BACKEND IN GO

SERVER CREATION:

 **Import the Required Packages**:

- **net/http**: For handling HTTP requests and responses.
- **fmt**: For printing logs to the console.

 **Define Routes (Endpoints)**:

- Use **http.HandleFunc(path, handlerFunc)** to specify which function will handle specific routes.

 **Start the Server**:

- Use **http.ListenAndServe(port, nil)** to start the server on a given port.
- **Port 8080** is commonly used for development servers.

```go
package main
import (
    "fmt"
    "net/http"
)
func main(){
    fmt.Println("server not yet started....")

    http.HandleFunc("/")
    err:=http.ListenAndServe(":8080",nil)
    if err!=nil{
        fmt.Println("server failed to start",err)
    }

}
```

```
C:\Users\tallu\OneDrive\Documents\go>go run server.go
# command-line-arguments
.\server.go:9:18: not enough arguments in call to http.HandleFunc
        have (string)
        want (string, func(http.ResponseWriter, *http.Request))
```

 **http.HandleFunc("/", homeHandler)**:

- Associates the / (root) URL path with the homeHandler function.

 **homeHandler**:

- Called whenever someone visits **http://localhost:8080/**
- Uses fmt.Fprintln(w, "message") to send an HTTP response back to the client.

 **http.ListenAndServe(":8080", nil)**:

- Starts a server on port **8080**.
- The second argument is nil, which means it uses the default **ServeMux** to handle requests.

```go
package main
import (
    "fmt"
    "net/http"
)

func home(w http.ResponseWriter,r *http.Request){
fmt.Fprintln(w)
fmt.Fprintln(w,"hello")
}
func main(){
    fmt.Println("server not yet started....")

    http.HandleFunc("/",home)
    err:=http.ListenAndServe(":8080",nil)
    if err!=nil{
        fmt.Println("server failed to start",err)
    }

}
```

```
C:\Users\tallu\OneDrive\Documents\go>go run server.go
server not yet started....
```

```
hello
```

Build in a project:

Create a folder with project name and open cmd in that folder and run following commands

**go mod init projectname**

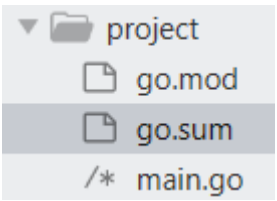
It will create go.mod file

```
C:\Users\tallu\OneDrive\Documents\go\project>go mod init tnrstudent
go: creating new go.mod: module tnrstudent
go: to add module requirements and sums:
        go mod tidy
```

**go get go.mongodb.org/mongo-driver/mongo**

It will add go.sum file configuring required modules

```
C:\Users\tallu\OneDrive\Documents\go\project>go get go.mongodb.org/mongo-driver/mongo
go: downloading go.mongodb.org/mongo-driver v1.17.1
go: downloading github.com/youmark/pkcs8 v0.0.0-20240726163527-a2c0da244d78
go: downloading github.com/golang/snappy v0.0.4
go: downloading github.com/klauspost/compress v1.13.6
go: downloading github.com/xdg-go/scram v1.1.2
go: downloading github.com/xdg-go/stringprep v1.0.4
go: downloading github.com/montanaflynn/stats v0.7.1
go: downloading golang.org/x/sync v0.8.0
go: downloading golang.org/x/crypto v0.26.0
go: downloading golang.org/x/text v0.17.0
go: downloading github.com/xdg-go/pbkdf2 v1.0.0
go: added github.com/golang/snappy v0.0.4
go: added github.com/klauspost/compress v1.13.6
go: added github.com/montanaflynn/stats v0.7.1
go: added github.com/xdg-go/pbkdf2 v1.0.0
go: added github.com/xdg-go/scram v1.1.2
go: added github.com/xdg-go/stringprep v1.0.4
go: added github.com/youmark/pkcs8 v0.0.0-20240726163527-a2c0da244d78
go: added go.mongodb.org/mongo-driver v1.17.1
go: added golang.org/x/crypto v0.26.0
go: added golang.org/x/sync v0.8.0
go: added golang.org/x/text v0.17.0
```



**Connecting to mongodb:**

Import packages:

```
import (
    "fmt"
    "net/http"
    "context"
    "time"
    "go.mongodb.org/mongo-driver/mongo"
    "go.mongodb.org/mongo-driver/mongo/options"

)
```

1.

- **context:** Used to set a timeout when connecting to the MongoDB server.
- **time:** Used to specify a 10-second connection timeout.
- **mongo:** MongoDB driver for Go.
- **options:** Used to configure the connection string for MongoDB.

```
var client *mongo.Client
```

- We declared a global client variable to be reused throughout the program.

```
func connectdb(){
    // Create a context with a 10-second timeout for the connection attempt
    ctx, cancel := context.WithTimeout(context.Background(), 10*time.Second)
    defer cancel()


    var err error
    clientoptions:= options.Client().ApplyURI(mongoURI)
    client,err=mongo.Connect(ctx,clientoptions)
    if err!=nil{
        fmt.Println("failed to connect mongodb:",err)
        return
    }
    err=client.Ping(ctx,nil)
    if err!=nil{
        fmt.Println("Failed to ping mongodb: ",err)
        return
    }
    fmt.Println("connected to mongodb successfully")
}
```

 **Create a context**: Sets a **10-second timeout** for connecting to the MongoDB server.

 **mongo.Connect()**: Attempts to connect to MongoDB.

 **client.Ping()**: Verifies the connection to the database.

Creating database and collection:

 Open **MongoDB Compass**.

 Connect to **mongodb://localhost:27017**.

 Select the **databasename** database.

 Check for the **collection name** collection.

Mongodb is lazy creation so weed to insert atleast one document to undertand whether db and collection is created or not.

```go
func createcollection(){
    dbname:="tnr"
    colname:="student"
    collection:=client.Database(dbname).Collection(colname)
    dummyDocument := bson.D{
        {Key: "name", Value: "TNR"},
        {Key: "age", Value: 22},
        {Key:"course",Value:"CSE"},
        {Key:"Collage",Value:"SRM"},
    }
    ctx,cancel:=context.WithTimeout(context.Background(),5*time.Second)
    defer cancel()

    result,err:=collection.InsertOne(ctx,dummyDocument)
    if err!=nil{
        fmt.Printf("failed")
    }else{
        fmt.Printf("successfully done %v",result.InsertedID)
    }

}
```

```
C:\Users\tallu\OneDrive\Documents\go\project>go run main.go
server not yet started....
connected to mongodb successfully
successfully done ObjectID("675e861468e5302c7e64a256")exit status 0xc000013a
```

```
test> show dbs
admin     144.00 KiB
config    108.00 KiB
local      88.00 KiB
practise  684.00 KiB
project   100.00 KiB
projects   80.00 KiB
tnr        72.00 KiB
test> use tnr
switched to db tnr
tnr> show collections
student
tnr> db.student.find()
[
  {
    _id: ObjectId('675e803d530b5ec47a791edc'),
    name: 'TNR Sample',
    age: 25
  },
  {
    _id: ObjectId('675e861468e5302c7e64a256'),
    name: 'TNR',
    age: 22,
    course: 'CSE',
    Collage: 'SRM'
  }
]
```

**Getting all values using GET method :**

```go
func getdata(w http.ResponseWriter,r *http.Request){
    if r.Method != http.MethodGet {
        http.Error(w, "that Method not allowed", http.StatusMethodNotAllowed
        return
    }
    dbname:="tnr"
    colname:="student"
    collection:=client.Database(dbname).Collection(colname)
    ctx,cancel:=context.WithTimeout(context.Background(),10*time.Second)
    defer cancel()

    cursor,err:=collection.Find(ctx,bson.D{})
    if err!=nil{
        http.Error(w,"error occured",http.StatusInternalServerError)
        return
    }
    var results []bson.M
    err=cursor.All(ctx,&results)
    if err!=nil{
        http.Error(w,"error occured",http.StatusInternalServerError)
        return
    }

    w.Header().Set("Content-Type","application/json")
    err=json.NewEncoder(w).Encode(results)
    if err!=nil{
        http.Error(w,"error occured",http.StatusInternalServerError)
        return
```

- Use http.MethodGet instead of default POST logic.
- Make sure to call collection.Find() properly and print the list as JSON.
- Set the content-type header properly.

```
C:\Users\tallu\OneDrive\Documents\go\project>go run main.go
server not yet started....
connected to mongodb successfully
successfully connected
```

```
[
  {
    "_id": "675e803d530b5ec47a791edc",
    "age": 25,
    "name": "TNR Sample"
  },
  {
    "Collage": "SRM",
    "_id": "675e861468e5302c7e64a256",
    "age": 22,
    "course": "CSE",
    "name": "TNR"
  },
  {
    "Collage": "SRM",
    "_id": "675e870f8b62b2fe3957c145",
    "age": 22,
    "course": "CSE",
    "name": "TNR"
  }
]
```


POST :

1. **Create a /postdata route** that accepts a **POST request**.
2. **Parse the JSON request body** into a struct.
3. **Insert the data into the MongoDB collection**.
4. **Return a success message with the inserted ID**.

```go
func postdata(w http.ResponseWriter,r *http.Request){
    if r.Method!=http.MethodPost{
        http.Error(w,"method not allowed",http.StatusMethodNotAllowed)
        return
    }
    w.Header().Set("Content-Type","application/json")
    var newData map[string]interface{}
    err:=json.NewDecoder(r.Body).Decode(&newData)
    if err!=nil{
        http.Error(w,"invalid json format",http.StatusBadRequest)
        return
    }
    fmt.Println("new data reciveed:",newData)
    dbname:="tnr"
    colname:="student"
    collection:=client.Database(dbname).Collection(colname)
    ctx,cancel:=context.WithTimeout(context.Background(),10*time.Second)
    defer cancel()
    result,err:=collection.InsertOne(ctx,newData)
    if err!=nil{
        http.Error(w,"error ocuured while inserting",http.StatusInternalServerError)
        return
    }
    fmt.Println("data inserted scueesfully",result.InsertedID)
    w.WriteHeader(http.StatusCreated)
    json.NewEncoder(w).Encode(map[string]interface{}{
        "message":"data inserted successfully",
```

 **POST Route /postdata**:

- Accepts only **POST requests**.
- Parses the **request body** as JSON and converts it to a **map[string]interface{}**.
- Inserts the data into the MongoDB **student** collection.
- Responds with a success message and the **inserted ID**.

 **New Route Added**:

- **/postdata** - Handles **POST requests** to add data to the collection.
- **/getdata** - Handles **GET requests** to retrieve the list of data.

 **Content-Type Handling**:

- Sets the **Content-Type** of the response to application/json in both GET and POST routes.

```
C:\Users\tallu\OneDrive\Documents\go\project>go run main.go
server not yet started....
connected to mongodb successfully
successfully connected
new data reciveed: map[College:NIT age:25 course:IT name:John]
data inserted scueesfully ObjectID("6762441a18dfb1018d842aae")
```

```
{
    "id": "6762441a18dfb1018d842aae",
    "message": "data inserted successfully"
}
```

DELETING record :

- **Create a /deletedata route** to handle **DELETE requests**.
- **Extract the "name" from query parameters**.
- **Use MongoDB's DeleteOne method** to remove the document with the matching name.
- **Return a success message** if the deletion is successful.

```go
func deletedata(w http.ResponseWriter, r *http.Request) {
    if r.Method != http.MethodDelete {
        http.Error(w, "Invalid request method", http.StatusMethodNotAllowed)
        return
    }

    dbname := "tnr"
    colname := "student"
    collection := client.Database(dbname).Collection(colname)
    var filter bson.M
    err := json.NewDecoder(r.Body).Decode(&filter)
    if err != nil {
        http.Error(w, "Invalid JSON data", http.StatusBadRequest)
        return
    }
    if len(filter) == 0 {
        http.Error(w, "No filter provided", http.StatusBadRequest)
        return
    }
    ctx, cancel := context.WithTimeout(context.Background(), 10*time.Second)
    defer cancel()

    result, err := collection.DeleteOne(ctx, filter)
    if err != nil {
        http.Error(w, "Error while deleting data", http.StatusInternalServerError)
        return
    }
    if result.DeletedCount == 0 {
```

```
128     if result.DeletedCount == 0 {
129         fmt.Fprintf(w, "❌ No records found matching filter: %v", filter)
130     } else {
131         fmt.Fprintf(w, "☑ Data deleted successfully matching filter: %v", filter)
132     }
133 }
```

```
C:\Users\tallu\OneDrive\Documents\go\project>go run main.go
server not yet started....
connected to mongodb successfully
successfully connected
```

```
1  {
2      "name":"John"
3  }
```

```
1  ✅ Data deleted successfully matching filter: map[name:John]
```