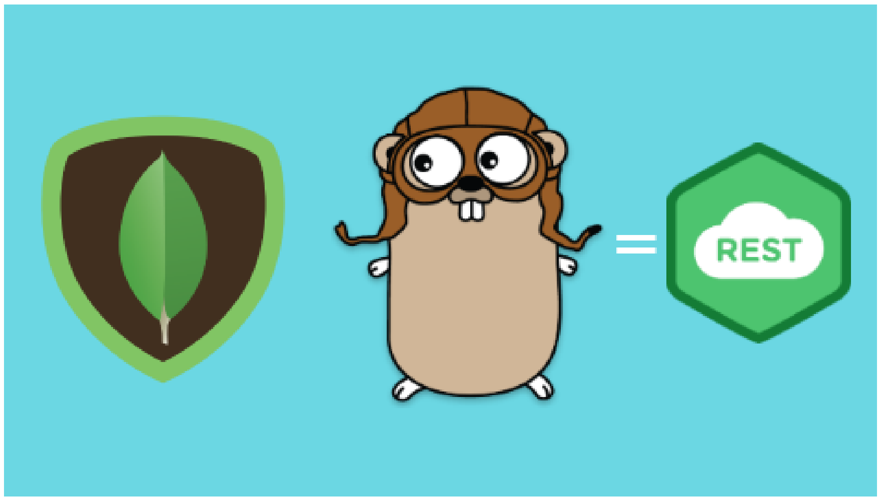
Mohamed Labouardy Follow
Software Engineer/DevOps Engineer 5x #AWS Certified - PSM 1 Certified — Author #Serverless #Containers #Alexa #Go #NLP #Android — Blog: http://labouardy.com
Nov 11, 2017 · 4 min read

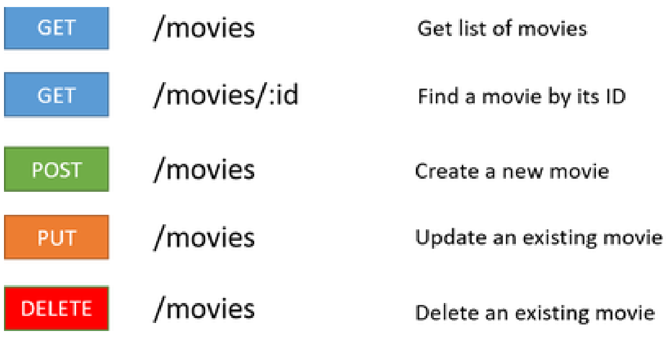
Build RESTful API in Go and MongoDB



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 get 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:

```go
package main

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

	"github.com/gorilla/mux"
)

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

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

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

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

func DeleteMovieEndPoint(w http.ResponseWriter, r *http.Re
```

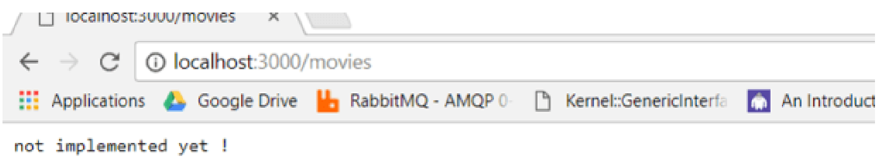
The code above creates a controller for each endpoint, then expose an **HTTP server** on port **3000**.

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:

> *go run app.go*

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



```
not implemented yet !
```

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:

```go
type Movie struct {
	ID          bson.ObjectId `bson:"_id" json:"id"`
	Name        string        `bson:"name" json:"name"`
	CoverImage  string        `bson:"cover_image" json:"
	Description string        `bson:"description" json:"
```

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

5—Data Access Object

5.1—Establish Connection

```go
package dao

import (
        "log"

        "github.com/mlabouardy/movies-restapi/models"
        mgo "gopkg.in/mgo.v2"
        "gopkg.in/mgo.v2/bson"
)

type MoviesDAO struct {
        Server   string
        Database string
}

var db *mgo.Database

const (
        COLLECTION = "movies"
```

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:

```go
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)
	return movie, err
}

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

```

6—Setup API Endpoints

6.1—Create a Movie

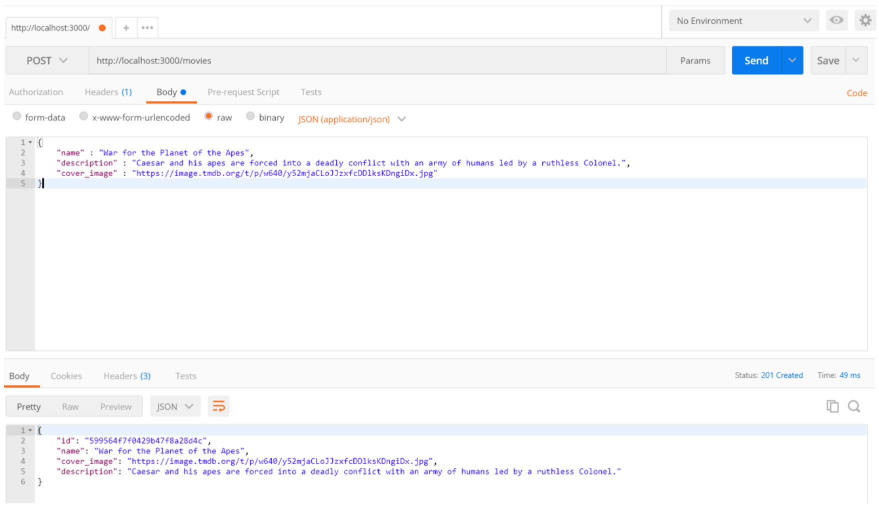
Update the **CreateMovieEndpoint** method as follows:

```go
func CreateMovieEndPoint(w http.ResponseWriter, r *http.Re
	defer r.Body.Close()
	var movie Movie
	if err := json.NewDecoder(r.Body).Decode(&movie); e
		respondWithError(w, http.StatusBadRequest,
		return
	}
	movie.ID = bson.NewObjectId()
	if err := dao.Insert(movie); err != nil {
		respondWithError(w, http.StatusInternalServ
```

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**:



POST http://localhost:3000/movies

JSON (application/json)

```
{
    "name" : "War for the Planet of the Apes",
    "description" : "Caesar and his apes are forced into a deadly conflict with an army of humans led by a ruthless Colonel.",
    "cover_image" : "https://image.tmdb.org/t/p/w640/y52mjaCLoJJzxfcDDlksKDngiDx.jpg"
}
```

Status: 201 Created Time: 49 ms

```
{
    "id": "599564f7f0429b47f8a28d4c",
    "name": "War for the Planet of the Apes",
    "cover_image": "https://image.tmdb.org/t/p/w640/y52mjaCLoJJzxfcDDlksKDngiDx.jpg",
    "description": "Caesar and his apes are forced into a deadly conflict with an army of humans led by a ruthless Colonel."
}
```

With **cURL**

> *curl -sSX POST -d '{"name":"dunkirk","cover_image":"https://image.tmdb.org/t/p/w640/cUqEgoP6kj8ykfNjJx3Tl5zHCcN.jpg", "description":"world war 2 movie"}' http://localhost:3000/movies | jq '.'*

6.2—List of Movies

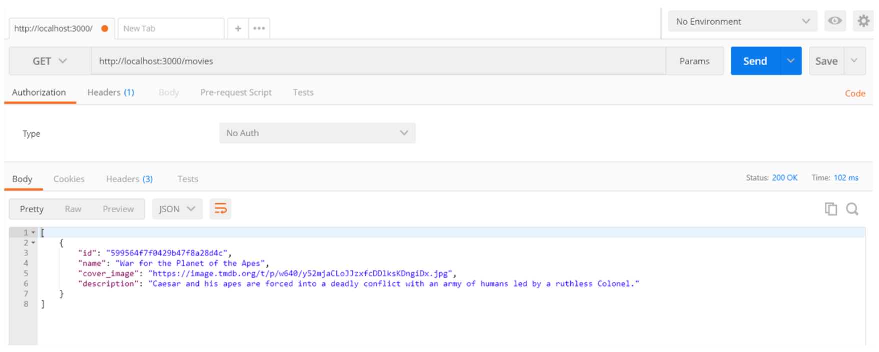
The code below is self explanatory:

```
func AllMoviesEndPoint(w http.ResponseWriter, r *http.Reque
	movies, err := dao.FindAll()
	if err != nil {
		respondWithError(w, http.StatusInternalServe
		return
	}
```

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

Let's test it out:

With **Postman**:



http://localhost:3000/movies

Type: No Auth

Status: 200 OK Time: 102 ms

```
[
    {
        "id": "599564f7f0429b47f8a28d4c",
        "name": "War for the Planet of the Apes",
        "cover_image": "https://image.tmdb.org/t/p/w640/y52mjaCLoJJzxfcDDlksKDngiDx.jpg",
        "description": "Caesar and his apes are forced into a deadly conflict with an army of humans led by a ruthless Colonel."
    }
]
```

With **cURL**:

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

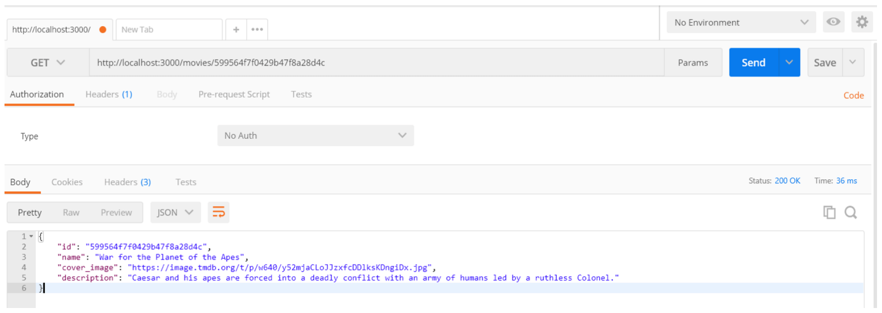
6.3—Find a Movie

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.Reque
	params := mux.Vars(r)
	movie, err := dao.FindById(params["id"])
	if err != nil {
		respondWithError(w, http.StatusBadRequest, "
		return
	}
```

Let's test it out:

With **Postman**:



http://localhost:3000/movies/599564f7f0429b47f8a28d4c

Type: No Auth

Status: 200 OK Time: 36 ms

```
{
    "id": "599564f7f0429b47f8a28d4c",
    "name": "War for the Planet of the Apes",
    "cover_image": "https://image.tmdb.org/t/p/w640/y52mjaCLoJJzxfcDDlksKDngiDx.jpg",
    "description": "Caesar and his apes are forced into a deadly conflict with an army of humans led by a ruthless Colonel."
}
```

With **cURL**:

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

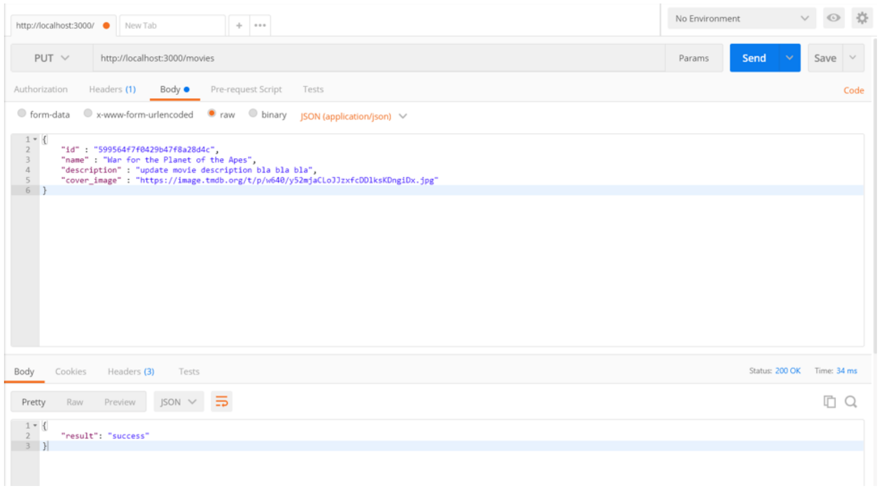
6.4—Update an existing Movie

Update the **UpdateMovieEndPoint** method as follows:

```
func UpdateMovieEndPoint(w http.ResponseWriter, r *http.Re
	defer r.Body.Close()
	var movie Movie
	if err := json.NewDecoder(r.Body).Decode(&movie); e
		respondWithError(w, http.StatusBadRequest,
		return
	}
	if err := dao.Update(movie); err != nil {
		respondWithError(w, http.StatusInternalServ
```

Let's test it out:

With **Postman**:



PUT http://localhost:3000/movies

Body: raw, JSON (application/json)

```
{
    "id" : "599564f7f0429b47f8a28d4c",
    "name" : "War for the Planet of the Apes",
    "description" : "update movie description bla bla bla",
    "cover_image" : "https://image.tmdb.org/t/p/w640/y52mjaCLoJJzxfcDDlksKDngiDx.jpg"
}
```

Status: 200 OK Time: 34 ms

```
{
    "result": "success"
}
```

With **cURL**:

> *curl -sSX PUT -d '{"name":"dunkirk","cover_image":"https://image.tmdb.org/t/p/w640/cUqEgoP6kj8ykfNjJx3Tl5zHCcN.jpg", "description":"world war 2 movie"}' http://localhost:3000/movies | jq '.'*

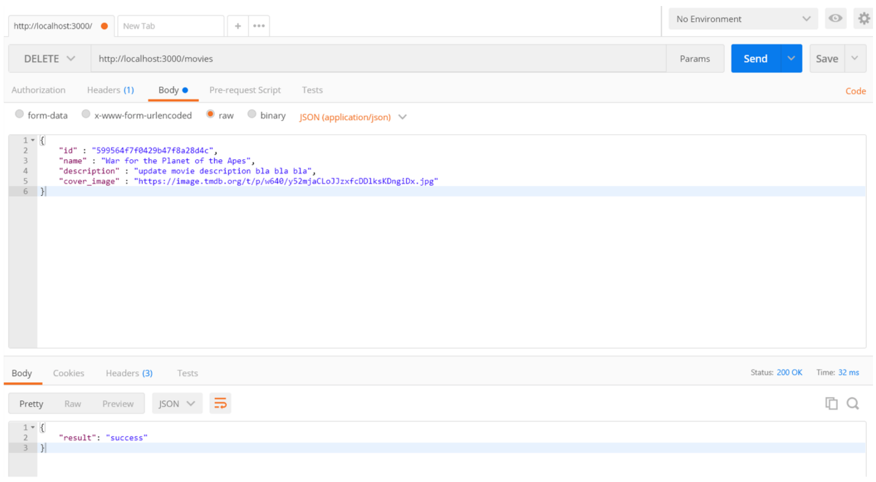
6.5—Delete an existing Movie

Update the **DeleteMovieEndPoint** method as follows:

```
func DeleteMovieEndPoint(w http.ResponseWriter, r *http.Re
	defer r.Body.Close()
	var movie Movie
	if err := json.NewDecoder(r.Body).Decode(&movie); e
		respondWithError(w, http.StatusBadRequest,
		return
	}
	if err := dao.Delete(movie); err != nil {
		respondWithError(w, http.StatusInternalServ
```

Let's test it out:

With **Postman**:


DELETE http://localhost:3000/movies

```
{
    "id" : "599564f7f0429b47f8a28d4c",
    "name" : "War for the Planet of the Apes",
    "description" : "update movie description bla bla bla",
    "cover_image" : "https://image.tmdb.org/t/p/w640/y52mjaCLoJJzxfcDDlksKDngiDx.jpg"
}
```

Status: 200 OK Time: 32 ms

```
{
    "result": "success"
}
```

With **cURL**:

> *curl -sSX DELETE -d '{"name":"dunkirk","cover_image":"https://image.tmdb.org/t/p/w640/cUqEgoP6kj8ykfNjJx3Tl5zHCcN.jpg", "description":"world war 2 movie"}' http://localhost:3000/movies | jq '.'*

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 !