Creating RESTful API

LAB 08

Learning Objective

After completing this lab session you should be able to

Design and Implement **RESTful API**

Write a **RESTful API client** in Go

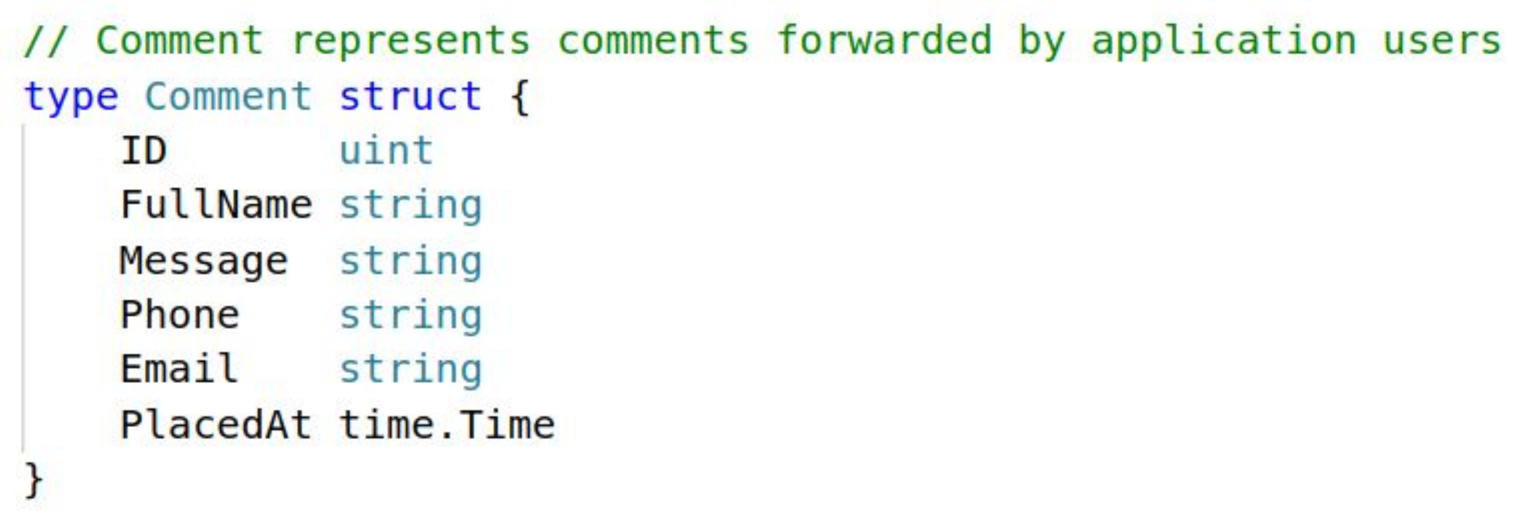
The sample code for this lab is found in the following link

https://github.com/betsegawlemma/sample-restaurant-rest-api

Do not forget to adjust the imports

Create RESTful API for Comment Service

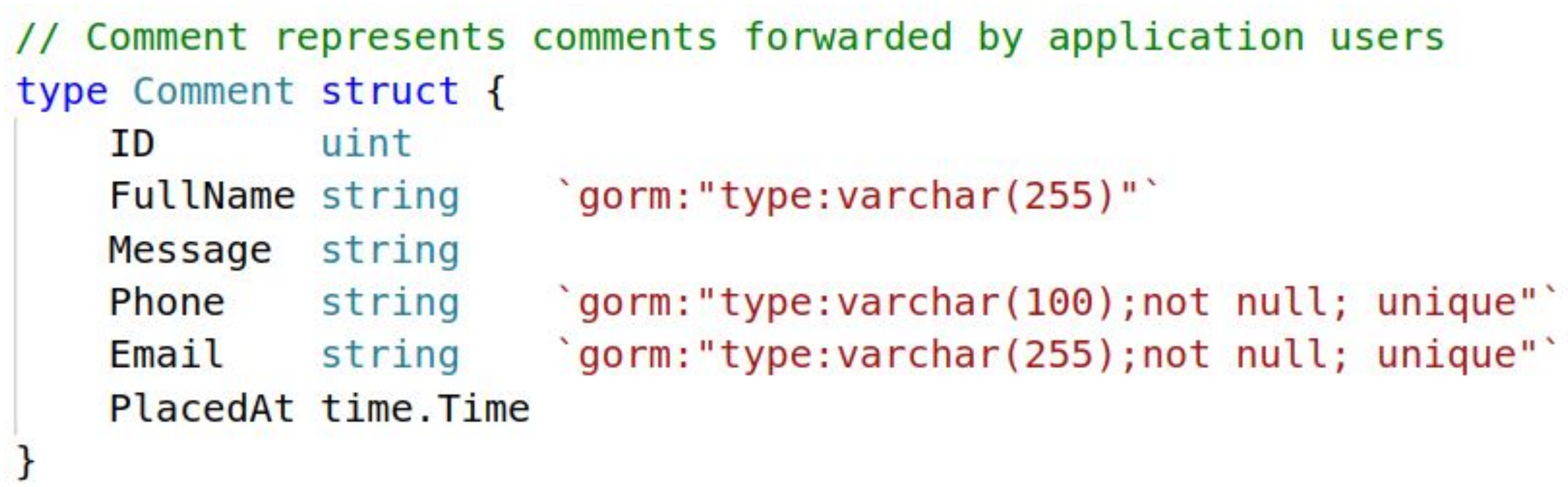
`Comment` Entity



```go
// Comment represents comments forwarded by application users
type Comment struct {
	ID       uint
	FullName string
	Message  string
	Phone    string
	Email    string
	PlacedAt time.Time
}
```

Create RESTful API for Comment Service

`Comment` Entity with **GORM** tags



```go
// Comment represents comments forwarded by application users
type Comment struct {
    ID       uint
    FullName string    `gorm:"type:varchar(255)"`
    Message  string
    Phone    string    `gorm:"type:varchar(100);not null; unique"`
    Email    string    `gorm:"type:varchar(255);not null; unique"`
    PlacedAt time.Time
}
```

Configuring GORM

Download GORM

```
go get github.com/jinzhu/gorm/dialects/postgres
```

Connecting to postgres database using GORM

Configuring GORM

Importing GORM

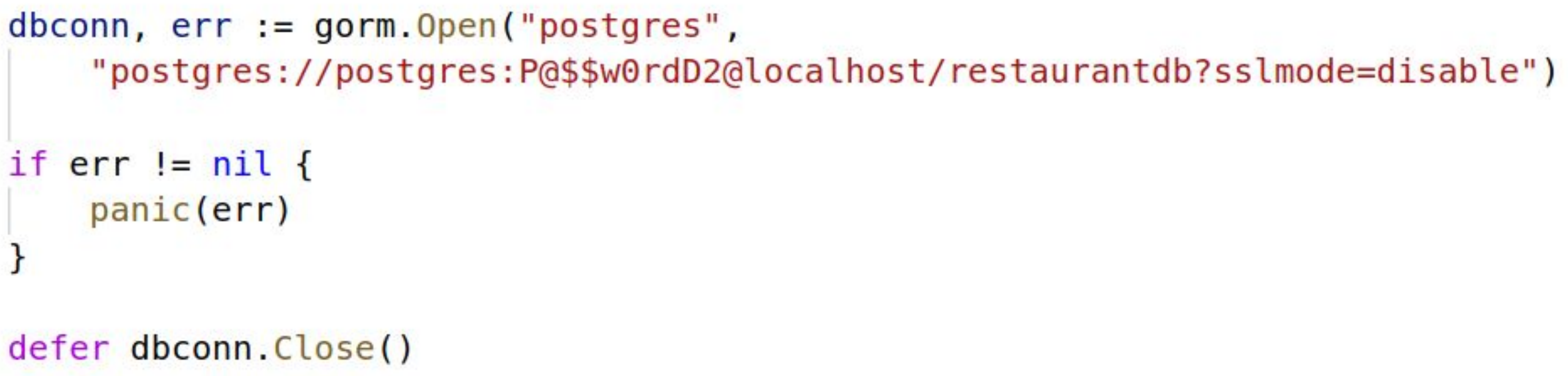
```
    "github.com/jinzhu/gorm"

    _ "github.com/jinzhu/gorm/dialects/postgres"
```

Configuring GORM

Connecting to postgres database using GORM

The difference to what you have seen before is, just changing the **sql** to **gorm**

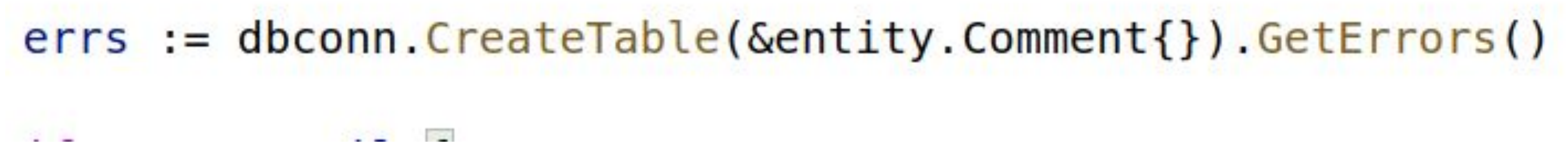
Do not forget to adjust the connection string according to your environment



```go
dbconn, err := gorm.Open("postgres",
    "postgres://postgres:P@$$w0rdD2@localhost/restaurantdb?sslmode=disable")

if err != nil {
    panic(err)
}

defer dbconn.Close()
```

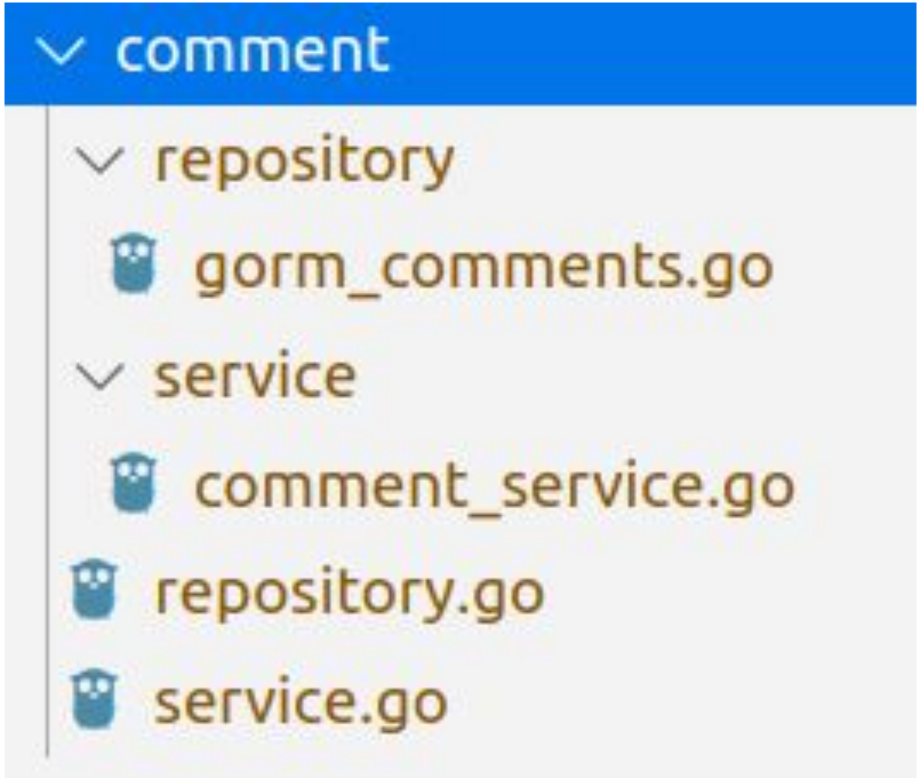
Configuring GORM

Creating tables automatically using GORM



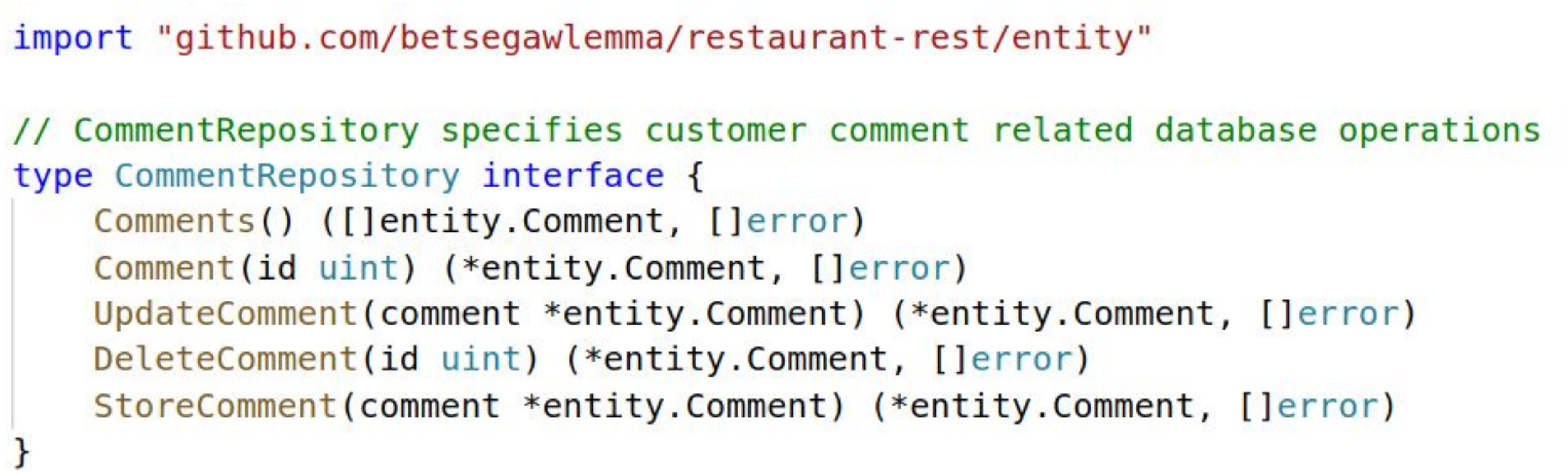
```
errs := dbconn.CreateTable(&entity.Comment{}).GetErrors()
```

Comment Featue



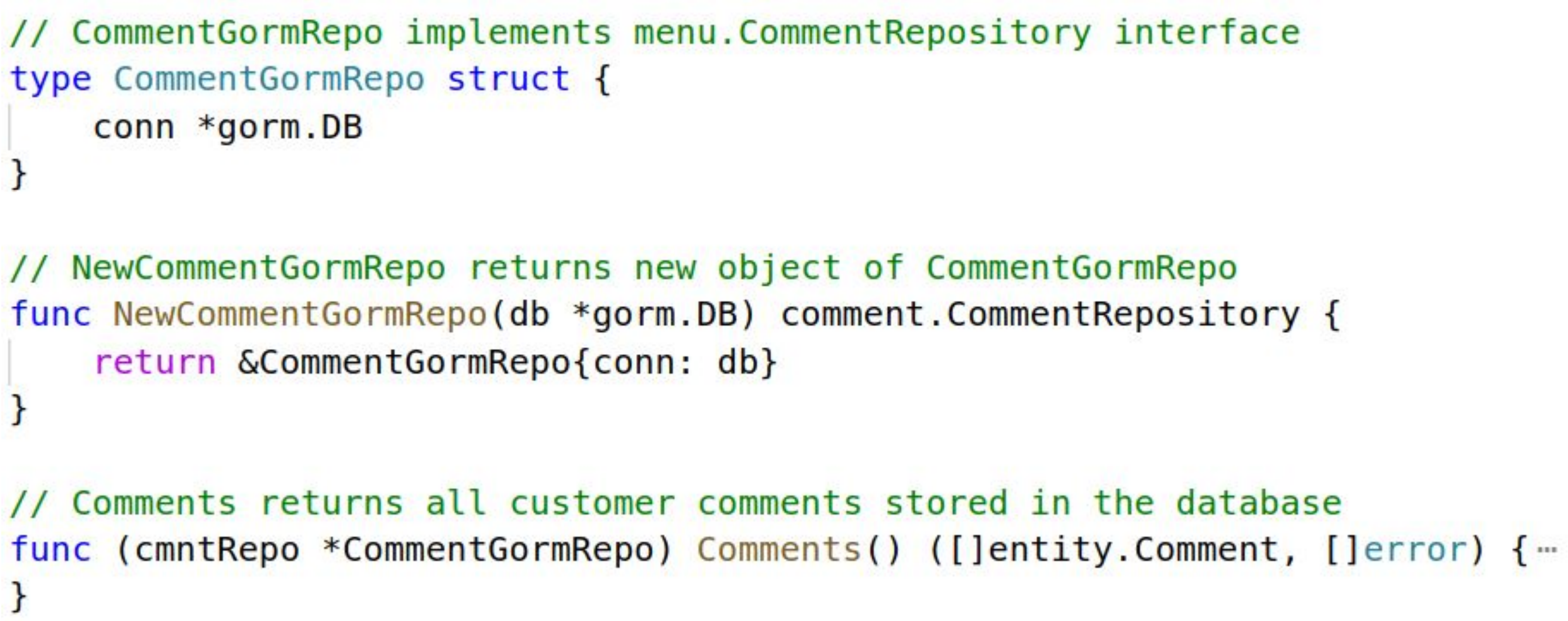
```
comment
  repository
    gorm_comments.go
  service
    comment_service.go
  repository.go
  service.go
```

CommentRepository Interfae



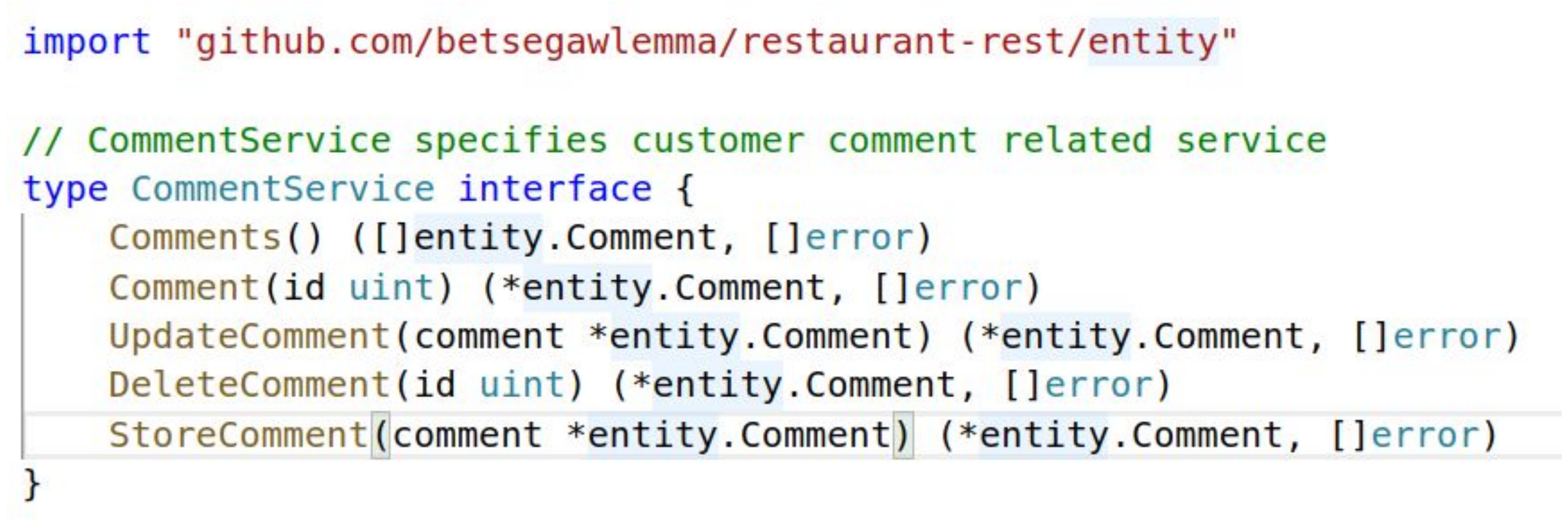
```
import "github.com/betsegawlemma/restaurant-rest/entity"

// CommentRepository specifies customer comment related database operations
type CommentRepository interface {
    Comments() ([]entity.Comment, []error)
    Comment(id uint) (*entity.Comment, []error)
    UpdateComment(comment *entity.Comment) (*entity.Comment, []error)
    DeleteComment(id uint) (*entity.Comment, []error)
    StoreComment(comment *entity.Comment) (*entity.Comment, []error)
}
```

CommentRepository Implementation



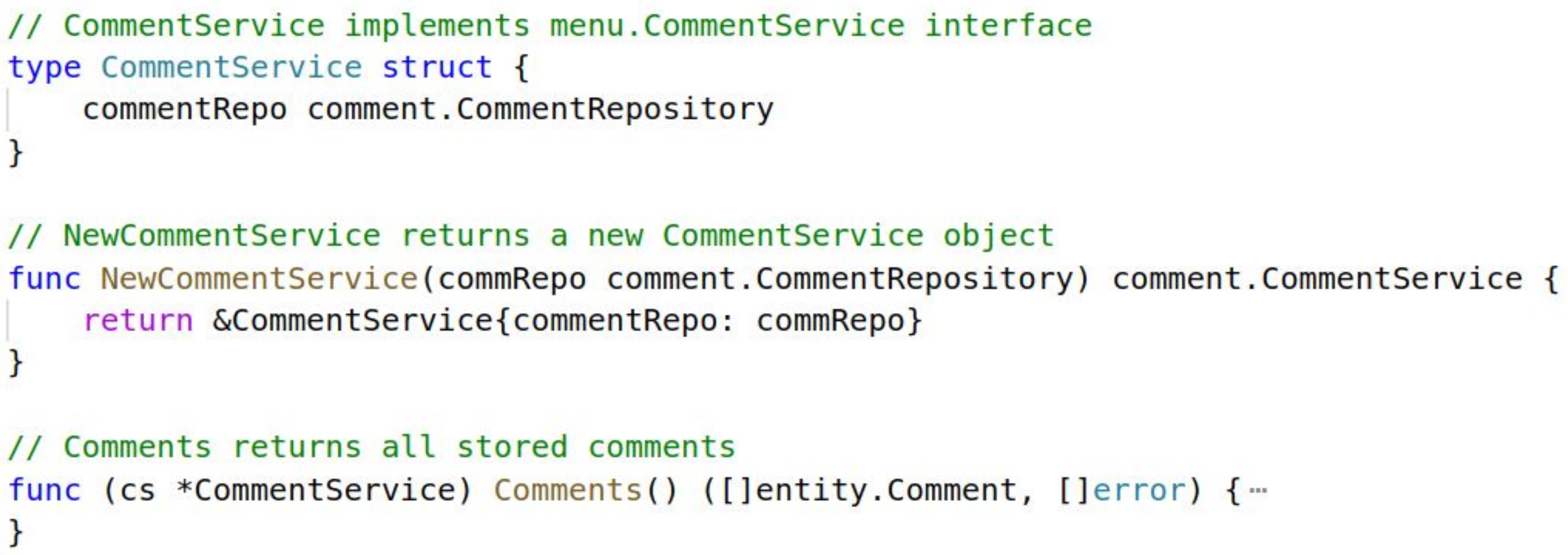
```go
// CommentGormRepo implements menu.CommentRepository interface
type CommentGormRepo struct {
    conn *gorm.DB
}

// NewCommentGormRepo returns new object of CommentGormRepo
func NewCommentGormRepo(db *gorm.DB) comment.CommentRepository {
    return &CommentGormRepo{conn: db}
}

// Comments returns all customer comments stored in the database
func (cmntRepo *CommentGormRepo) Comments() ([]entity.Comment, []error) {⋯
}
```

CommentService Interfae



```go
import "github.com/betsegawlemma/restaurant-rest/entity"

// CommentService specifies customer comment related service
type CommentService interface {
	Comments() ([]entity.Comment, []error)
	Comment(id uint) (*entity.Comment, []error)
	UpdateComment(comment *entity.Comment) (*entity.Comment, []error)
	DeleteComment(id uint) (*entity.Comment, []error)
	StoreComment(comment *entity.Comment) (*entity.Comment, []error)
}
```

CommentService Implementation



```go
// CommentService implements menu.CommentService interface
type CommentService struct {
    commentRepo comment.CommentRepository
}

// NewCommentService returns a new CommentService object
func NewCommentService(commRepo comment.CommentRepository) comment.CommentService {
    return &CommentService{commentRepo: commRepo}
}

// Comments returns all stored comments
func (cs *CommentService) Comments() ([]entity.Comment, []error) {⋯
}
```

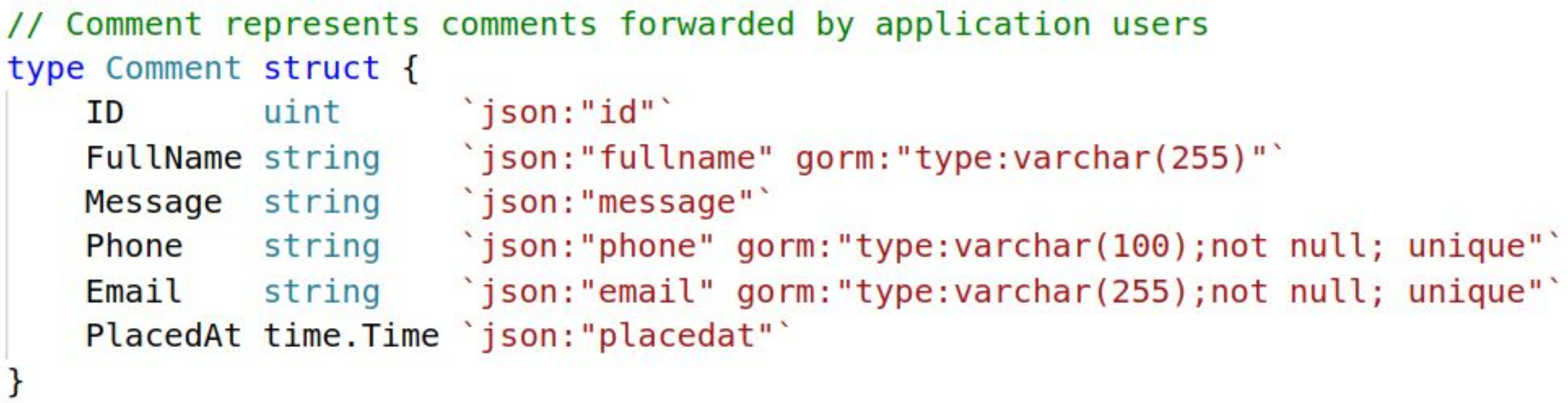
Path Requirement for Comment RESTful API

Method	Route/Path	HTTP status on Success	HTTP status on Failure
`GET`	`/v1/admin/comments`	`StatusOK (200)`	`StatusNotFound (404)`
`GET`	`/v1/admin/comments/:id`	`StatusOK (200)`	`StatusNotFound (404)`
`POST`	`/v1/admin/comments`	`StatusCreated (201)`	`StatusNotFound (404)`
`PUT`	`/v1/admin/comments/:id`	`StatusOK (200), StatusNoContent (204)`	`StatusNotFound (404)`
`DELETE`	`/v1/admin/comments/:id`		

Add JSON tag to the `Comment` Entity

Tag each field of the `Comment` entity using JSON tag as shown

Note that you can mix use multiple tags on a single field



```go
// Comment represents comments forwarded by application users
type Comment struct {
    ID       uint      `json:"id"`
    FullName string    `json:"fullname" gorm:"type:varchar(255)"`
    Message  string    `json:"message"`
    Phone    string    `json:"phone" gorm:"type:varchar(100);not null; unique"`
    Email    string    `json:"email" gorm:"type:varchar(255);not null; unique"`
    PlacedAt time.Time `json:"placedat"`
}
```

`httprouter` Library

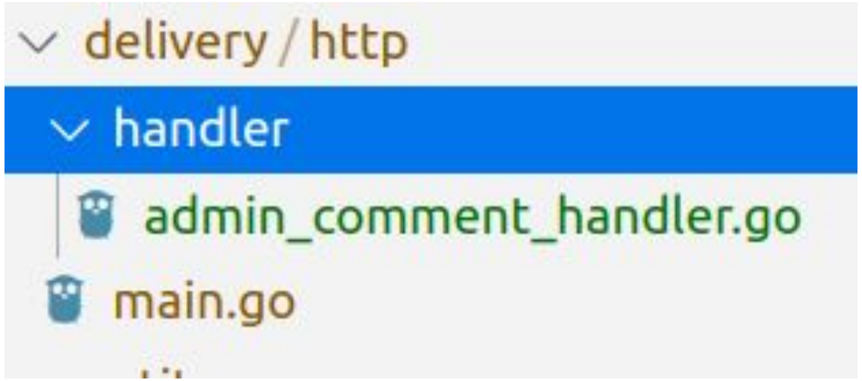
Download the following library

```
    go get github.com/julienschmidt/httprouter
```

Used for constructing RESTful style resource paths shown in the previous slide

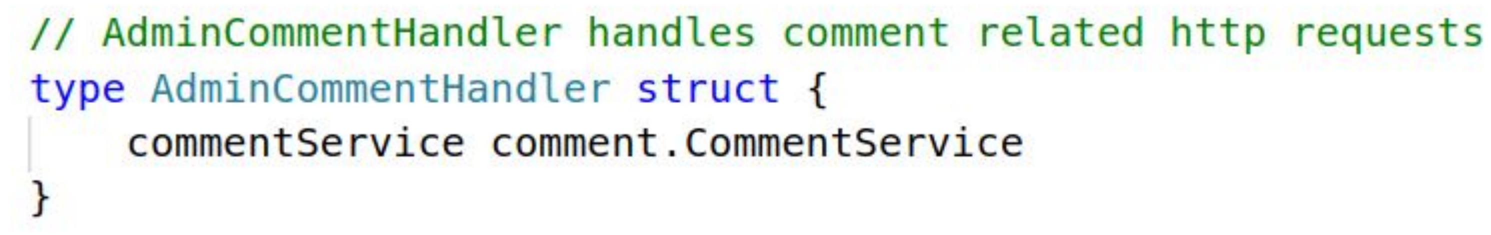
Create `AdminCommentHandler`

Create a file called **`admin_comment_handler.go`** under **`deliver/http/handler`** directory



Create `AdminCommentHandler`

Inside `admin_comment_handler.go` create a struct type that we can use to define the handler functionality

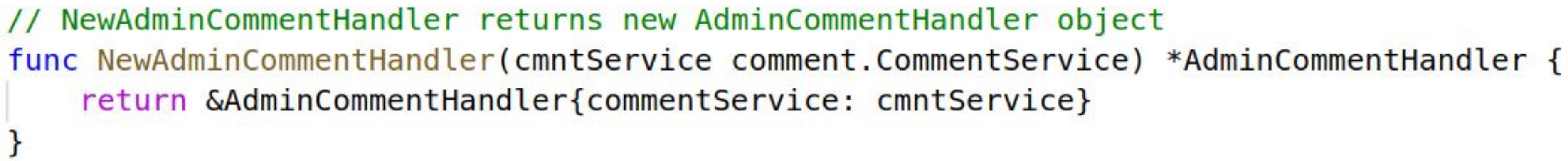


```go
// AdminCommentHandler handles comment related http requests
type AdminCommentHandler struct {
    commentService comment.CommentService
}
```

We use the `CommentService` to interact with the data persistency layer

Create `AdminCommentHandler`

Inside `admin_comment_handler.go` create also a constructor



```go
// NewAdminCommentHandler returns new AdminCommentHandler object
func NewAdminCommentHandler(cmntService comment.CommentService) *AdminCommentHandler {
    return &AdminCommentHandler{commentService: cmntService}
}
```

Create `AdminCommentHandler`

Now we can specify handlers that can handle the following requests

```
GET /v1/admin/comments

GET /v1/admin/comments/:id

POST /v1/admin/comments

PUT /v1/admin/comments/:id

DELETE /v1/admin/comments/:id
```

Create `AdminCommentHandler`

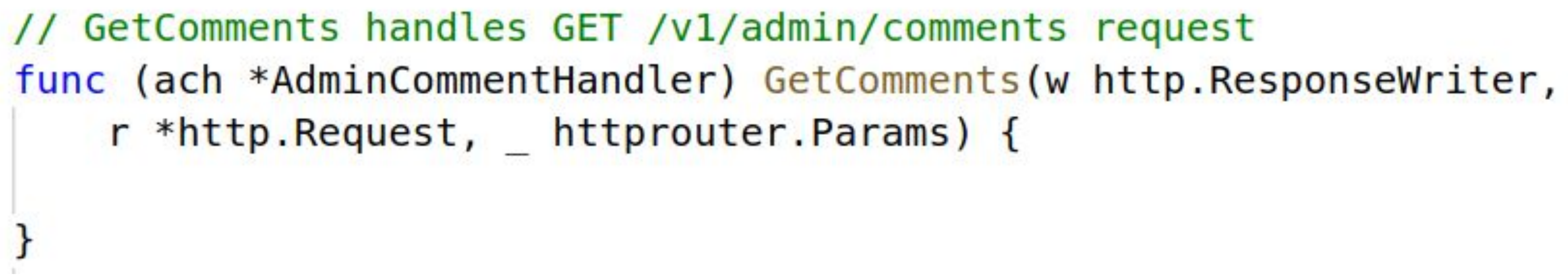
Steps for Handling **`GET /v1/admin/comments`**

- Define the handler function
- Read comments from the persistence layer using the **`CommentService`**
- Convert read data to JSON format
- Write the JSON data to **`ResponseWriter`** object

Create `AdminCommentHandler`

Steps for Handling **`GET /v1/admin/comments`**

Define the handler function



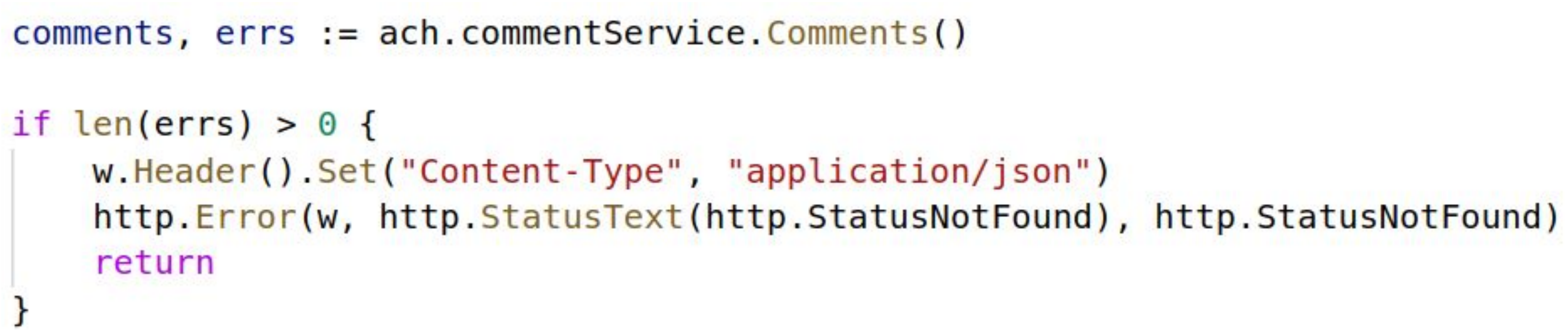
```go
// GetComments handles GET /v1/admin/comments request
func (ach *AdminCommentHandler) GetComments(w http.ResponseWriter,
    r *http.Request, _ httprouter.Params) {

}
```

Note that the handler function signature is different from what you have used to sofar. That is because of **`httprouter`** library

Create `AdminCommentHandler`

Steps for Handling `GET /v1/admin/comments`

Read comments from the persistence layer using the `CommentService`

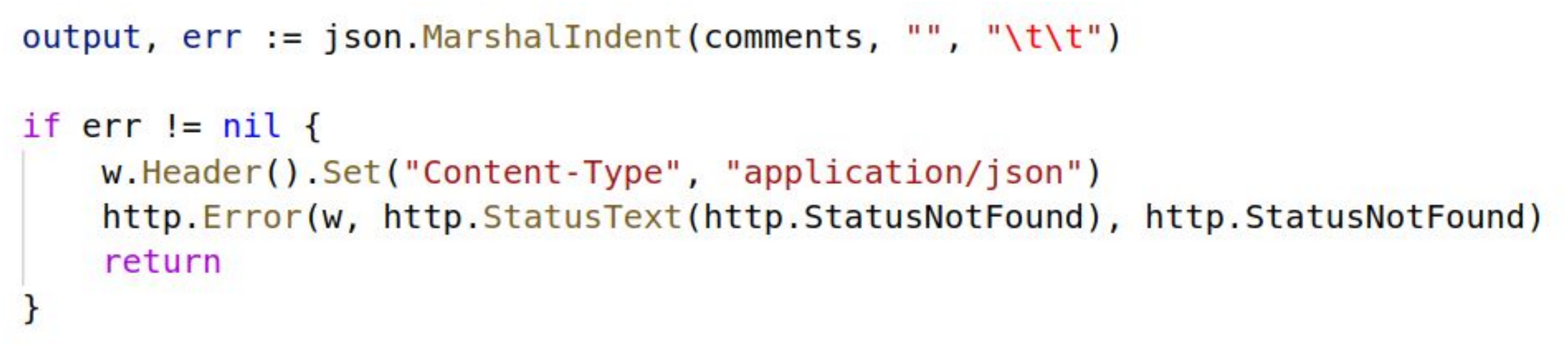


```
comments, errs := ach.commentService.Comments()

if len(errs) > 0 {
    w.Header().Set("Content-Type", "application/json")
    http.Error(w, http.StatusText(http.StatusNotFound), http.StatusNotFound)
    return
}
```

Create AdminCommentHandler

Steps for Handling **GET /v1/admin/comments**

Convert read data to JSON format



```go
output, err := json.MarshalIndent(comments, "", "\t\t")

if err != nil {
    w.Header().Set("Content-Type", "application/json")
    http.Error(w, http.StatusText(http.StatusNotFound), http.StatusNotFound)
    return
}
```

Create AdminCommentHandler

Steps for Handling **GET /v1/admin/comments**

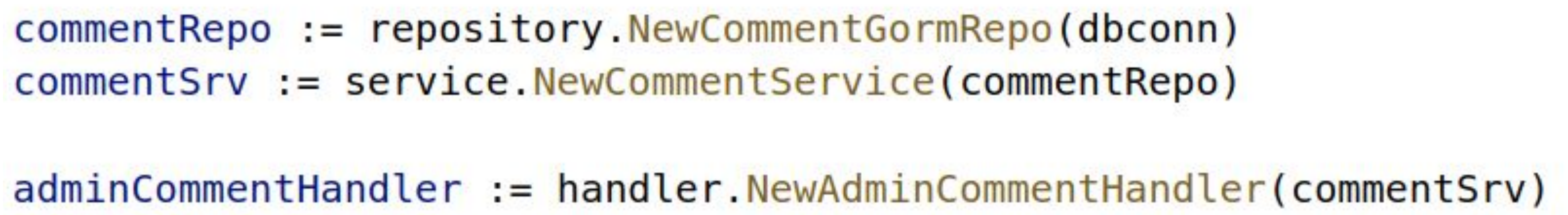
Write the JSON data to `ResponseWriter` object

```
output, err := json.MarshalIndent(comments, "", "\t\t")

if err != nil {⋯
}

w.Header().Set("Content-Type", "application/json")
w.Write(output)
return
```

Using `AdminCommentHandler`

Create and Instantiate the following types inside `main` function

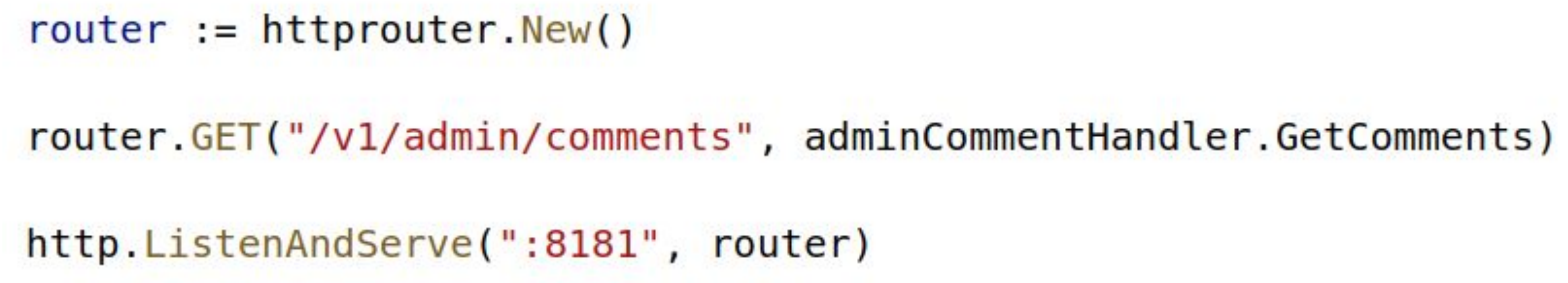
CommentRepostory, CommentService and AdminCommentHandler



```go
commentRepo := repository.NewCommentGormRepo(dbconn)
commentSrv := service.NewCommentService(commentRepo)

adminCommentHandler := handler.NewAdminCommentHandler(commentSrv)
```

Using `AdminCommentHandler`

Register the handler with the route inside `main` function using `httprouter`



```
router := httprouter.New()

router.GET("/v1/admin/comments", adminCommentHandler.GetComments)

http.ListenAndServe(":8181", router)
```

Do not forget to import the `httprouter` library

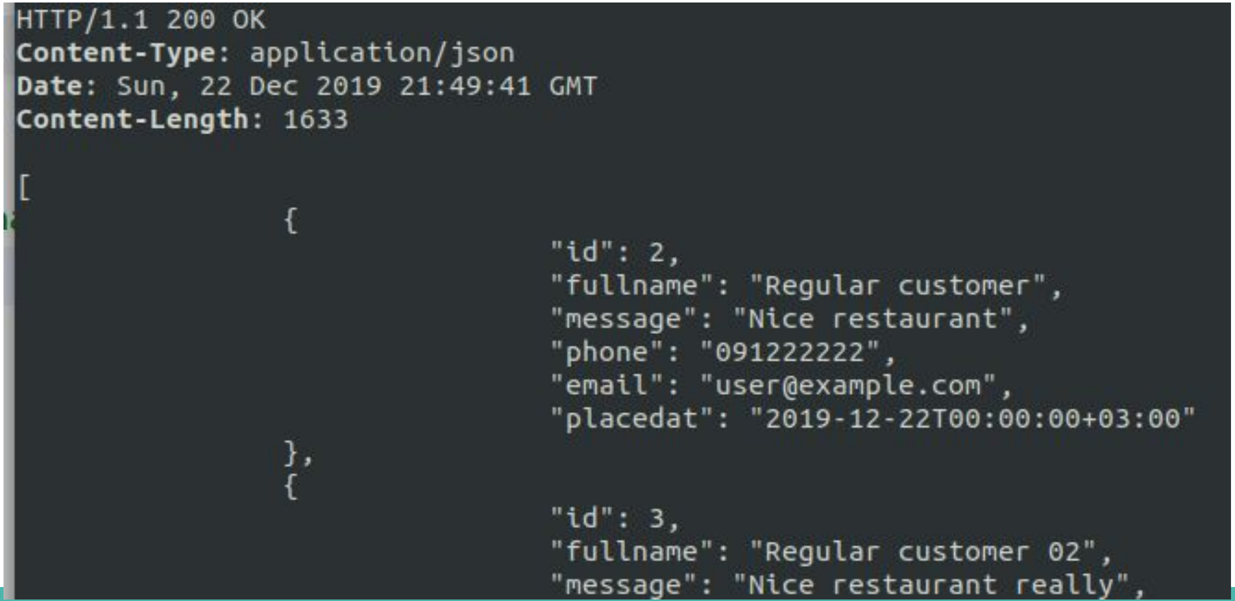
```
"github.com/julienschmidt/httprouter"
```

Testing `AdminCommentHandler`

Run the server and check the following on your terminal

```
curl -i -X GET http://localhost:8181/v1/admin/comments
```

You should see some comments displayed in **JSON** format if you have some data on the `comments` table



```
HTTP/1.1 200 OK
Content-Type: application/json
Date: Sun, 22 Dec 2019 21:49:41 GMT
Content-Length: 1633

[
                {
                                "id": 2,
                                "fullname": "Regular customer",
                                "message": "Nice restaurant",
                                "phone": "091222222",
                                "email": "user@example.com",
                                "placedat": "2019-12-22T00:00:00+03:00"
                },
                {
                                "id": 3,
                                "fullname": "Regular customer 02",
                                "message": "Nice restaurant really",
```

Create `AdminCommentHandler`

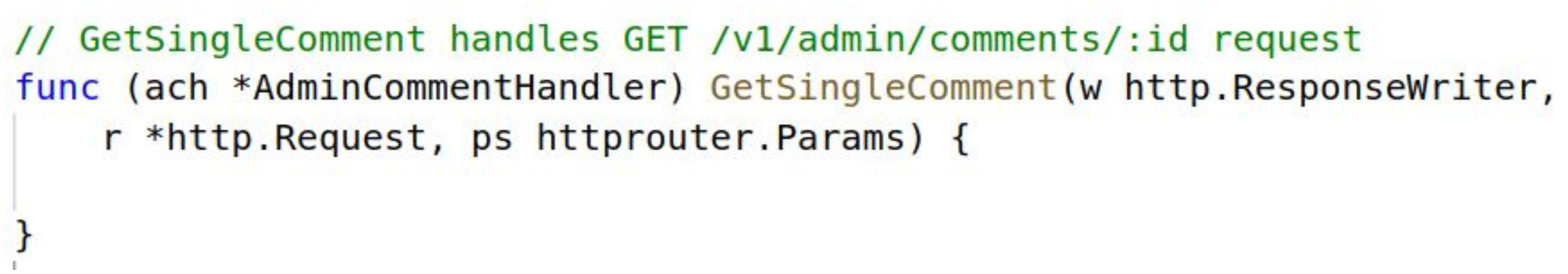
Steps for Handling `GET /v1/admin/comments/:id`

- Define the handler function
- Read the path parameter (`:id`) from the URL
- Read a with a given `id` from the persistence layer using the `CommentService`
- Convert read data to JSON format
- Write the JSON data to `ResponseWriter` object

Create `AdminCommentHandler`

Steps for Handling `GET /v1/admin/comments/:id`

Define the handler function



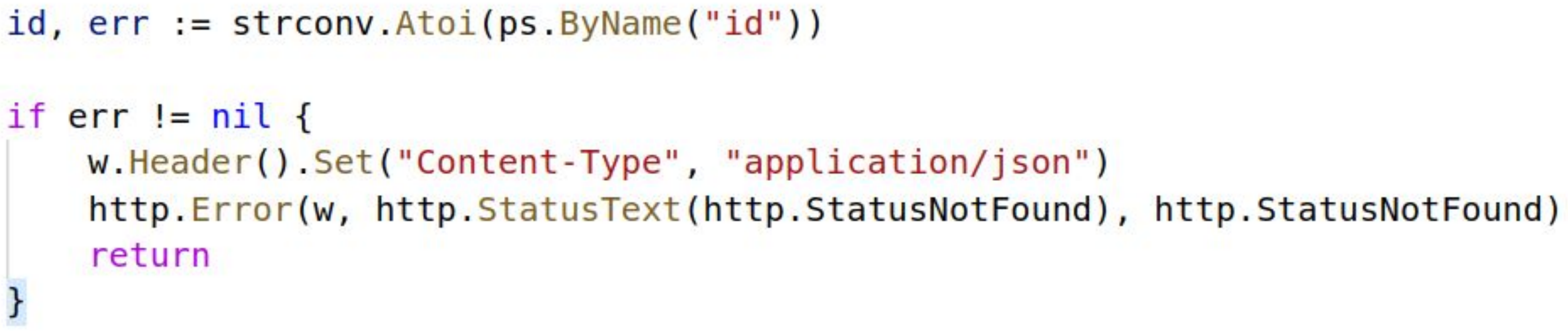
```go
// GetSingleComment handles GET /v1/admin/comments/:id request
func (ach *AdminCommentHandler) GetSingleComment(w http.ResponseWriter,
    r *http.Request, ps httprouter.Params) {

}
```

Note that in the previous case we have dropped the **ps** value now we will use it to retrieve the path parameter

Create `AdminCommentHandler`

Steps for Handling `GET /v1/admin/comments/:id`

Read the path parameter (`:id`) from the URL

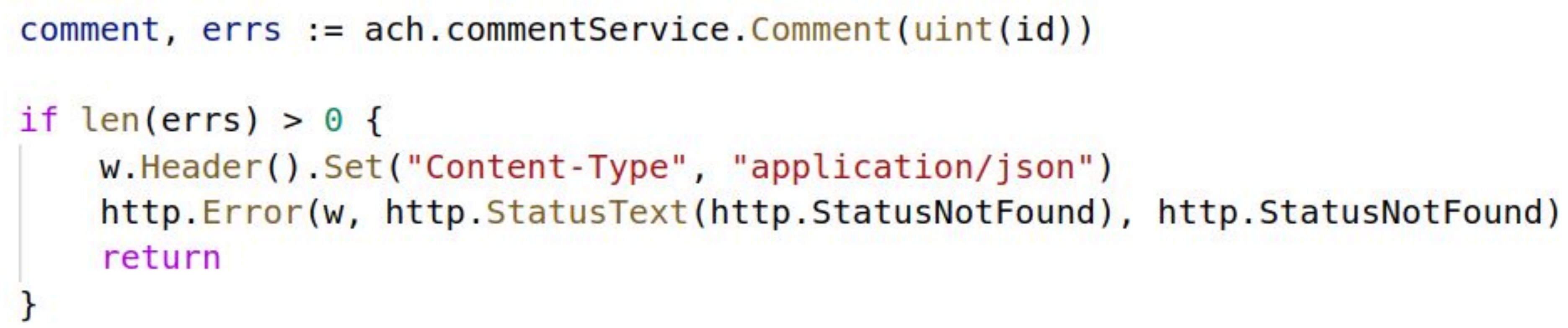


```go
id, err := strconv.Atoi(ps.ByName("id"))

if err != nil {
    w.Header().Set("Content-Type", "application/json")
    http.Error(w, http.StatusText(http.StatusNotFound), http.StatusNotFound)
    return
}
```

Create `AdminCommentHandler`

Steps for Handling **`GET /v1/admin/comments/:id`**

Read a with a given `id` from the persistence layer using the `CommentService`

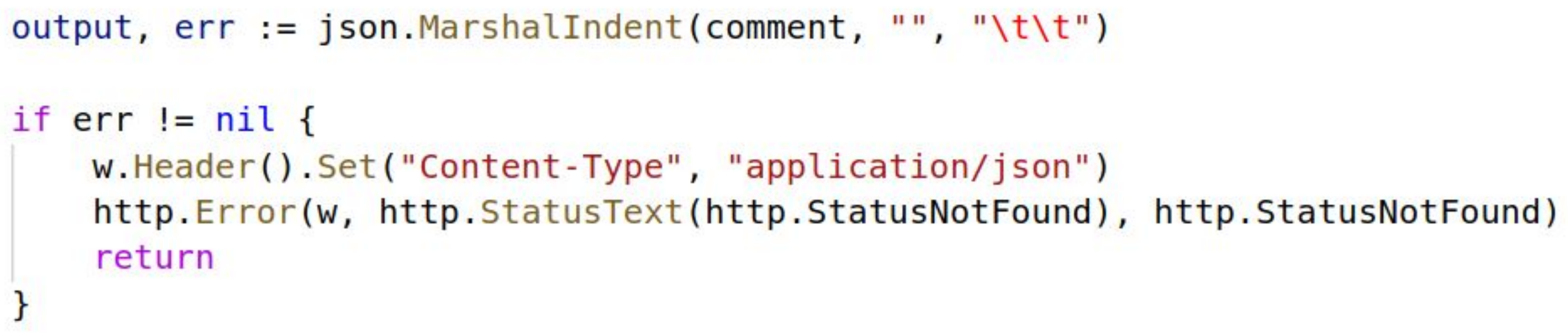


```go
comment, errs := ach.commentService.Comment(uint(id))

if len(errs) > 0 {
    w.Header().Set("Content-Type", "application/json")
    http.Error(w, http.StatusText(http.StatusNotFound), http.StatusNotFound)
    return
}
```

Create `AdminCommentHandler`

Steps for Handling **`GET /v1/admin/comments/:id`**

Convert the read data to JSON format

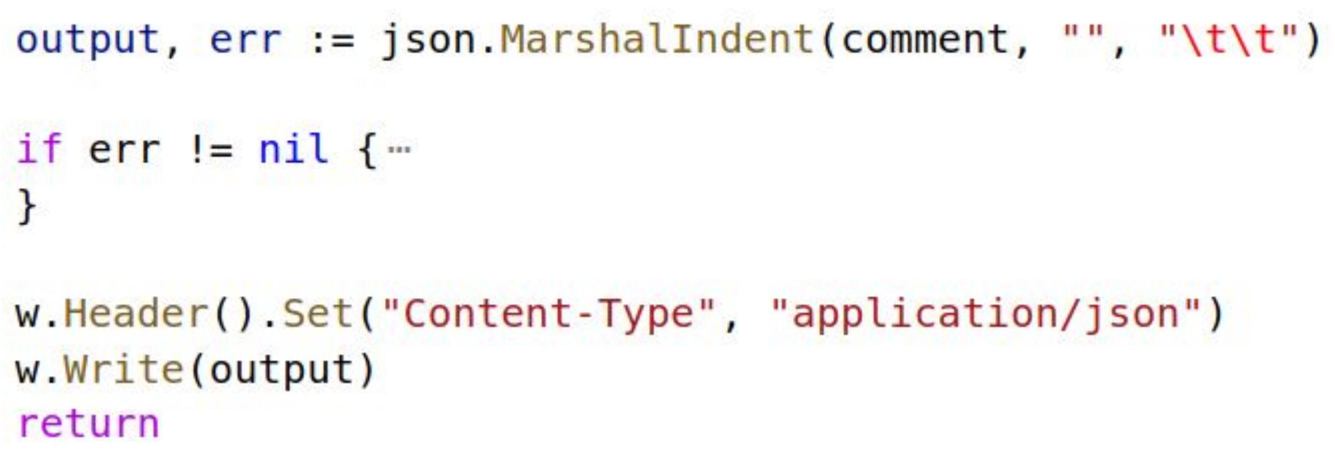


```
output, err := json.MarshalIndent(comment, "", "\t\t")

if err != nil {
    w.Header().Set("Content-Type", "application/json")
    http.Error(w, http.StatusText(http.StatusNotFound), http.StatusNotFound)
    return
}
```

Create `AdminCommentHandler`

Steps for Handling `GET /v1/admin/comments/:id`

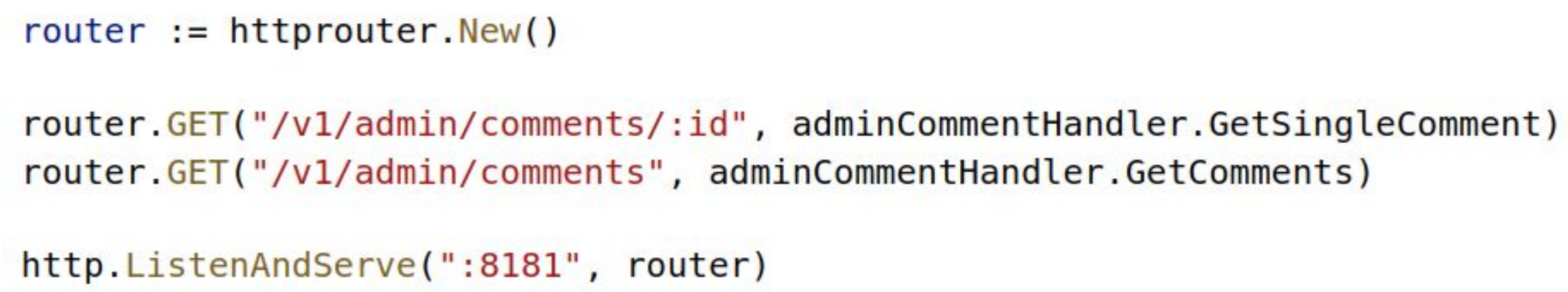
Write the JSON data to `ResponseWriter` object



```go
output, err := json.MarshalIndent(comment, "", "\t\t")

if err != nil {…
}

w.Header().Set("Content-Type", "application/json")
w.Write(output)
return
```

Using `AdminCommentHandler`

Register the handler with the route inside `main` function using `httprouter`



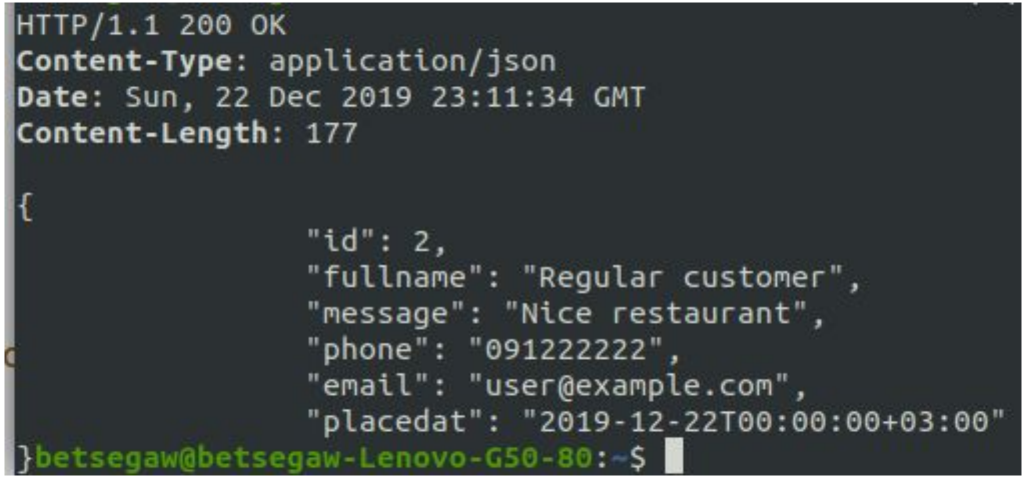
```
router := httprouter.New()

router.GET("/v1/admin/comments/:id", adminCommentHandler.GetSingleComment)
router.GET("/v1/admin/comments", adminCommentHandler.GetComments)

http.ListenAndServe(":8181", router)
```

Testing `AdminCommentHandler`

Run the server and check the following on your terminal

```
curl -i -X GET http://localhost:8181/v1/admin/comments/2
```

You should see a comment displayed in **JSON** format if you have some data on the `comments` table



```
HTTP/1.1 200 OK
Content-Type: application/json
Date: Sun, 22 Dec 2019 23:11:34 GMT
Content-Length: 177

{
                "id": 2,
                "fullname": "Regular customer",
                "message": "Nice restaurant",
                "phone": "091222222",
                "email": "user@example.com",
                "placedat": "2019-12-22T00:00:00+03:00"
}betsegaw@betsegaw-Lenovo-G50-80:~$
```

Other requests

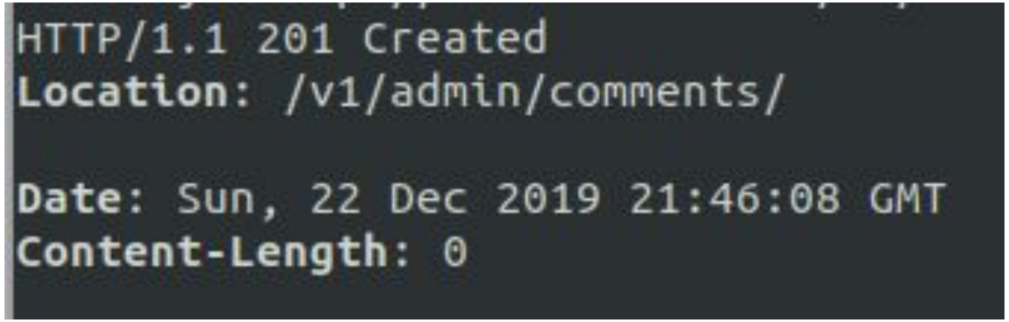
You can use similar approach to implement the remaining requests

```
POST /v1/admin/comments

PUT /v1/admin/comments/:id

DELETE /v1/admin/comments/:id
```

Test POST /v1/admin/comments

```
curl -i -X  POST -H "Content-Type: application/json" -d
'{"FullName": "Regular customer 11", "Message": "Fantastic
Restaurant", "Phone": "0911111111", "Email":
"user11@example.com", "PlacedAt":
"2019-12-22T00:00:00+03:00"}'
http://localhost:8181/v1/admin/comments
```

Response

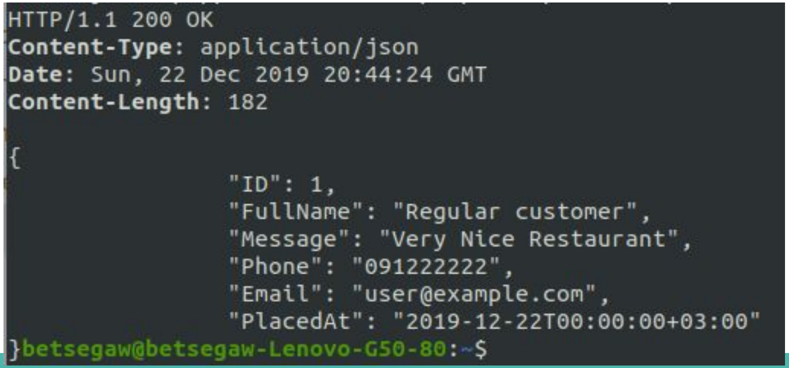


```
HTTP/1.1 201 Created
Location: /v1/admin/comments/

Date: Sun, 22 Dec 2019 21:46:08 GMT
Content-Length: 0
```

Test `PUT /v1/admin/comments/:id`

```
curl -i -X PUT -H "Content-Type: application/json" -d
'{"ID":1, "FullName": "Regular customer", "Message": "Very
Nice Restaurant", "Phone": "091222222", "Email":
"user@example.com", "PlacedAt":
"2019-12-22T00:00:00+03:00"}'
http://localhost:8181/v1/admin/comments/1
```

Response

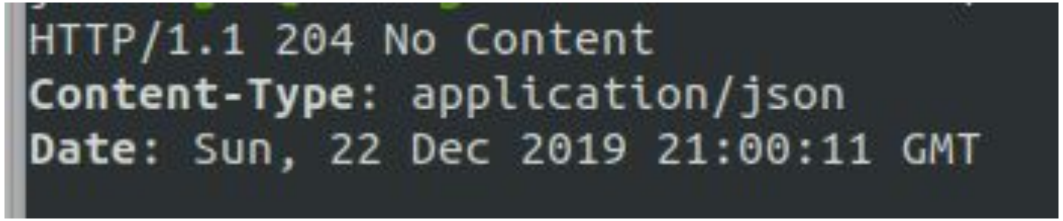


```
HTTP/1.1 200 OK
Content-Type: application/json
Date: Sun, 22 Dec 2019 20:44:24 GMT
Content-Length: 182

{
                "ID": 1,
                "FullName": "Regular customer",
                "Message": "Very Nice Restaurant",
                "Phone": "091222222",
                "Email": "user@example.com",
                "PlacedAt": "2019-12-22T00:00:00+03:00"
}betsegaw@betsegaw-Lenovo-G50-80:~$
```

Test `DELETE /v1/admin/comments/:id`

```
curl -i -X DELETE http://localhost:8181/v1/admin/comments/1
```

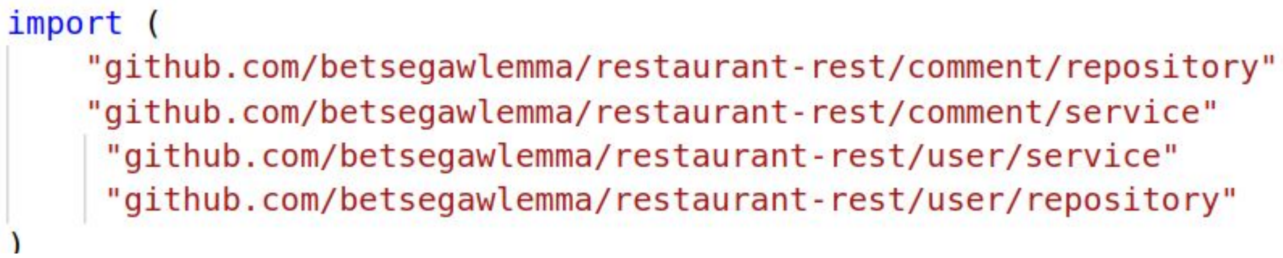
Response



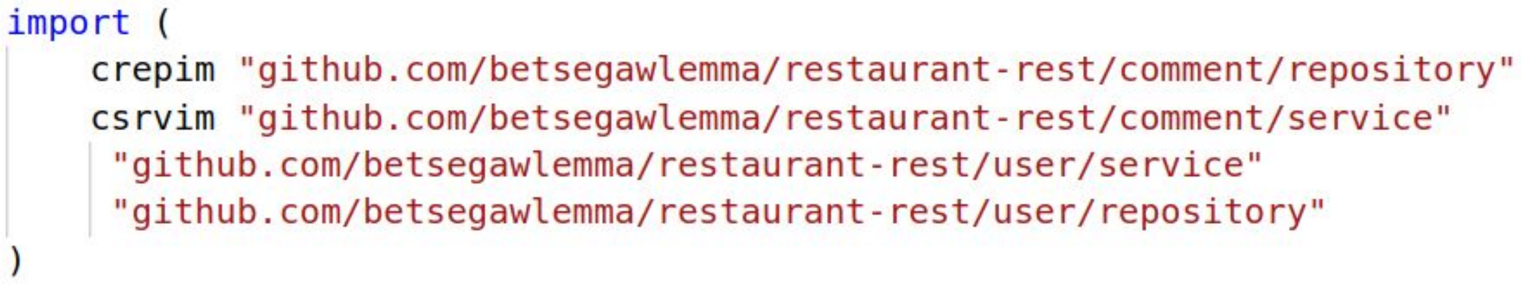
```
HTTP/1.1 204 No Content
Content-Type: application/json
Date: Sun, 22 Dec 2019 21:00:11 GMT
```

Avoiding Import Conflict

If you end up with conflicting imports like the one shown, one way of handling them is to give a different name for some of the imports as shown in the bottom. Then you can use the new name



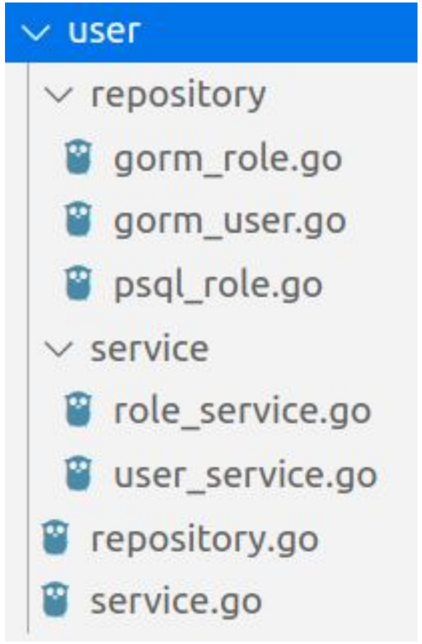
```go
import (
	"github.com/betsegawlemma/restaurant-rest/comment/repository"
	"github.com/betsegawlemma/restaurant-rest/comment/service"
	 "github.com/betsegawlemma/restaurant-rest/user/service"
	 "github.com/betsegawlemma/restaurant-rest/user/repository"
)
```



```go
import (
	crepim "github.com/betsegawlemma/restaurant-rest/comment/repository"
	csrvim "github.com/betsegawlemma/restaurant-rest/comment/service"
	 "github.com/betsegawlemma/restaurant-rest/user/service"
	 "github.com/betsegawlemma/restaurant-rest/user/repository"
)
```

Assignment-01

Implement the RESTful API for `User` feature



```
user
  repository
    gorm_role.go
    gorm_user.go
    psql_role.go
  service
    role_service.go
    user_service.go
  repository.go
  service.go
```

Assignment-02

Write a Go client for REST APIs such as `https://fixer.io/documentation`

Steps (you may need to have API key to make a request)

- Create a web form to collect some input that you will use when you make the API request
- Create a struct type with fields matching the information you want to use when a response arrives for your request
- Unmarshal the JSON response to you request to your struct
- Display the result on another web page