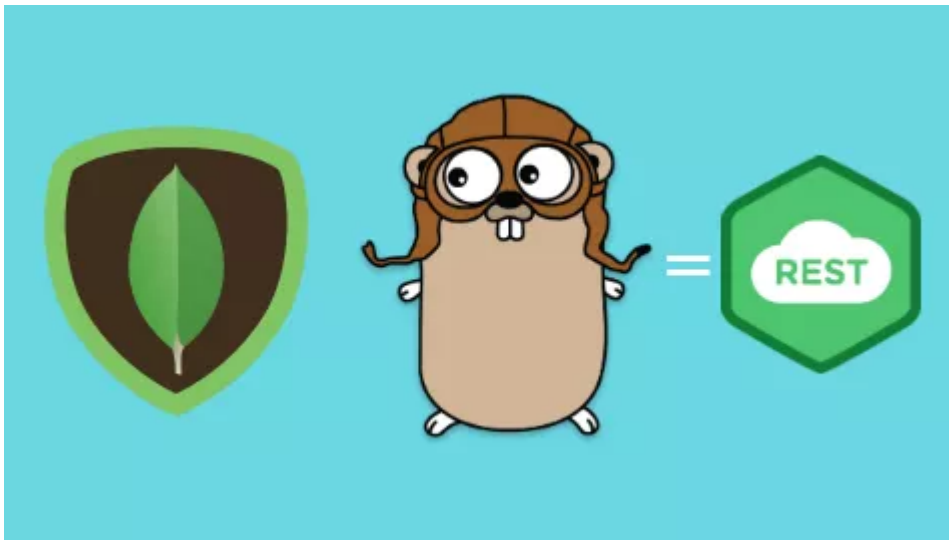
Mohamed Labouardy

Software Engineer/DevOps Engineer, 3x AWS Certified Interested in Serverless, Containers, Go, Distributed Systems & NLP.



Build RESTful API in Go and MongoDB

In this tutorial I will illustrate how you can build your own **RESTful API** in **Go** and **MongoDB**. All the code used in this demo can be found on my Github. 😊

1 – API Specification

The **REST API service** will expose endpoints to manage a store of movies. The operations that our endpoints will allow are:

GET	/movies	Get list of movies
GET	/movies/:id	Find a movie by its ID
POST	/movies	Create a new movie
PUT	/movies	Update an existing movie
DELETE	/movies	Delete an existing movie

2 – Fetching Dependencies

Before we begin, we need to grap the packages we need to setup the API:

```
go get github.com/BurntSushi/toml gopkg.in/mgo.v2 github.com/gorilla/mux
```

- **toml** : Parse the configuration file (MongoDB server & credentials)
- **mux** : Request router and dispatcher for matching incoming requests to their respective handler
- **mgo** : MongoDB driver

3 – API structure

Once the dependencies are installed, we create a file called "**app.go**", with the following content:

```
package main

import (
    "fmt"
    "log"
    "net/http"

    "github.com/gorilla/mux"
)

func AllMoviesEndPoint(w http.ResponseWriter, r *http.Request) {
    fmt.Fprintln(w, "not implemented yet !")
}

func FindMovieEndpoint(w http.ResponseWriter, r *http.Request) {
    fmt.Fprintln(w, "not implemented yet !")
}

func CreateMovieEndPoint(w http.ResponseWriter, r *http.Request) {
    fmt.Fprintln(w, "not implemented yet !")
}

func UpdateMovieEndPoint(w http.ResponseWriter, r *http.Request) {
    fmt.Fprintln(w, "not implemented yet !")
}

func DeleteMovieEndPoint(w http.ResponseWriter, r *http.Request) {
    fmt.Fprintln(w, "not implemented yet !")
}

func main() {
    r := mux.NewRouter()
    r.HandleFunc("/movies", AllMoviesEndPoint).Methods("GET")
    r.HandleFunc("/movies", CreateMovieEndPoint).Methods("POST")
    r.HandleFunc("/movies", UpdateMovieEndPoint).Methods("PUT")
    r.HandleFunc("/movies", DeleteMovieEndPoint).Methods("DELETE")
    r.HandleFunc("/movies/{id}", FindMovieEndpoint).Methods("GET")
    if err := http.ListenAndServe(":3000", r); err != nil {
        log.Fatal(err)
    }
```

```
41 }
```

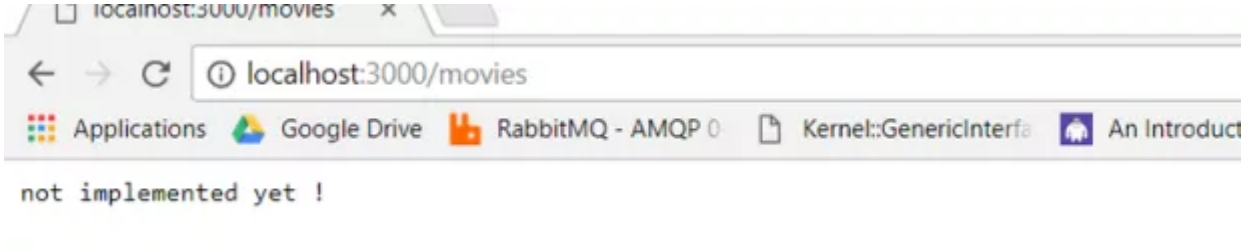
The code above creates a controller for each endpoint, then expose an **HTTP server** on port **3000**.

Note: We are using **GET**, **POST**, **PUT**, and **DELETE** where appropriate. We are also defining parameters that can be passed in

To run the server in local, type the following command:

```
1 go run app.go
```

If you point your browser to **http://localhost:3000/movies**, you should see:



4 – Model

Now that we have a minimal application, it's time to create a basic **Movie** model. In **Go**, we use **struct** keyword to create a model:

```
1 type Movie struct {
2     ID          bson.ObjectId `bson:"_id" json:"id"`
3     Name        string        `bson:"name" json:"name"`
4     CoverImage  string        `bson:"cover_image" json:"cover_image"`
5     Description string        `bson:"description" json:"description"`
6 }
```

Next, we will create the **Data Access Object** to manage database operations.

5 – Data Access Object

5.1 – Establish Connection

```
1  package dao
2
3  import (
4      "log"
5
6      "github.com/mlabouardy/movies-restapi/models"
7      mgo "gopkg.in/mgo.v2"
8      "gopkg.in/mgo.v2/bson"
9  )
10
11 type MoviesDAO struct {
```

```
    Server   string
    Database string
}

var db *mgo.Database

const (
    COLLECTION = "movies"
)

func (m *MoviesDAO) Connect() {
    session, err := mgo.Dial(m.Server)
    if err != nil {
        log.Fatal(err)
    }
    db = session.DB(m.Database)
}
```

The **connect()** method as its name implies, establish a connection to **MongoDB database.**

5.2 – Database Queries

The implementation is relatively straighforward and just includes issuing right method using **db.C(COLLECTION)** object and returning the results. These methods can be implemented as follows:

```
func (m *MoviesDAO) FindAll() ([]Movie, error) {
    var movies []Movie
    err := db.C(COLLECTION).Find(bson.M{}).All(&movies)
    return movies, err
}

func (m *MoviesDAO) FindById(id string) (Movie, error) {
    var movie Movie
    err := db.C(COLLECTION).FindId(bson.ObjectIdHex(id)).One(&movie)
    return movie, err
}

func (m *MoviesDAO) Insert(movie Movie) error {
    err := db.C(COLLECTION).Insert(&movie)
    return err
}

func (m *MoviesDAO) Delete(movie Movie) error {
    err := db.C(COLLECTION).Remove(&movie)
    return err
}

func (m *MoviesDAO) Update(movie Movie) error {
    err := db.C(COLLECTION).UpdateId(movie.ID, &movie)
    return err
}
```

6 – Setup API Endpoints

6.1 – Create a Movie

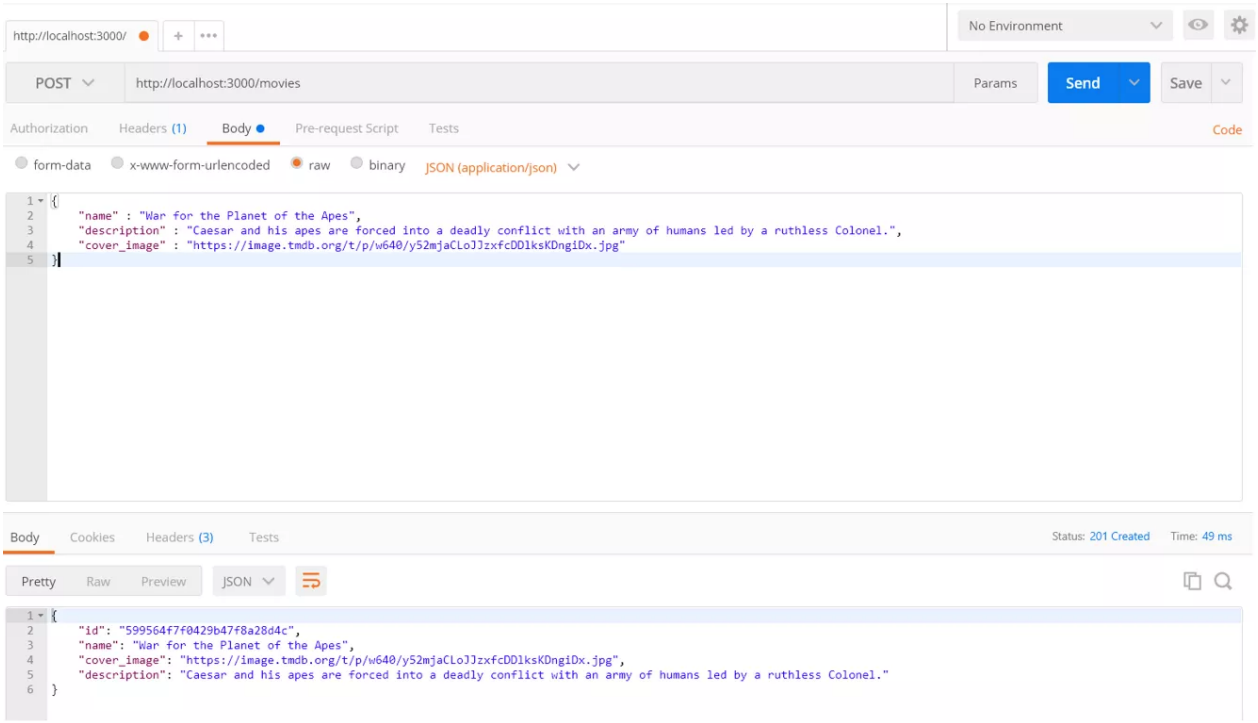
Update the **CreateMovieEndpoint** method as follows:

```
func CreateMovieEndPoint(w http.ResponseWriter, r *http.Request) {
    defer r.Body.Close()
    var movie Movie
    if err := json.NewDecoder(r.Body).Decode(&movie); err != nil {
        respondWithError(w, http.StatusBadRequest, "Invalid request payload")
        return
    }
    movie.ID = bson.NewObjectId()
    if err := dao.Insert(movie); err != nil {
        respondWithError(w, http.StatusInternalServerError, err.Error())
        return
    }
    respondWithJson(w, http.StatusCreated, movie)
}
```

It decodes the request body into a **movie** object, assign it an **ID**, and uses the **DAO Insert** method to create a **movie** in database.

Let's test it out:

With **Postman**:



With **cURL**

```
curl -sSX POST -d '{"name":"dunkirk","cover_image":"https://image.tmdb.org/t/p/w640/
```

6.2 – List of Movies

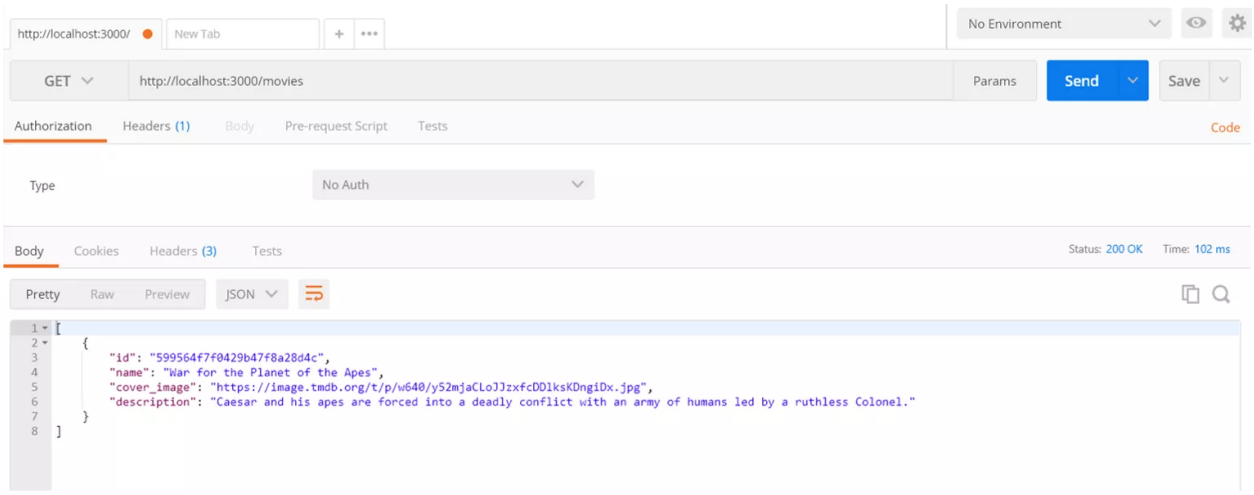
The code below is self explanatory:

```
func AllMoviesEndPoint(w http.ResponseWriter, r *http.Request) {
    movies, err := dao.FindAll()
    if err != nil {
        respondWithError(w, http.StatusInternalServerError, err.Error())
        return
    }
    respondWithJson(w, http.StatusOK, movies)
}
```

It uses **FindAll** method of **DAO** to fetch list of movies from database.

Let's test it out:

With **Postman**:



With **cURL**:

```
curl -sSX GET http://localhost:3000/movies | jq '.'
```

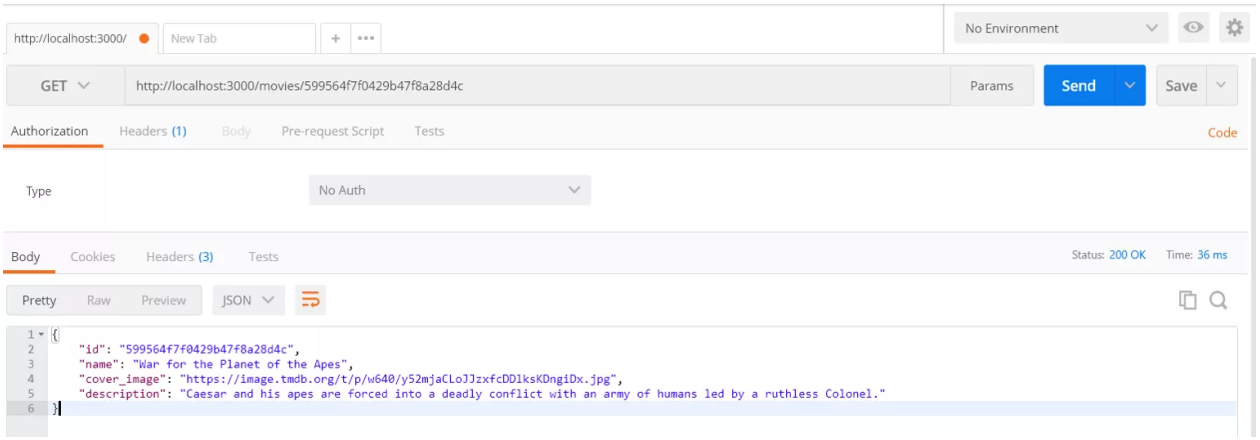
6.3 – Find a Movie

We will use the mux library to get the parameters that the users passed in with the request:

```
func FindMovieEndpoint(w http.ResponseWriter, r *http.Request) {
    params := mux.Vars(r)
    movie, err := dao.FindById(params["id"])
    if err != nil {
        respondWithError(w, http.StatusBadRequest, "Invalid Movie ID")
        return
    }
    respondWithJson(w, http.StatusOK, movie)
}
```

Let's test it out:

With **Postman**:



With **cURL**:

```
curl -sSX GET http://localhost:3000/movies/599570faf0429b4494cfa5d4 | jq '.'
```

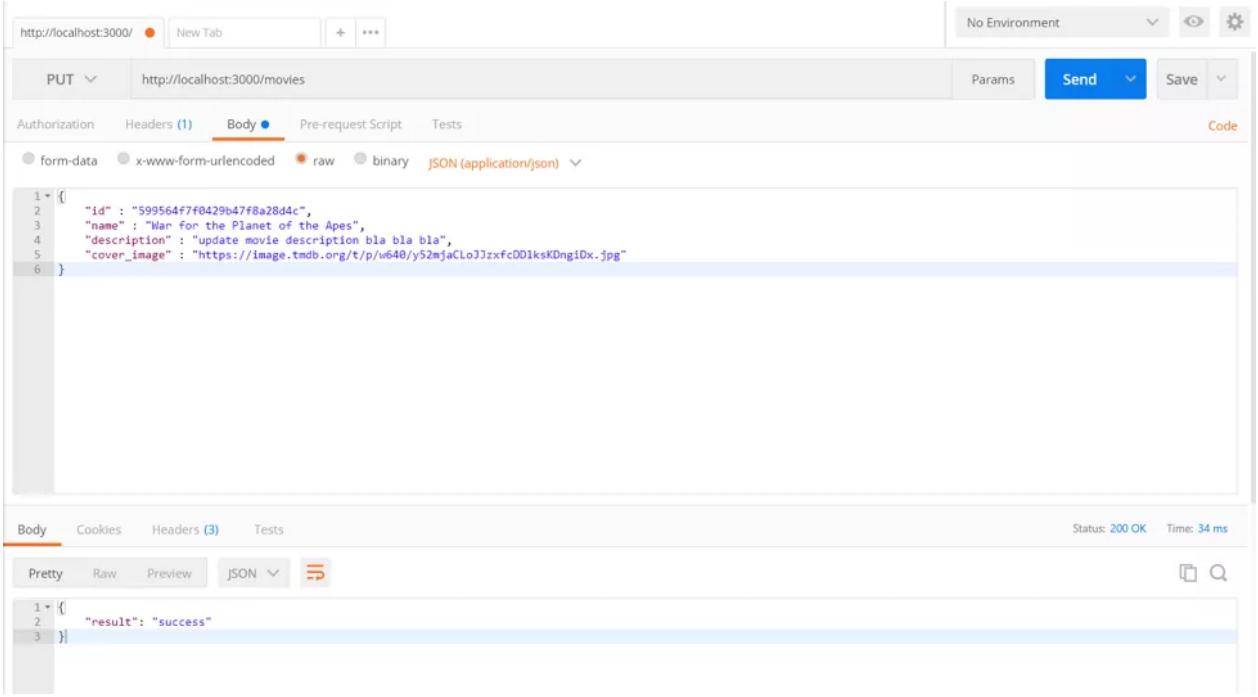
6.4 – Update an existing Movie

Update the **UpdateMovieEndPoint** method as follows:

```
func UpdateMovieEndPoint(w http.ResponseWriter, r *http.Request) {
    defer r.Body.Close()
    var movie Movie
    if err := json.NewDecoder(r.Body).Decode(&movie); err != nil {
        respondWithError(w, http.StatusBadRequest, "Invalid request payload")
        return
    }
    if err := dao.Update(movie); err != nil {
        respondWithError(w, http.StatusInternalServerError, err.Error())
        return
    }
    respondWithJson(w, http.StatusOK, map[string]string{"result": "success"})
}
```

Let's test it out:

With **Postman**:



With **cURL**:

```
curl -sSX PUT -d '{"name":"dunkirk","cover_image":"https://image.tmdb.org/t/p/w640/c
```

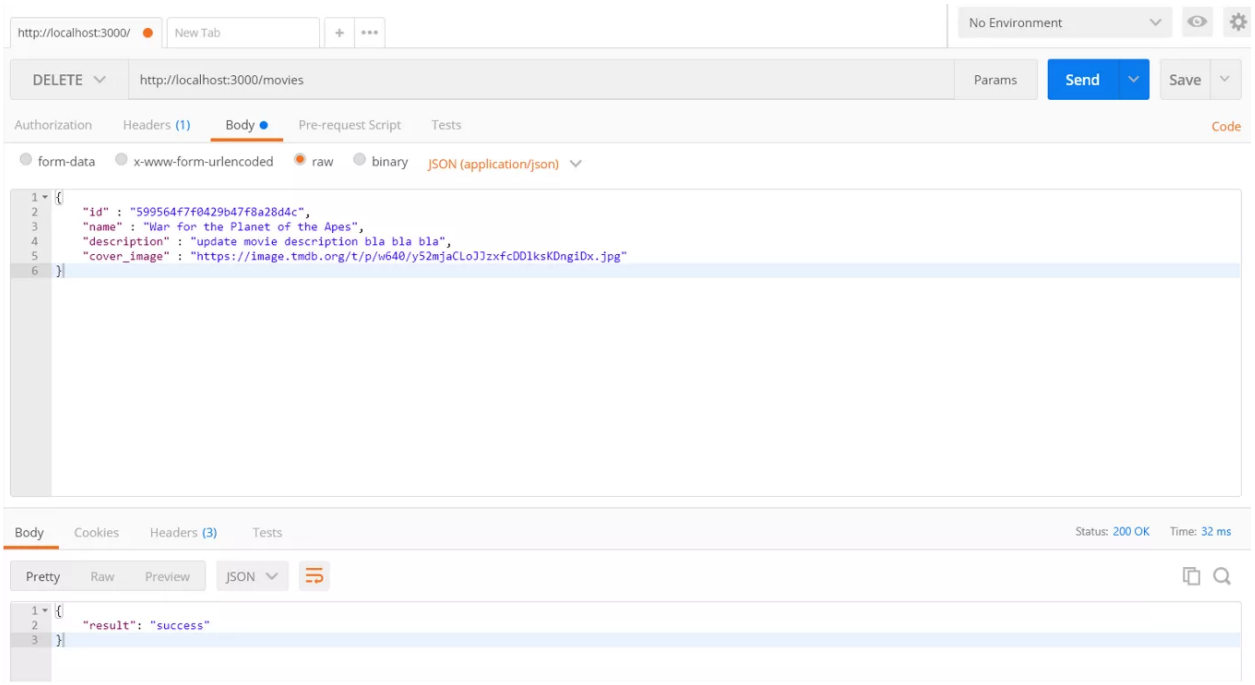
6.5 – Delete an existing Movie

Update the **DeleteMovieEndPoint** method as follows:

```
func DeleteMovieEndPoint(w http.ResponseWriter, r *http.Request) {
    defer r.Body.Close()
    var movie Movie
    if err := json.NewDecoder(r.Body).Decode(&movie); err != nil {
        respondWithError(w, http.StatusBadRequest, "Invalid request payload")
        return
    }
    if err := dao.Delete(movie); err != nil {
        respondWithError(w, http.StatusInternalServerError, err.Error())
        return
    }
    respondWithJson(w, http.StatusOK, map[string]string{"result": "success"})
}
```

Let's test it out:

With **Postman**:



With **cURL**:

```
curl -sSX DELETE -d '{"name":"dunkirk","cover_image":"https://image.tmdb.org/t/p/w64
```

7 – **Taking this further**

On my upcoming posts, I will show you how :

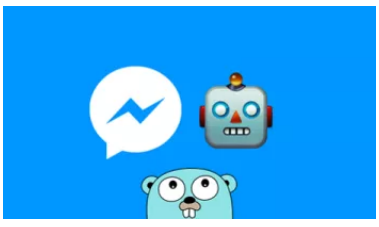
- Write **Unit Tests** in **Go** for each Endpoint
- Build a UI in **Angular 4**
- Setup a **CI/CD** with **CircleCI**
- Deploy the stack on **AWS** and much more ...

So stay tuned 😇

SHARE THIS:

RELATED

Create 9Gag Android Application



Build a Facebook Messenger bot with Go and Messenger

Serverless Golang API with AWS Lambda

In "Android" **API** In "Angular"
In "Bot"

Comments

10 comments

9 Comments Sort by Oldest

Add a comment...

Martin Alejandro Pérez Güendulain

Thanks, you explain very well

Like · Reply · 1 · 1y

Med Labouardy

I'm glad you like it

Like · Reply · 1y

Евгений Шевченко

Thank you for this article !

Like · Reply · 1 · 1y

Kaido Kariste

Good job, simple hands on API

Like · Reply · 1 · 1y

Sohail Shahul Hameed

Thanks for sharing the knowledge.
When i run this api below is the error part :
dao/movies_dao.go:5:2: imported and not used: "github.com/Projects/movies-restapi-master/models"
dao/movies_dao.go:31:34: undefined: Movie
dao/movies_dao.go:32:15: undefined: Movie
dao/movies_dao.go:38:42: undefined: Movie
dao/movies_dao.go:39:12: undefined: Movie

Like · Reply · 47w

Auttapon Jame Chutiphornphanit

i have one question we should close connection mongodb or not when query complete ?

Like · Reply · 45w

Load 4 more comments

Facebook Comments Plugin

August 16, 2017 Mohamed Labouardy AWS, AWS S3, Go, MongoDB

Designed by Mohamed Labouardy