the actual password "pinkbox"

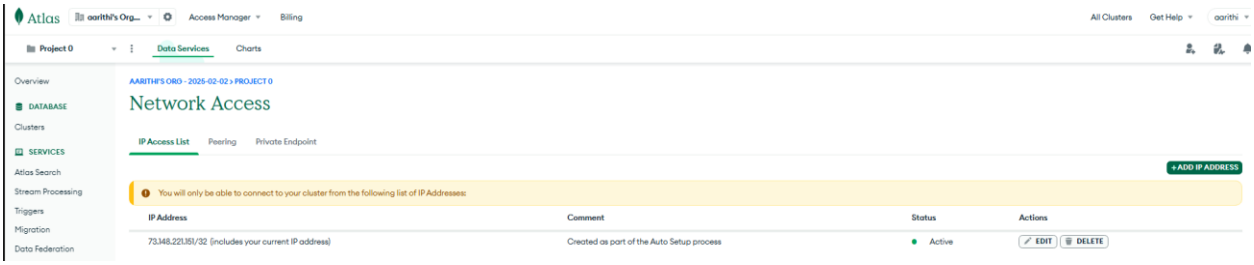
mongodb+srv://pinkbox:pinkbox@cluster0.jf5f4.mongodb.net/

Click on “Done”

Step 3: To access the database from any host/IP address:

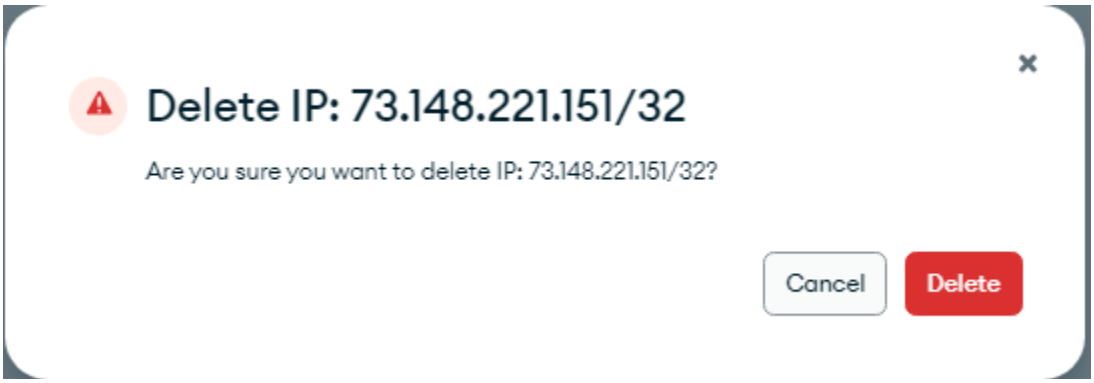


Click on “Network Access”

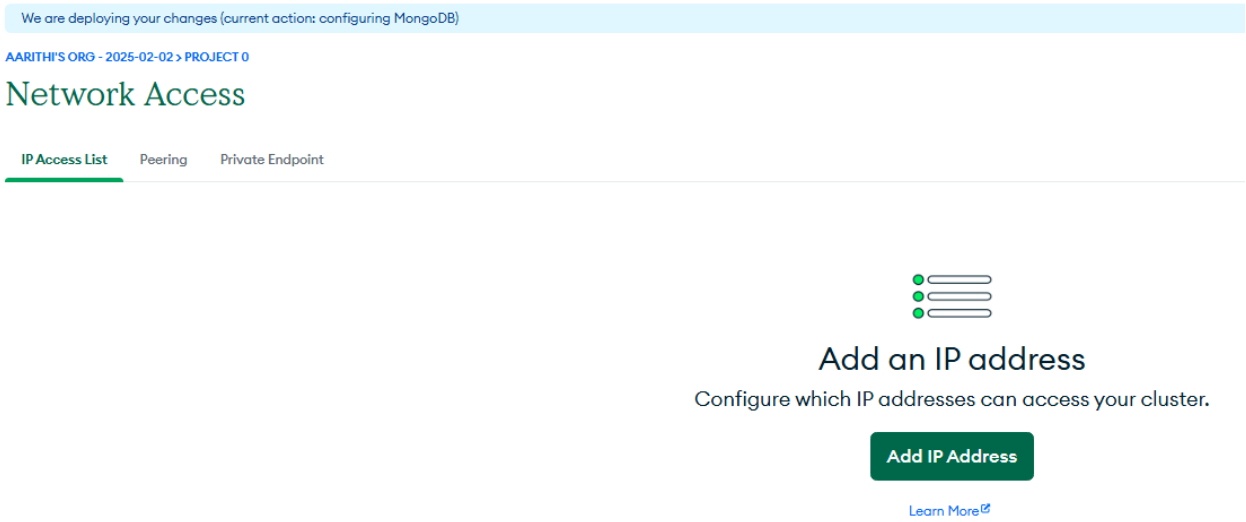


Database can be accessed only from host/ip address: 73.148.221.151

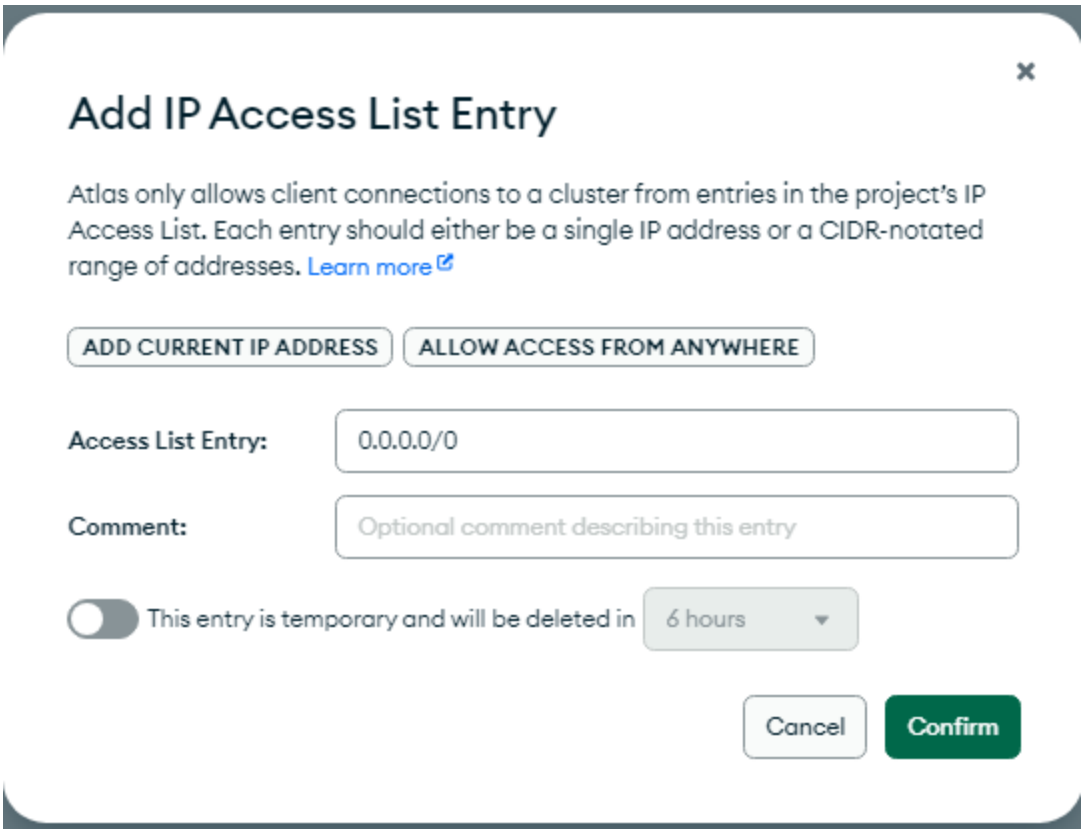
To allow any IP to access the database, deleted the existing one by clicking “DELETE”



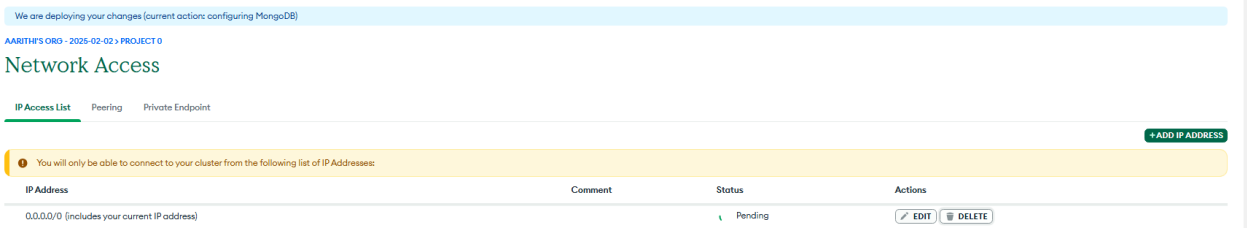
Confirm Delete.



Click on “Add IP Address”



Enter "0.0.0.0/0" to all allow IP addresses and click on "Confirm"

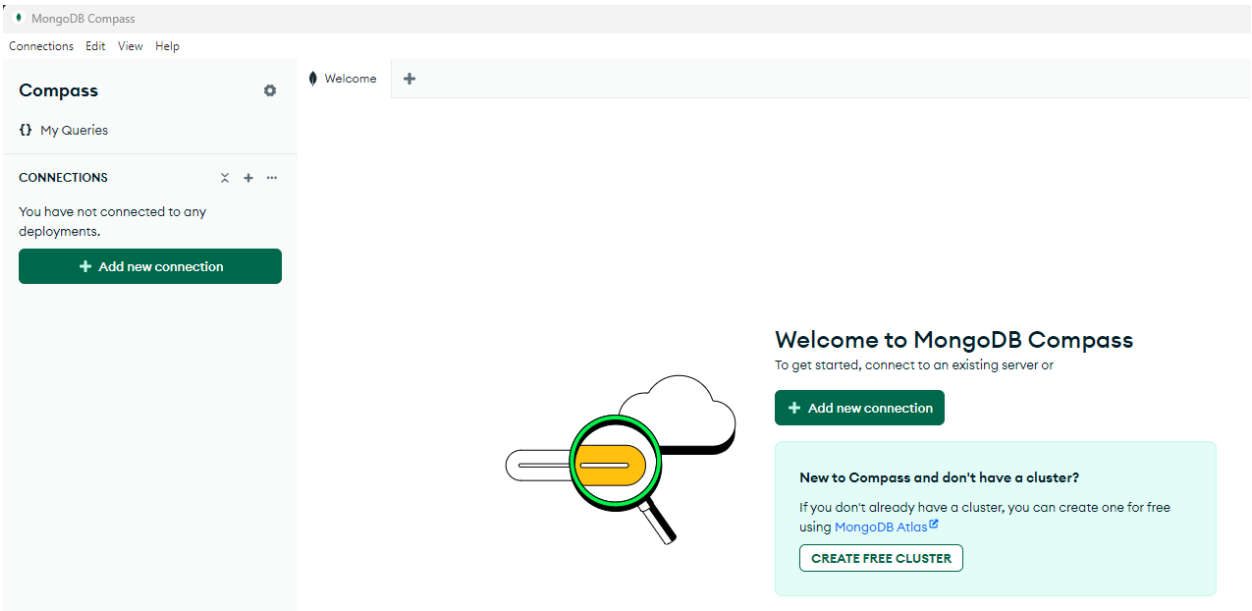


This completes deploying MongoDB database on Google Cloud and grating database access from any host/laptop/IP address.

MongoDB Compass:

MongoDB Compass is a free, open-source graphical user interface (GUI) for exploring, analyzing, and manipulating MongoDB data. MongoDB Atlas deployed on Goole Cloud requires google account credential to access UI. Whereas Compass requires only connection string to connect to the database and it has more user friendly UI to manage database.

Install MongDB Compass:
Step 1: Download MangoDB Compass at https://www.mongodb.com/try/download/compass



Step 2. Add database

Click on “Add new conection”
mongodb+srv://pinkbox:pinkbox@cluster0.jf5f4.mongodb.net/

MongoDB Collections:

There are two collections are defined and they are:

1. Movie

```
// Schema for creating Movie|

const Movie = mongoose.model("Movie", {
  id: { type: Number, required: true },
  title: { type: String, required: true },
  description: { type: String, required: true },
  genre: { type: String, required: true },
  cost: { type: Number },
  image: { type: String, required: true },
  dor: { type: Date, required: true },
  streaming_url: { type: String, required: true },
  new_release: { type: String, required: true, default: false },
  available: { type: Boolean, default: true },
});
```

2. Users

```
// Schema for creating user model

const Users = mongoose.model("Users", {
  name: { type: String },
  email: { type: String, unique: true },
  password: { type: String },
  cartData: { type: Object },
  date: { type: Date, default: Date.now() },
});
```

PinkBox Setup Guide:

Step 1: Clone the PinkBox repository at http://github.com/aarithi123/pinkbox

The repository has four folders.

1. backend
2. frontend
3. inventory
4. documentation

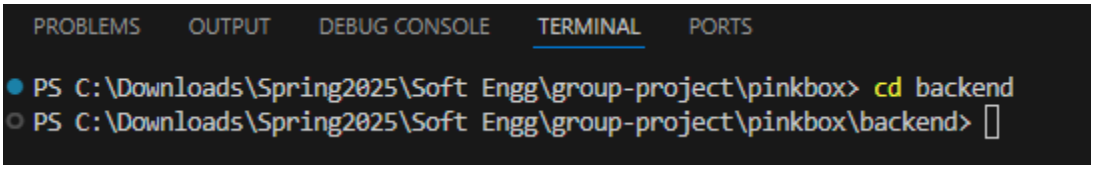
Before starting frontend, please make sure backend is started first ad frontend requires endpoints exposed by backend to add/update/delete documents from MongoDB document database.

Backend:

How to restart the backend application?

Open the folder C:\Downloads\Spring2025\Soft Engg\group-project\pinkbox (pinkbox project) in vscode OR navigate to <download location/pinkbox/backend

Open the Terminal and switch to backend project folder.



Install backend application dependencies.

Pre-requisite: Node.js

To install Node.js, go to https://nodejs.org/en/download

Download windows installer and install “node.js”

To install dependencies, type the below command in the terminal

npm install

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

PS C:\Downloads\Spring2025\Soft Engg\group-project\pinkbox> cd backend
PS C:\Downloads\Spring2025\Soft Engg\group-project\pinkbox\backend> npm install

added 138 packages, and audited 139 packages in 15s

14 packages are looking for funding
  run `npm fund` for details

10 vulnerabilities (3 low, 1 moderate, 5 high, 1 critical)
```

```
To address all issues, run:
  npm audit fix

Run `npm audit` for details.
PS C:\Downloads\Spring2025\Soft Engg\group-project\pinkbox\backend>
```

```
To address all issues, run:
  npm audit fix

Run `npm audit` for details.
PS C:\Downloads\Spring2025\Soft Engg\group-project\pinkbox\backend>
```

Dependencies:
1. Install Express Server:
npm install express

2. Install jwt package for user auth:
npm install jsonwebtoken

3. Install mongodb package
npm install mangoose

4. Install multer for storing images
npm install multer

Note that the free version of MongoDB Atlas has only 512MB storage. The movie posters takes more space and therefore mutler is used to store on folder on local storage.

5. Install cors to grant permission to the frontend application to access the backend application
npm install cors

(CORS stands for Cross-Origin Resource Sharing. It is a security feature implemented by web browsers to control how web pages can request resources from different origins (domains) than their own)

start the Express server:

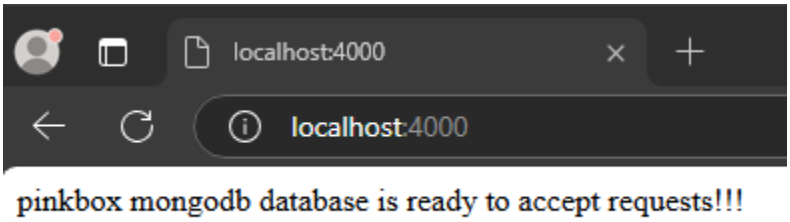
node index.js

It connects to mongodb database and exposes endpoints

```
PS C:\Downloads\Spring2025\Soft Engg\group-project\pinkbox\backend> node index.js
(node:15744) [DEP0040] DeprecationWarning: The `punycode` module is deprecated. Please use a userland alternative instead.
(Use `node --trace-deprecation ...` to show where the warning was created)
Server Running on port 4000
PS C:\Downloads\Spring2025\Soft Engg\group-project\pinkbox\backend> node index.js
(node:1524) [DEP0040] DeprecationWarning: The `punycode` module is deprecated. Please use a userland alternative instead.
(Use `node --trace-deprecation ...` to show where the warning was created)
Server Running on port 4000
```

Express server is now listening on port 4000

To check Express server is ready to accept requests, http://localhost:4000



Endpoints:

1. login endpoint for login the user and sending auth-token (jwt)

/login

Request:
Response:

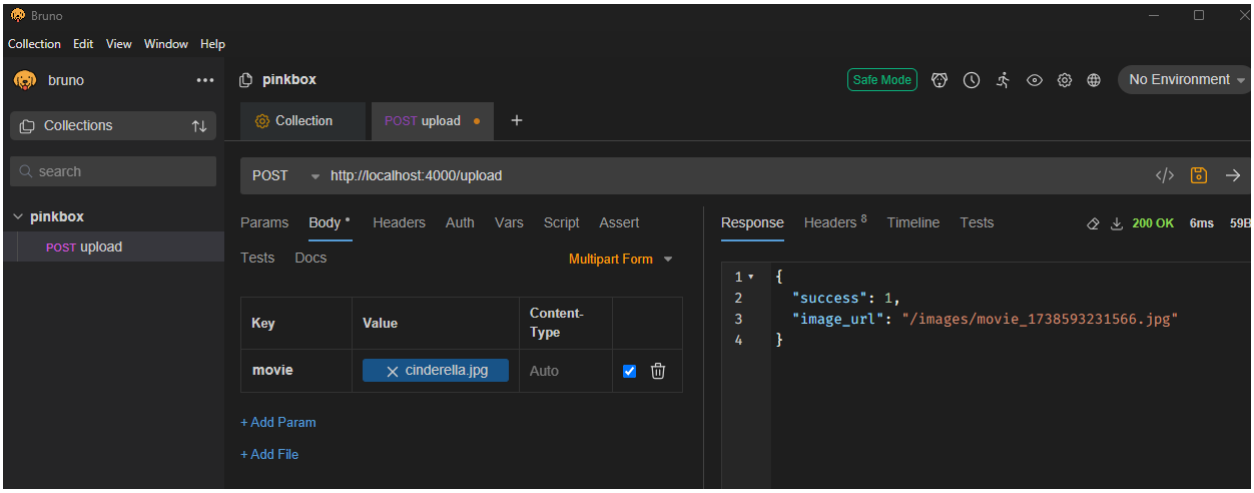
Testing:

2. upload movie posters
/upload

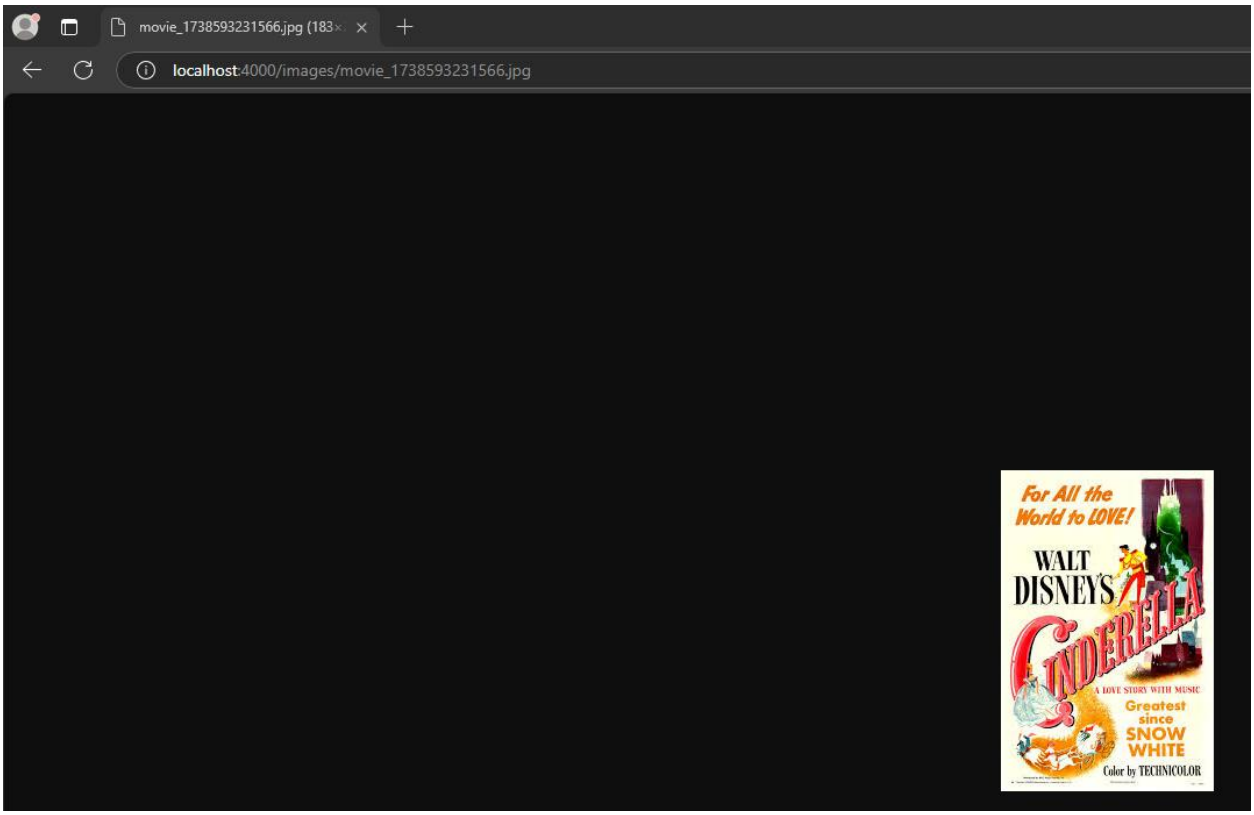
The free version of “Thunder Client” does not allow imageFile POST.

Please install open source Bruno from https://www.usebruno.com/downloads

Testing:



Once image upload is successful, it can ben viewed at http://localhost:4000/<image_url>
http://localhost:4000/images/movie_1738593231566.jpg

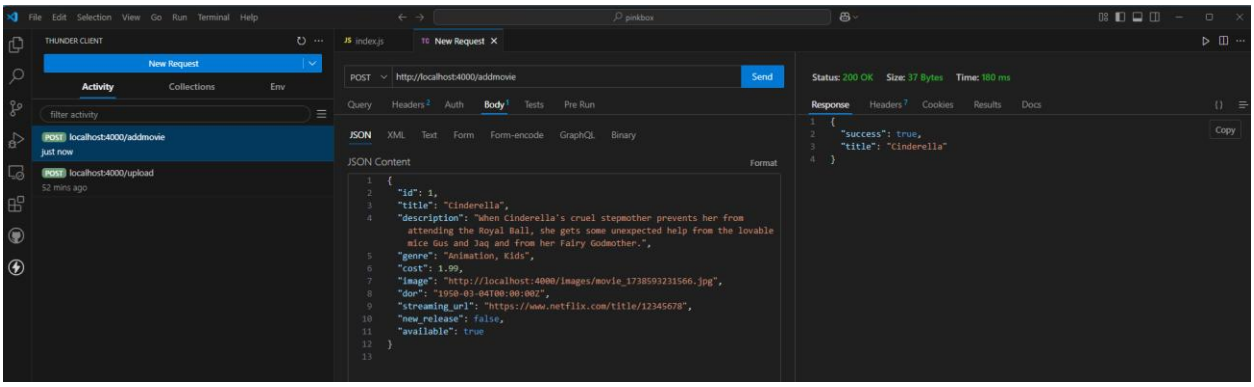


/addmovie:
request:

{
  "id": 1,
  "title": "Cinderella",
  "description": "When Cinderella's cruel stepmother prevents her from attending the Royal Ball, she gets some unexpected help from the lovable mice Gus and Jaq and from her Fairy Godmother.",
  "genre": "Animation, Kids",
  "cost": 1.99,
  "image": "http://localhost:4000/images/movie_1738593231566.jpg",
  "dor": "1950-03-04T00:00:00Z",
  "streaming_url": "https://www.netflix.com/title/12345678",
  "new_release": false,
  "available": true
}

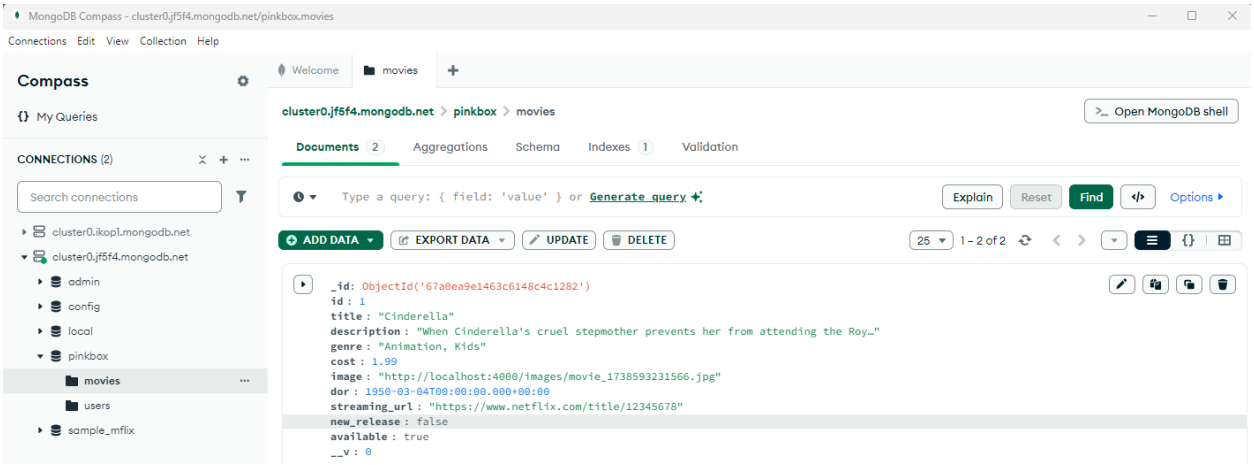
response:
{
  "success": true,

"title": "Cinderella"
}



MongoDB Compass:

Success scenario



Failure scenario:

response:
{
"success": false,
"error": " Title must be unique. Duplicate title found."
}

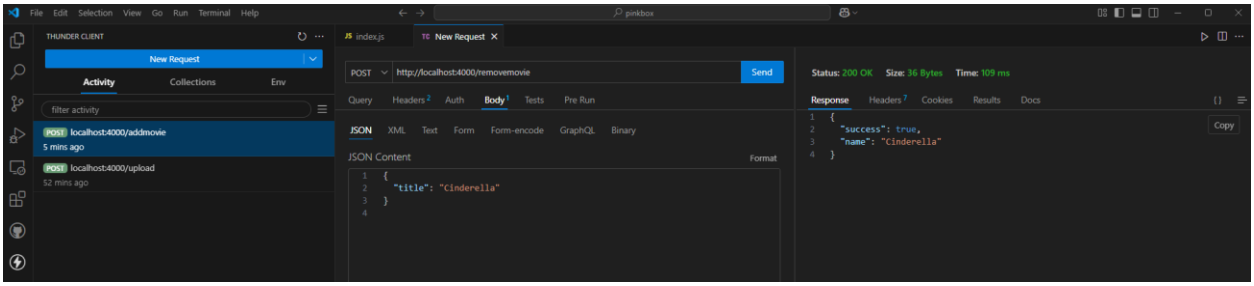
/removemovie:
request:

{

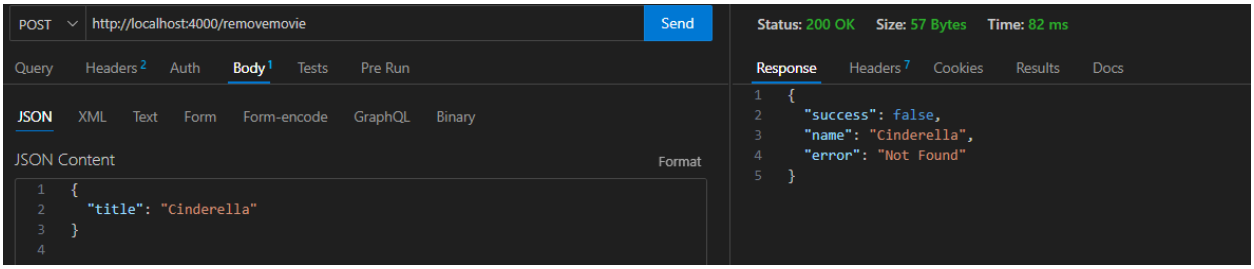
"title": “Cinderella”
}

response:
{

"success": true,
"title": "Cinderella"
}



If movie not found to delete:



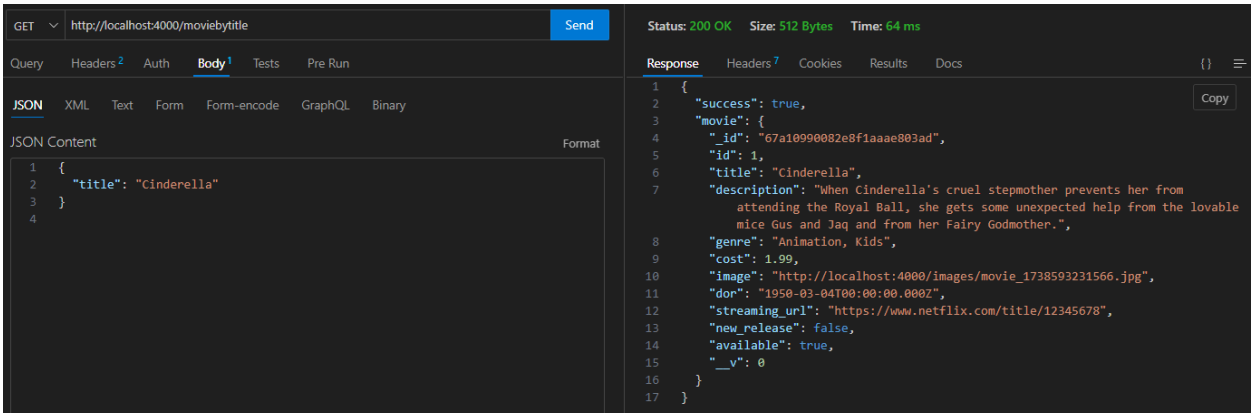
/moviebytitle:
request:

{

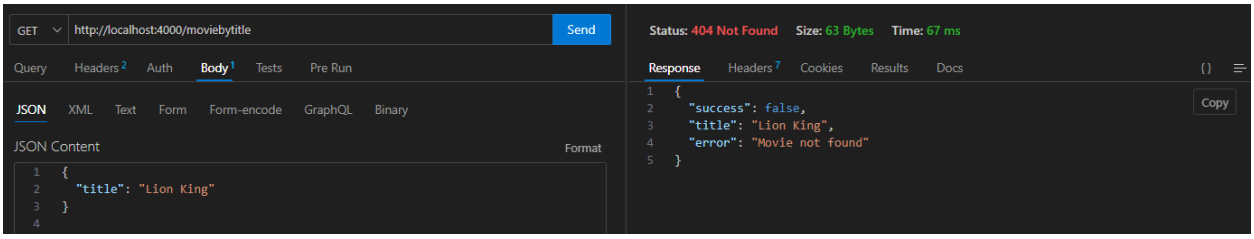
"title": “Cinderella”
}

response:
{
"success": true,
"movie": {
"_id": "67a10990082e8f1aaae803ad",
"id": 1,
"title": "Cinderella",
"description": "When Cinderella's cruel stepmother prevents her from attending the Royal Ball, she gets some unexpected help from the lovable mice Gus and Jaq and from her Fairy Godmother.",
"genre": "Animation, Kids",
"cost": 1.99,
"image": "http://localhost:4000/images/movie_1738593231566.jpg",
"dor": "1950-03-04T00:00:00.000Z",
"streaming_url": "https://www.netflix.com/title/12345678",
"new_release": false,

"available": true,
"__v": 0
}



Movie not found condition:



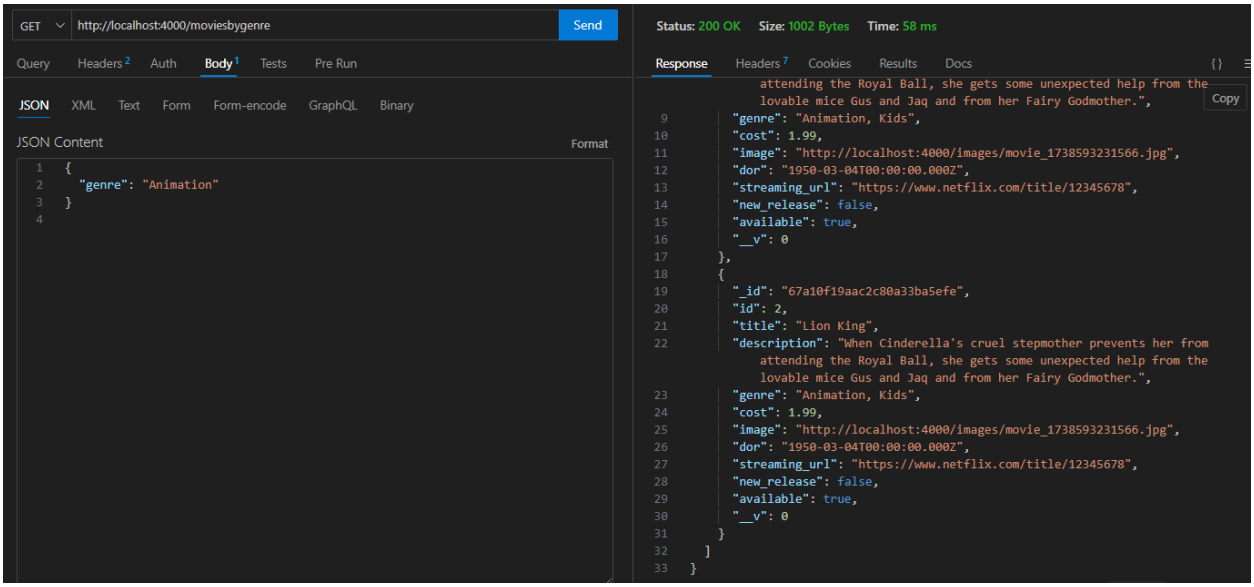
/moviesbygenre:
request:

{

"genre": “Animation”
}

response:
{
"success": true,
"movies": [
{
"_id": "67a10990082e8f1aaae803ad",
"id": 1,
"title": "Cinderella",
"description": "When Cinderella's cruel stepmother prevents her from attending the Royal Ball, she gets some unexpected help from the lovable mice Gus and Jaq and from her Fairy Godmother.",
"genre": "Animation, Kids",
"cost": 1.99,
"image": "http://localhost:4000/images/movie_1738593231566.jpg",

"dor": "1950-03-04T00:00:00.000Z",
"streaming_url": "https://www.netflix.com/title/12345678",
"new_release": false,
"available": true,
"__v": 0
},
{
"_id": "67a10f19aac2c80a33ba5efe",
"id": 2,
"title": "Lion King",
"description": "When Cinderella's cruel stepmother prevents her from attending the Royal Ball, she gets some unexpected help from the lovable mice Gus and Jaq and from her Fairy Godmother.",
"genre": "Animation, Kids",
"cost": 1.99,
"image": "http://localhost:4000/images/movie_1738593231566.jpg",
"dor": "1950-03-04T00:00:00.000Z",
"streaming_url": "https://www.netflix.com/title/12345678",
"new_release": false,
"available": true,
"__v": 0
}
]
}



Movies not found condition:

GET http://localhost:4000/moviesbygenre Send

Query Headers 2 Auth Body 1 Tests Pre Run

JSON XML Text Form Form-encode GraphQL Binary

JSON Content Format

```
{
  "genre": "Thriller"
}
```

Status: 404 Not Found Size: 68 Bytes Time: 54 ms

Response Headers 7 Cookies Results Docs

Copy

```
{
  "success": false,
  "error": "No movies found with the specified genre"
}
```

GET http://localhost:4000/moviebytitle Send

Query Headers 2 Auth Body 1 Tests Pre Run

JSON XML Text Form Form-encode GraphQL Binary

JSON Content Format

```
{
  "title": "Lion King"
}
```

Status: 404 Not Found Size: 63 Bytes Time: 67 ms

Response Headers 7 Cookies Results Docs

Copy

```
{
  "success": false,
  "title": "Lion King",
  "error": "Movie not found"
}
```

/allmovies:

GET http://localhost:4000/allmovies Send

Query Headers 2 Auth Body Tests Pre Run

JSON XML Text Form Form-encode GraphQL Binary

JSON Content Format

Status: 200 OK Size: 1002 Bytes Time: 67 ms

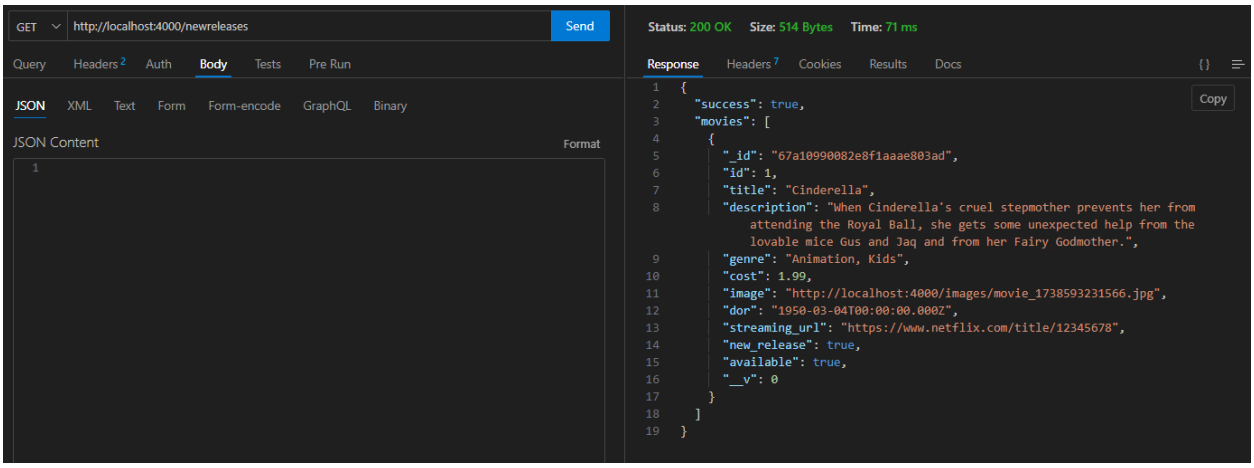
Response Headers 7 Cookies Results Docs

Copy


```
{
  "success": true,
  "movies": [
    {
      "_id": "67a10990082e8f1aaae803ad",
      "id": 1,
      "title": "Cinderella",
      "description": "When Cinderella's cruel stepmother prevents her from attending the Royal Ball, she gets some unexpected help from the lovable mice Gus and Jaq and from her Fairy Godmother.",
      "genre": "Animation, Kids",
      "cost": 1.99,
      "image": "http://localhost:4000/images/movie_1738593231566.jpg",
      "dor": "1950-03-04T00:00:00.000Z",
      "streaming_url": "https://www.netflix.com/title/12345678",
      "new_release": false,
      "available": true,
      "__v": 0
    },
    {
      "_id": "67a10f19aac2c80a33ba5efe",
      "id": 2,
      "title": "Lion King",
      "description": "When Cinderella's cruel stepmother prevents her from attending the Royal Ball, she gets some unexpected help from the lovable mice Gus and Jaq and from her Fairy Godmother.",
      "genre": "Animation, Kids",
      "cost": 1.99,
      "image": "http://localhost:4000/images/movie_1738593231566.jpg",
```


Response Chart

/newreleases



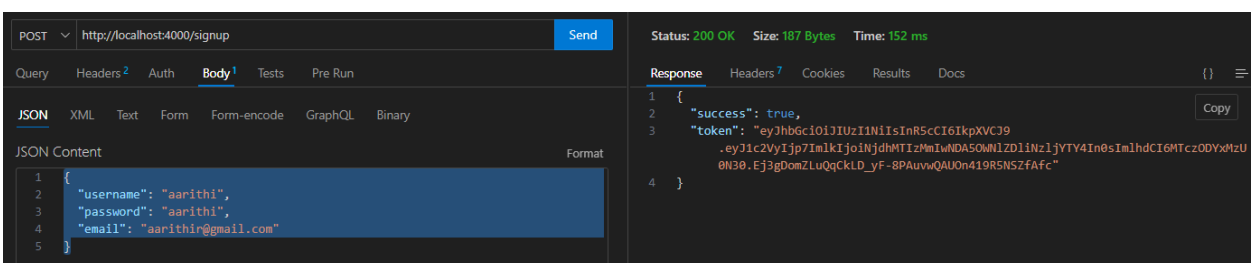
/signup

Request:

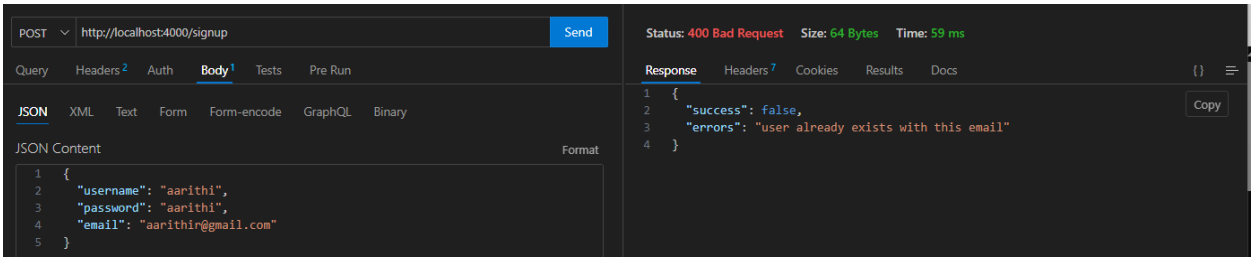
{
"username": "aarithi",
"password": "aarithi",
"email": aarithir@gmail.com
}

Response:

{
"success": true,
"token":
"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VyIjp7ImlkIjoiNjdhMTIzMmIwNDA5OWNlZDliNzljYTY4In0sImlhdCI6MTczODYxMzU0N30.Ej3gDomZLuQqCkLD_yF-8PAuvwQAUOn419R5NSZfAfc"
}



Validation: email should be unique for each user



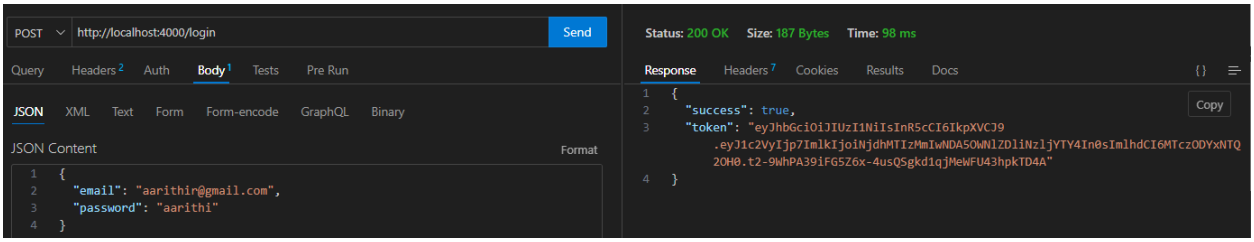
“token” in the above response is the jwt authentication token. The frontend application, when calls endpoints using http POST/GET methods, adds "auth-token" header with the value of “token”. During user login also, auth-token (jwt token) is generated and passed as response to the caller. Frontend application adds this token to http header by name "auth-token". This auth-token is valid until user logs off. Until then every call that frontend UI should validate the presence of this token. If not, UI should default to Login page. Backend application validates for below endpoints to make sure that user is authenticated.

/addtocart
/removefromcart
/getcart

/login

Request:
{
"email": "aarithir@gmail.com",
"password": "aarithi"
}

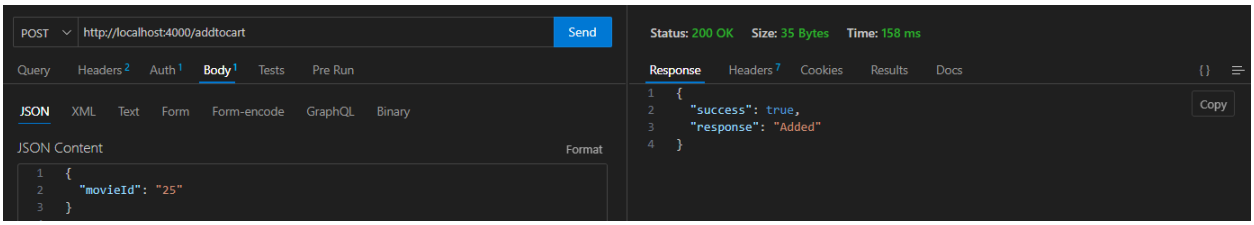
Response:
{
"success": true,
"token":
"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VyIjp7ImlkIjoiNjdhMTIzMmIwNDA5OWNlZDliNzljYTY4In0sImlhdCI6MTczODYxNTQ2OH0.t2-9WhPA39iFG5Z6x-4usQSgkd1qjMeWFU43hpkTD4A"
}



Validation: email should be unique for each user

/addtocart
Request:
{
 "movieId": "25"
}

Response:
{
 "success": true,
 "response": "Added"
}



Note: Upon user signup, user document added to "Users" collection in mongodb. Each user has cartData object initialized with 100 items. The assumption is that there will 100 movies loaded to "Movie" collection with movieId starting from 1 to 100.

When "Add to Cart" is clicked, the frontend UI will make a call to /addtocart endpoint (http://locathost:4000/addtocart) with:

{ "movieId": "<movieID>" }

The frontend UI will know the movieId that user selecting either making the call to /moviebytitle endpoint OR string the movieId of somewhere ….

The movieId corresponds to the item number in User.cartData

Any operation on cart, /addtocart, /removefromcart and /getcart requires auth-token. Frontend will store the auth-token (jwt) created during sign-up or login and pass the same to cart operations by setting this http header.

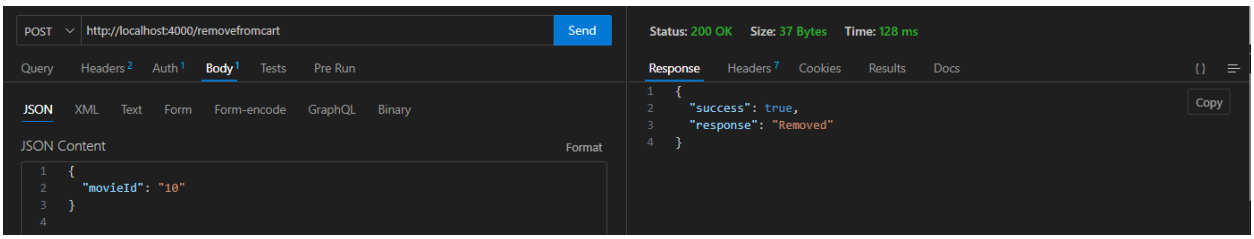
Example:
```
fetch('http://localhost:4000/addtocart', {
  method: 'POST',
  headers: {
    Accept: 'application/form-data',
    'auth-token': `${localStorage.getItem('auth-token')}`,        //assuming jwt token stored in localStorage
    'Content-Type': 'application-json',
  },
  body.JSON.stringify({"movieId": moviedId}),
})
.then
```


}

```
  _id: ObjectId('67a1232b04099ced9b79ca68')
  name : "aarithi"
  email : "aarithir@gmail.com"
  password : "aarithi"
▾ cartData : Object
    0 : 0
    1 : 0
    2 : 0
    3 : 0
    4 : 0
    5 : 0
    6 : 0
    7 : 0
    8 : 0
    9 : 0
    10 : 0
    11 : 0
```

The corresponding itemId (which is same as movieId) in cartData will be incremented by 1.
If there are five movies selected by the user, frontend UI will make call to /addtocart five times, each time changing the movieId.

/removefromcart



/getcart

POST http://localhost:4000/getcartTest Send

Query Headers 2 Auth 1 Body Tests Pre Run

JSON XML Text Form Form-encode GraphQL Binary

JSON Content Format

Status: 200 OK Size: 719 Bytes Time: 61 ms

Response Headers 7 Cookies Results Docs

Copy


```
{
  "success": true,
  "cartData": {
    "0": 0,
    "1": 0,
    "2": 0,
    "3": 0,
    "4": 0,
    "5": 0,
    "6": 0,
    "7": 0,
    "8": 0,
    "9": 0,
    "10": 1,
    "11": 0,
    "12": 0,
    "13": 0,
```


....

....


```
    "94": 0,
    "95": 0,
    "96": 0,
    "97": 0,
    "98": 0,
    "99": 0
  }
}
```


---

npx create-react-app .

npm install react-router-dom

npm start

backend:

mongodb:

https://www.mongodb.com/atlas

techstack/techstack

mongodb+srv://techstack:<db_password>@cluster0.ikop1.mongodb.net/

login: https://account.mongodb.com/account/login?signedOut=true

nodejs:

https://nodejs.org/en/download

vscode terminal:

npm install

node index.js (start Express server)

http://localhost:4000

frontend:

vscode terminal

npm install

npm start (start frontend)

http://localhost:3000 (app url)

backend:

Express.js (Express) to create API and json web token (jwt) for user authentication.

Express is a web application framework that provides a robust set of features for web and mobile applications. It simplifies the process of building server-side applications.

Mongodb Atlas database

to store the image -> Multer is a Node.js middleware for handling multipart/form-data, which is primarily used for uploading files. It is an essential part of any web application that needs to handle file uploads, especially when you're working with forms that include file inputs.

vscode terminal

cd backend

npm init

package name: backend

Entry point: index.js

Install Express Server:

npm install express

install jwt package for user auth:

npm install jsonwebtoken

install mongodb package

npm install mangoose

install multer for storing images

npm install multer

install cors to grant permission to the application to access the backend

npm install cors

(CORS stands for Cross-Origin Resource Sharing. It is a security feature implemented by web browsers to control how web pages can request resources from different origins (domains) than their own)

install ThunderClient in vscode

admin:

npm vite@latest .

vite is next gen frontend development tool and it requires Node.js v18 and above

select "React" framework

slect "javascript"

npm install

npm install react-router-dom

to start => npm run dev