

You have **2** free member-only stories left this month. [Sign up for Medium and get an extra one](#)

# Build a Simple To-Do List App in Golang

One of the most instructive ways to wrap your head around a new language



Mohamad Fadhil

Follow

Feb 3 · 7 min read ★

This blog is originally published on [my personal blog](#).



Illustration by [Mohamed Hassan](#) on [Pixabay](#).



## Why Golang?

I've wanted to learn Golang for a long time. According to a Stack Overflow survey, Golang was the third-most wanted programming language in the year 2019. Coming from a Python background, learning a new programming language like Golang gives me a unique perspective on how to write code.

I was inspired by a helpful [tutorial](#) by Keiran Scot. The following tutorial is heavily influenced by Keiran's tutorial but is made to use MySQL instead of Mongo as part of my exercise to learn Golang.

Prsonally, I believe Golang is the new Java. Many open-source projects (e.g., Jaeger, Kubernetes, Docker, InfluxDB, Serf, etc.) are now written in Golang. In contrast, the older projects (e.g., Apache Kafka, Apache Hadoop, Apache Spark, etc.) are written in Java. I wish I could participate and contribute to OSS Golang projects in the future.

There are many advantages of Golang compared to other languages, including:

- **Golang is blazing fast**

[See it yourself.](#)

- **Golang is built for concurrency**

Golang comes with goroutines, which were made to tackle this problem. I was mesmerized with this feature when I first learned about Golang. Unfortunately, we won't be seeing how goroutines work yet in this tutorial.

- **Golang is a statically typed language**

Coming from a Python background, I found this feature in Golang a bit irritating at first. However, writing in a statically typed programming language means you'll introduce fewer bugs because you'll make lesser mistakes on interpreting and passing the data.

- **Go compiles the program into a single binary**

When you're building an application in Go, you can always run your app anywhere. You don't have to worry about its dependencies, because all of them are bundled together.

- **Golang is an opinionated language**

There's a rigorous way to write a Golang code. This means there's less variation on how people should write code in Golang, which eventually implies code consistency, readability, and maintainability. For a big and complex project, this is very helpful.

- You can read more about its advantages [here](#).

## Prerequisites

- [Golang](#) and [Docker](#) installed on your system
- A basic knowledge of Golang and JQuery

I've prepared a front-end part for this app, so you don't have to worry about this and can focus on writing the Golang code. All you need is to clone the repository in Step 7.

## What we'll build

We'll build a classic and simple Golang API server that connects to a front-end page. Our Golang API server will use:

- MySQL as our database
- GORM as an ORM to interact with our database
- Request router using `gorilla/mux`
- Logrus for logging

## How our entire app will work



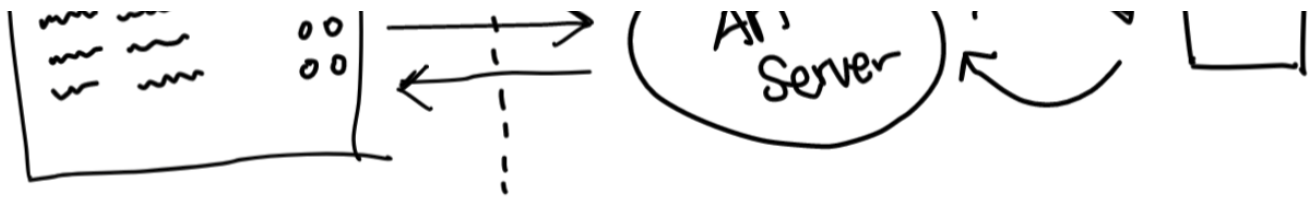


Diagram of how our app will work

We'll be building what's on the right side of the diagram. The specifications for our API Server are:

- It listens to port 8000 on the `localhost`
- It has five endpoints: `healthz`, `createItem`, `getCompletedItems`, `getIncompleteItems`, `updateItem`, and `deleteItem`
- The `TodoItem` model consists of `Id`, `Description`, and `Completed` status attributes

The web front end I've prepared will execute AJAX requests to `localhost:8000` for all operations. Check this JS script to see how it works first. The initial code was made by [themaxsandelin](#).

## Let's Begin

### 1. Bootstrap a project

First, let's create a new directory and install `gorilla/mux` and `logrus` packages in the directory.

```
$ mkdir todo-list-mysql-go
$ cd todo-list-mysql-go
$ go get -u github.com/gorilla/mux
$ go get -u github.com/sirupsen/logrus
```

Now create a `todo-list.go` file with the following content:

```
1 package main
```

```

2
3 import (
4     "io"
5     "net/http"
6     "github.com/gorilla/mux"
7     log "github.com/sirupsen/logrus"
8 )
9
10 func Healthz(w http.ResponseWriter, r *http.Request) {
11     log.Info("API Health is OK")
12     w.Header().Set("Content-Type", "application/json")
13     io.WriteString(w, `{"alive": true}`)
14 }
15
16 func init() {
17     log.SetFormatter(&log.TextFormatter{})
18     log.SetReportCaller(true)
19 }
20
21 func main() {
22     log.Info("Starting Todolist API server")
23     router := mux.NewRouter()
24     router.HandleFunc("/healthz", Healthz).Methods("GET")
25     http.ListenAndServe(":8000", router)
26 }

```

todolist.go hosted with ❤ by GitHub

[view raw](#)

In lines 3-8, we're importing all the necessary packages we get in our previous `go get -u` commands. Beware that you'll get a compilation error when importing any package in Go code without actually using it.

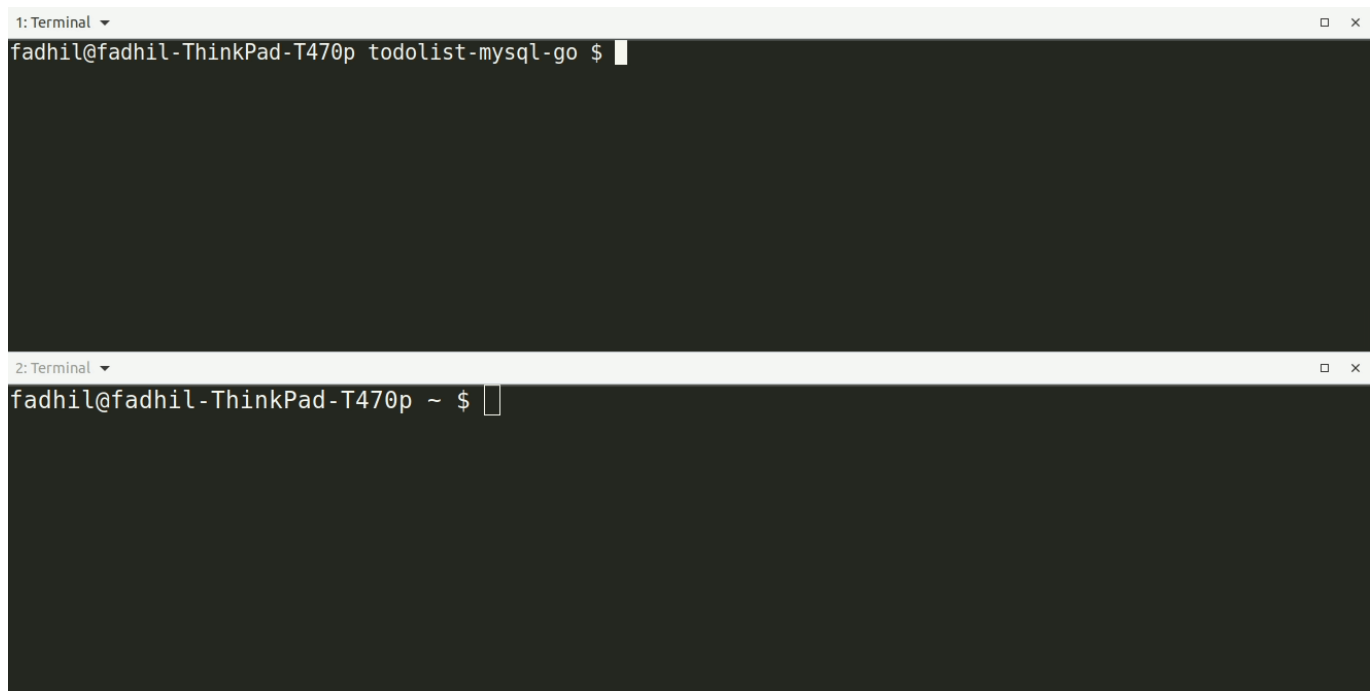
In lines 10-14, we created a `Healthz` function that'll respond `{"alive": true}` to the client every time it's invoked.

In lines 16-19, we set our `init` function to set up our `logrus` logger settings. In Golang, `init()` will be executed when the program first begins. You can read more [here](#).

In lines 21-26, we're initializing our `gorilla/mux` HTTP router with a walrus operator. We route `/healthz` HTTP GET requests to the `Health()` function. The router will listen to port 8000.

It's important to set the `Content-Type: application/json` so the client software understands the response.

Simple enough, right? Let's see what we've got so far.



```
1: Terminal
fadhil@fadhil-ThinkPad-T470p todolist-mysql-go $

2: Terminal
fadhil@fadhil-ThinkPad-T470p ~ $
```

Great! Our first endpoint is now live.

## 2. Connect to the MySQL database

Before connecting our Go app to the MySQL database, first we'll need to have our MySQL server up. We'll be using Docker, in this case, to launch a MySQL container and expose the port 3306 to the `localhost`. We'll create a `todolist` database in the MySQL container.

```
$ docker run -d -p 3306:3306 --name mysql -e
MYSQL_ROOT_PASSWORD=root mysql
$ docker exec -it mysql mysql -uroot -proot -e 'CREATE DATABASE
todolist'
```

Back to our Go app, we'll need to install the GORM libraries and dialects.

```
$ go get -u github.com/jinzhu/gorm
$ go get -u github.com/go-sql-driver/mysql
```

```
$ go get -u github.com/jinzhu/gorm/dialects/mysql
```

Add these lines in our `todolist.go` code:

```
1  package main
2
3  import (
4      "io"
5      "net/http"
6      "github.com/gorilla/mux"
7      log "github.com/sirupsen/logrus"
8  +   _ "github.com/go-sql-driver/mysql"
9  +   "github.com/jinzhu/gorm"
10 +   _ "github.com/jinzhu/gorm/dialects/mysql"
11 )
12
13 +var db, _ = gorm.Open("mysql", "root:root@/todolist?charset=utf8&parseTime=True&loc=Loc
14
15 ...
16
17 func main() {
18 +   defer db.Close()
19 +
20     log.Info("Starting Todolist API server")
21     router := mux.NewRouter()
22     router.HandleFunc("/healthz", Healthz).Methods("GET")
23     http.ListenAndServe(":8000", router)
24 }
```

`todolist-2.go` hosted with ❤️ by GitHub

[view raw](#)

Add our new imports into the import list in lines 8-10. In Golang, if you don't intend to use the variable, you have to name it `_`. You can read more [here](#).

Next, we'll initialize our MySQL connection to our database using GORM. The following lines are to check if there's an error connecting to the database. If there is, log it.

`defer db.Close()` means we'll close our database connection when our `main()` function is returned.

### 3. Auto migrate the TodoItem ORM model

GORM comes with a neat database automigration feature, where it'll create a database table based on your struct definition in the code.

Our `TodoItem` struct will consist of:

- `Id int` : This will be our `primary_key`
- `Description string` : This is what we'll display in our front-end UI
- `Completed bool` : This is to determine if the `TodoItem` is done or not

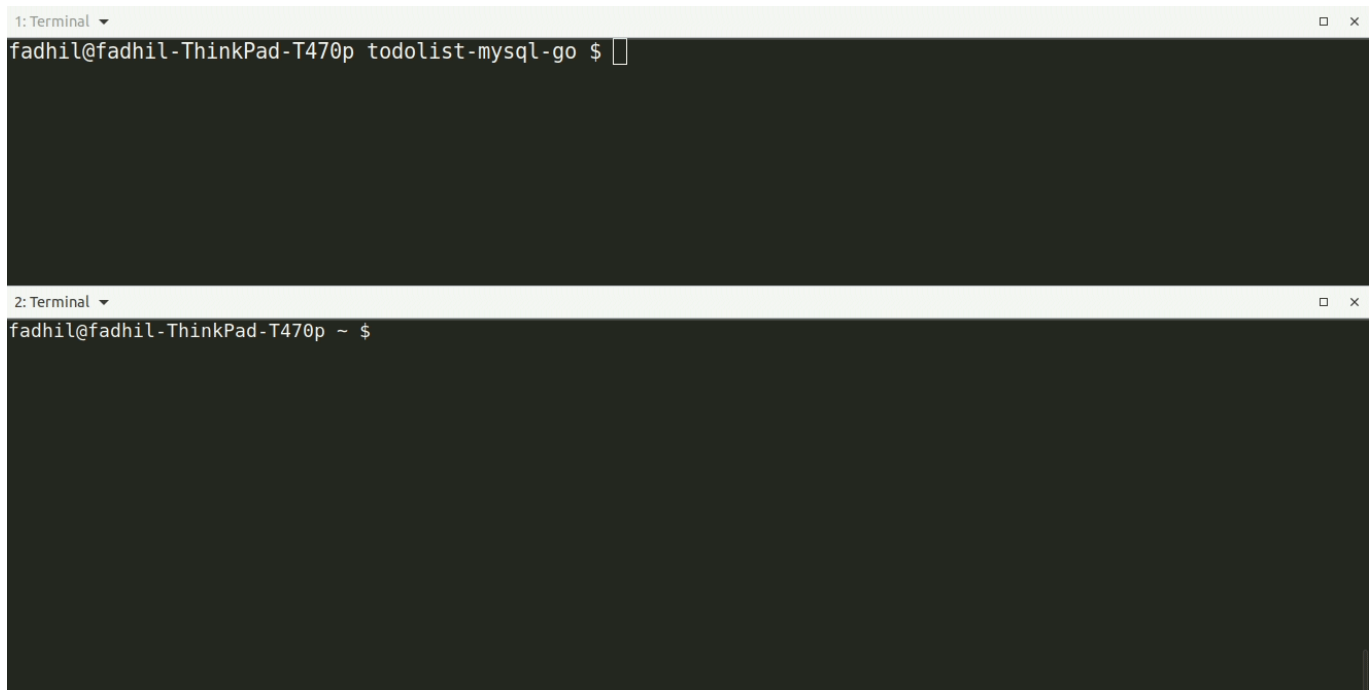
Add these lines to your Go code:

```
1  package main
2
3  ...
4
5  var db, _ := gorm.Open("mysql", "root:root@/todolist?charset=utf8&parseTime=True&loc=Loc
6
7  +type TodoItemModel struct{
8  +      Id int `gorm:"primary_key"`
9  +      Description string
10 +      Completed bool
11 +}
12 +
13
14 ...
15
16 func main() {
17     defer db.Close()
18
19 +     db.Debug().DropTableIfExists(&TodoItemModel{})
20 +     db.Debug().AutoMigrate(&TodoItemModel{})
21 +
22     log.Info("Starting Todolist API server")
23     router := mux.NewRouter()
24     router.HandleFunc("/healthz", Healthz).Methods("GET")
25     http.ListenAndServe(":8000", router)
26 }
```



In lines 7-11, we're defining the `TodoItem` model as what we described earlier. Everything here is self-explanatory.

In lines 19-20, we're running `automigration` against our MySQL database immediately after starting our API server. See it in action here:



The image shows two terminal windows. The top window, titled '1: Terminal', shows the command prompt 'fadhil@fadhil-ThinkPad-T470p todolist-mysql-go \$' with a cursor. The bottom window, titled '2: Terminal', shows the command prompt 'fadhil@fadhil-ThinkPad-T470p ~ \$' with a cursor.

Yay, our database is ready.

## 4. Create `TodoItem` operation

Add the following lines in our `todolist.go` file:

```
1  package main
2
3  import (
4      "io"
5      "net/http"
6      "github.com/gorilla/mux"
7      log "github.com/sirupsen/logrus"
8      _ "github.com/go-sql-driver/mysql"
9      "github.com/jinzhu/gorm"
10     _ "github.com/jinzhu/gorm/dialects/mysql"
11 +   "encoding/json"
12 )
13
14 ...
```

```

15
16 +func CreateItem(w http.ResponseWriter, r *http.Request) {
17 +    description := r.FormValue("description")
18 +    log.WithFields(log.Fields{"description": description}).Info("Add new TodoItem. S
19 +    todo := &TodoItemModel{Description: description, Completed: false}
20 +    db.Create(&todo)
21 +    result := db.Last(&todo)
22 +    w.Header().Set("Content-Type", "application/json")
23 +    json.NewEncoder(w).Encode(result.Value)
24 +
25 +}
26
27 ...
28
29     log.Info("Starting Todolist API server")
30     router := mux.NewRouter()
31     router.HandleFunc("/healthz", Healthz).Methods("GET")
32 +    router.HandleFunc("/todo", CreateItem).Methods("POST")
33     http.ListenAndServe(":8000", router)
34 }

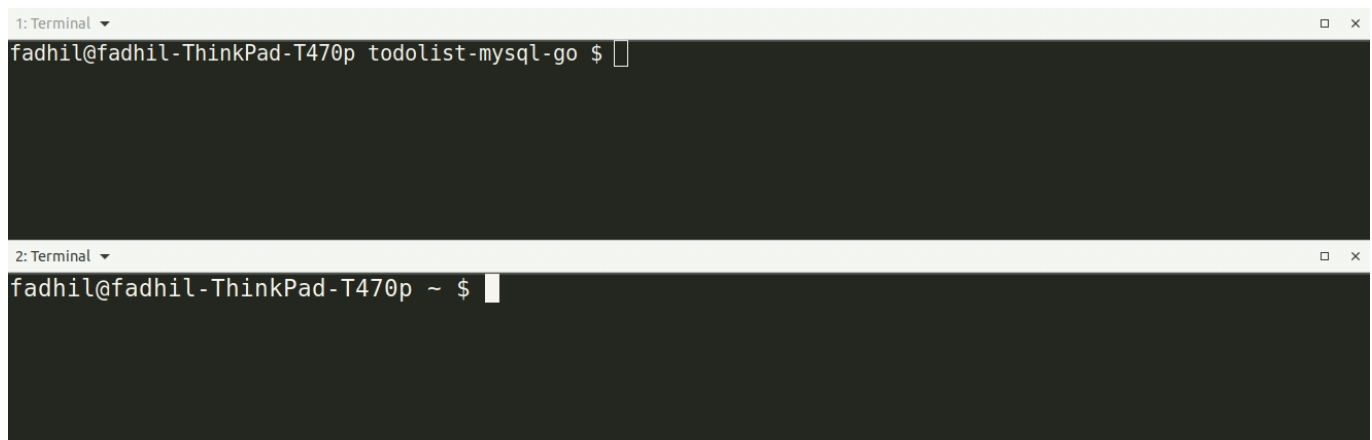
```

todolist-4.go hosted with ❤️ by GitHub

[view raw](#)

On line 16, we obtain the value from the POST operation with `r.FormValue("parameter")`. We use the value as the description to insert into our database. After that, we create the `todo` object and persist it in our database. Lastly, we query the database and return the query result to the client to make sure the operation is success.

In line 32, we register the new route `/todo` with an HTTP POST request into our new `CreateItem()` function.



The image shows two terminal windows. The top window, titled '1: Terminal', shows the command prompt 'fadhil@fadhil-ThinkPad-T470p todolist-mysql-go \$' with a cursor. The bottom window, titled '2: Terminal', shows the command prompt 'fadhil@fadhil-ThinkPad-T470p ~ \$' with a cursor.

fadhil@fadhil-ThinkPad-T470p ~ \$

## 5. Continue with the rest of the CRUD operations

Let's continue with the rest of the Read , Update , and Delete operations.

```
1
2  import (
3      "io"
4      "net/http"
5      "github.com/gorilla/mux"
6      log "github.com/sirupsen/logrus"
7      _ "github.com/go-sql-driver/mysql"
8      "github.com/jinzhu/gorm"
9      _ "github.com/jinzhu/gorm/dialects/mysql"
10     "encoding/json"
11     + "strconv"
12 )
13
14 ...
15
16 +func UpdateItem(w http.ResponseWriter, r *http.Request) {
17 +    // Get URL parameter from mux
18 +    vars := mux.Vars(r)
19 +    id, _ := strconv.Atoi(vars["id"])
20 +
21 +    // Test if the TodoItem exist in DB
22 +    err := GetItemByID(id)
23 +    if err == false {
24 +        w.Header().Set("Content-Type", "application/json")
25 +        io.WriteString(w, `{"updated": false, "error": "Record Not Found"}`)
26 +    } else {
27 +        completed, _ := strconv.ParseBool(r.FormValue("completed"))
28 +        log.WithFields(log.Fields{"Id": id, "Completed": completed}).Info("Update")
29 +        todo := &TodoItemModel{}
30 +        db.First(&todo, id)
31 +        todo.Completed = completed
32 +        db.Save(&todo)
33 +        w.Header().Set("Content-Type", "application/json")
34 +        io.WriteString(w, `{"updated": true}`)
```

```

35 +     }
36 +}
37 +
38 +func DeleteItem(w http.ResponseWriter, r *http.Request) {
39 +    // Get URL parameter from mux
40 +    vars := mux.Vars(r)
41 +    id, _ := strconv.Atoi(vars["id"])
42 +
43 +    // Test if the TodoItem exist in DB
44 +    err := GetItemByID(id)
45 +    if err == false {
46 +        w.Header().Set("Content-Type", "application/json")
47 +        io.WriteString(w, `{"deleted": false, "error": "Record Not Found"}`)
48 +    } else {
49 +        log.WithFields(log.Fields{"Id": id}).Info("Deleting TodoItem")
50 +        todo := &TodoItemModel{}
51 +        db.First(&todo, id)
52 +        db.Delete(&todo)
53 +        w.Header().Set("Content-Type", "application/json")
54 +        io.WriteString(w, `{"deleted": true}`)
55 +    }
56 +}
57 +
58 +func GetItemByID(Id int) bool {
59 +    todo := &TodoItemModel{}
60 +    result := db.First(&todo, Id)
61 +    if result.Error != nil{
62 +        log.Warn("TodoItem not found in database")
63 +        return false
64 +    }
65 +    return true
66 +}
67 +
68 +func GetCompletedItems(w http.ResponseWriter, r *http.Request) {
69 +    log.Info("Get completed TodoItems")
70 +    completedTodoItems := GetTodoItems(true)
71 +    w.Header().Set("Content-Type", "application/json")
72 +    json.NewEncoder(w).Encode(completedTodoItems)
73 +}
74 +
75 +func GetIncompleteItems(w http.ResponseWriter, r *http.Request) {
76 +    log.Info("Get Incomplete TodoItems")
77 +    IncompleteTodoItems := GetTodoItems(false)
78 +    w.Header().Set("Content-Type", "application/json")

```

```

79 +     json.NewEncoder(w).Encode(IncompleteTodoItems)
80 +}
81
82 +func GetTodoItems(completed bool) interface{} {
83 +     var todos []TodoItemModel
84 +     TodoItems := db.Where("completed = ?", completed).Find(&todos).Value
85 +     return TodoItems
86 }
87
88 ...
89
90     log.Info("Starting Todolist API server")
91     router := mux.NewRouter()
92     router.HandleFunc("/healthz", Healthz).Methods("GET")
93 +     router.HandleFunc("/todo-completed", GetCompletedItems).Methods("GET")
94 +     router.HandleFunc("/todo-incomplete", GetIncompleteItems).Methods("GET")
95     router.HandleFunc("/todo", CreateItem).Methods("POST")
96 +     router.HandleFunc("/todo/{id}", UpdateItem).Methods("POST")
97 +     router.HandleFunc("/todo/{id}", DeleteItem).Methods("DELETE")
98     http.ListenAndServe(":8000", router)
99 }

```

todolist-5.go hosted with ❤ by GitHub

[view raw](#)

We should be able to perform Create , Read , Update , and Delete operations on the `TodoItem` s.

In order to update a `TodoItem` object, we'll first query our database to ensure the item actually exists. I created a function, `GetItemById()` , for this purpose. The equivalent SQL query is `SELECT * FROM todo_item_models WHERE Id = <Id> LIMIT 1; .`

Both `UpdateItem()` and `DeleteItem()` functions will invoke `GetItemById()` first before actually updating or deleting them.

Notice I imported the `strconv` package in there. The package is used to convert the `String` variable into an `Integer` before we run the SQL queries.

The `GetCompletedItems()` and `GetIncompletedItems()` functions are self-explanatory as well. It'll execute a SQL `SELECT` query and encode it into JSON before returning it to the client.

## 6. Connect with the front end

In order to connect the API server to the front-end page, we need to give the CORS headers to the response. CORS is a mechanism that allows restricted resources on a web page to be requested from another domain outside the domain from which the first resource was served.

To do so, you'll need to install the Go `cors` package.

```
$ go get -u github.com/rs/cors
```

Add these lines to your Go code:

```
1  package main
2
3  import (
4      "io"
5      "net/http"
6      "github.com/gorilla/mux"
7      log "github.com/sirupsen/logrus"
8      _ "github.com/go-sql-driver/mysql"
9      "github.com/jinzhu/gorm"
10     _ "github.com/jinzhu/gorm/dialects/mysql"
11     "encoding/json"
12     "strconv"
13     + "github.com/rs/cors"
14 )
15
16 ...
17
18 log.Info("Starting Todolist API server")
19 router := mux.NewRouter()
20 router.HandleFunc("/healthz", Healthz).Methods("GET")
21 router.HandleFunc("/todo-completed", GetCompletedItems).Methods("GET")
22 router.HandleFunc("/todo-incomplete", GetIncompleteItems).Methods("GET")
23 router.HandleFunc("/todo", CreateItem).Methods("POST")
24 router.HandleFunc("/todo/{id}", UpdateItem).Methods("POST")
25 router.HandleFunc("/todo/{id}", DeleteItem).Methods("DELETE")
26 - http.ListenAndServe(":8000", router)
27 +
28 + handler := cors.New(cors.Options{
```

```
29 +             AllowedMethods: []string{"GET", "POST", "DELETE", "PATCH", "OPTIONS"},
30 +         }).Handler(router)
31 +
32 +     http.ListenAndServe(":8000", handler)
33 }
```

todolist-6.go hosted with ❤ by GitHub

[view raw](#)

In lines 28-32, we're wrapping the CORS handler around our existing application.

Now clone my front-end site for this Go app.

```
$ git clone https://github.com/sdil/todo.git
```

Then, open the `index.html` page in your browser. There's nothing special in this simple jQuery page. I've modified the original code made by [themaxsandelin](#) to execute AJAX requests when you click buttons in the page. Check the `resources/js/main.js` file to see how the web front end works.

## 7. Build the package

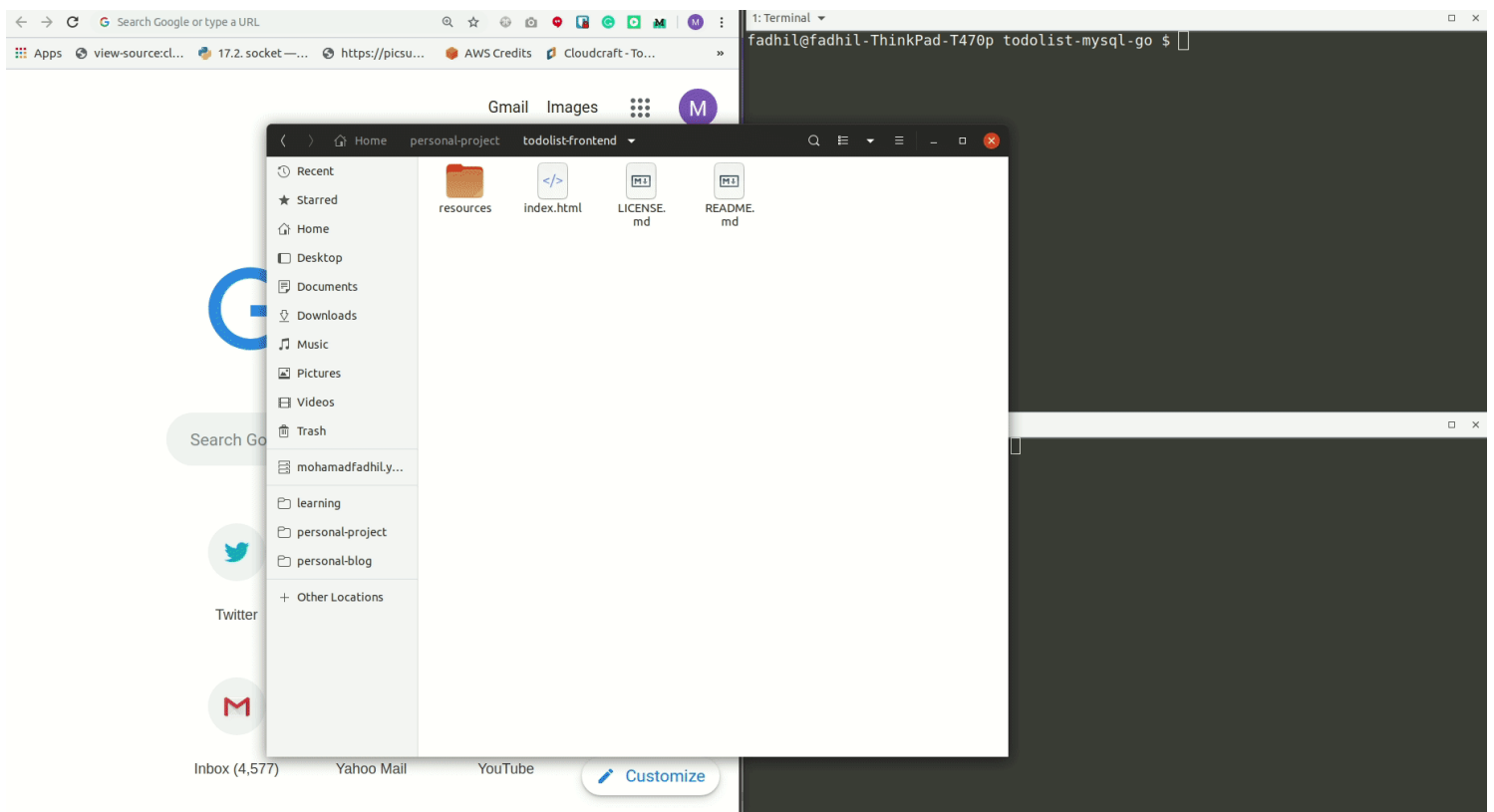




Now we're done with our coding. You can now compile your code.

```
$ go build
$ ./todolist-mysql-go
```

## A Little End-to-End Demonstration



Congratulations, you've built yourself a to-do list in Golang!

You can find the full source code in my GitHub repository. If you're stuck at any point in this tutorial, feel free to refer there.

sdil/learning




Compilation of my learning projects. Contribute to sdil/learning development by creating an account on GitHub.  
github.com

## Don't Stop Learning

This is, in fact, only a tiny part of Golang.

There are many more things you can explore next in Golang. You'll have more reasons to love Golang. I hope this tutorial inspired you to go explore more.

Some rights reserved 

[Golang](#) [Api Development](#) [Tutorial](#) [Programming](#) [Software Development](#)

[About](#) [Help](#) [Legal](#)

Get the Medium app

