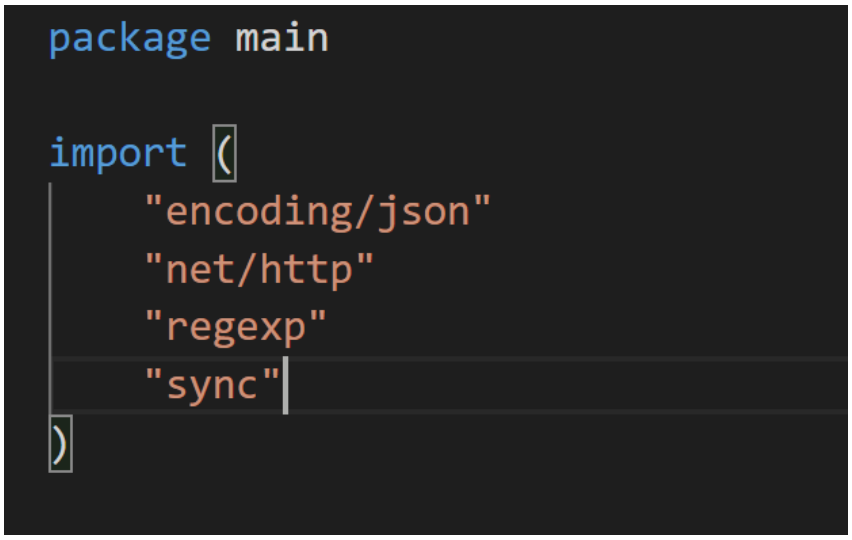
Documentation work

Name : Charu Anant Rajput
Reg.no: 19BCE1644
University : VIT Chennai

The task was to make a backend API using the Get and Post functions wherein the GET and POST methods have been used for harnessing the aforementioned functionalities.

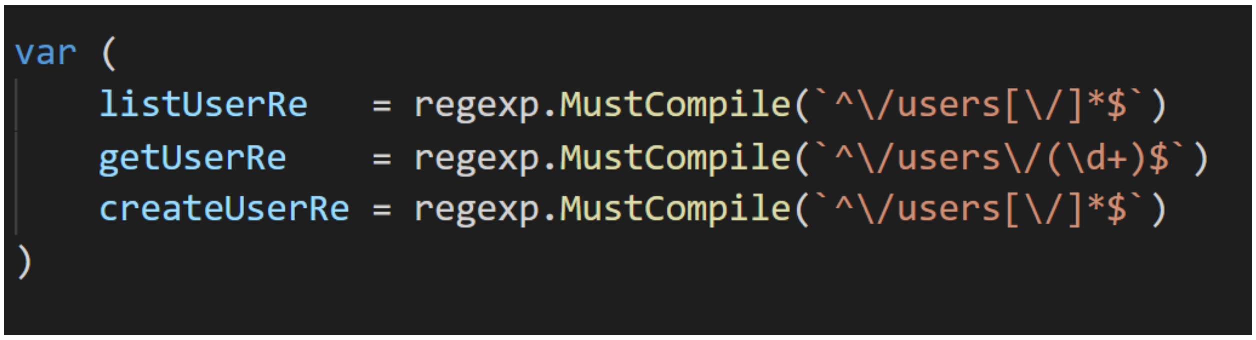
Also, the constraints were put for using the standard library packages.

(i) Importing the libraries :



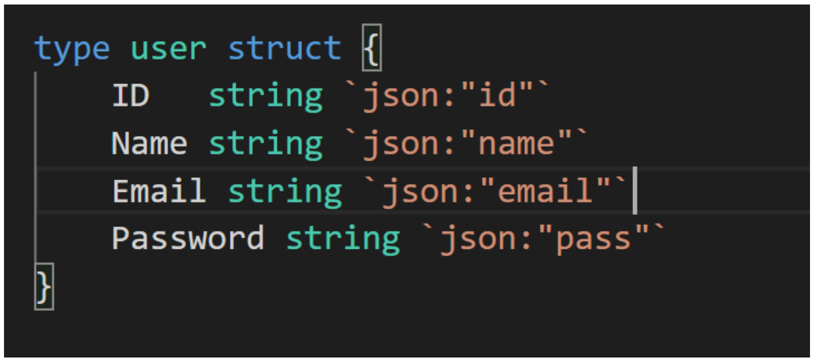
```go
package main

import (
    "encoding/json"
    "net/http"
    "regexp"
    "sync"
)
```

(ii) Defining the regex operations for each of the functionality



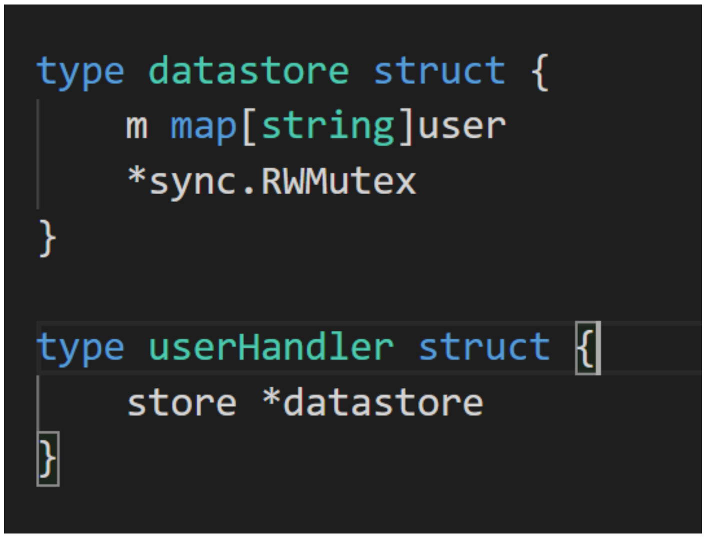
```go
var (
    listUserRe   = regexp.MustCompile(`^\/users[\/]*$`)
    getUserRe    = regexp.MustCompile(`^\/users\/(\d+)$`)
    createUserRe = regexp.MustCompile(`^\/users[\/]*$`)
)
```

(iii) Defining the struct of the user with the attributes:



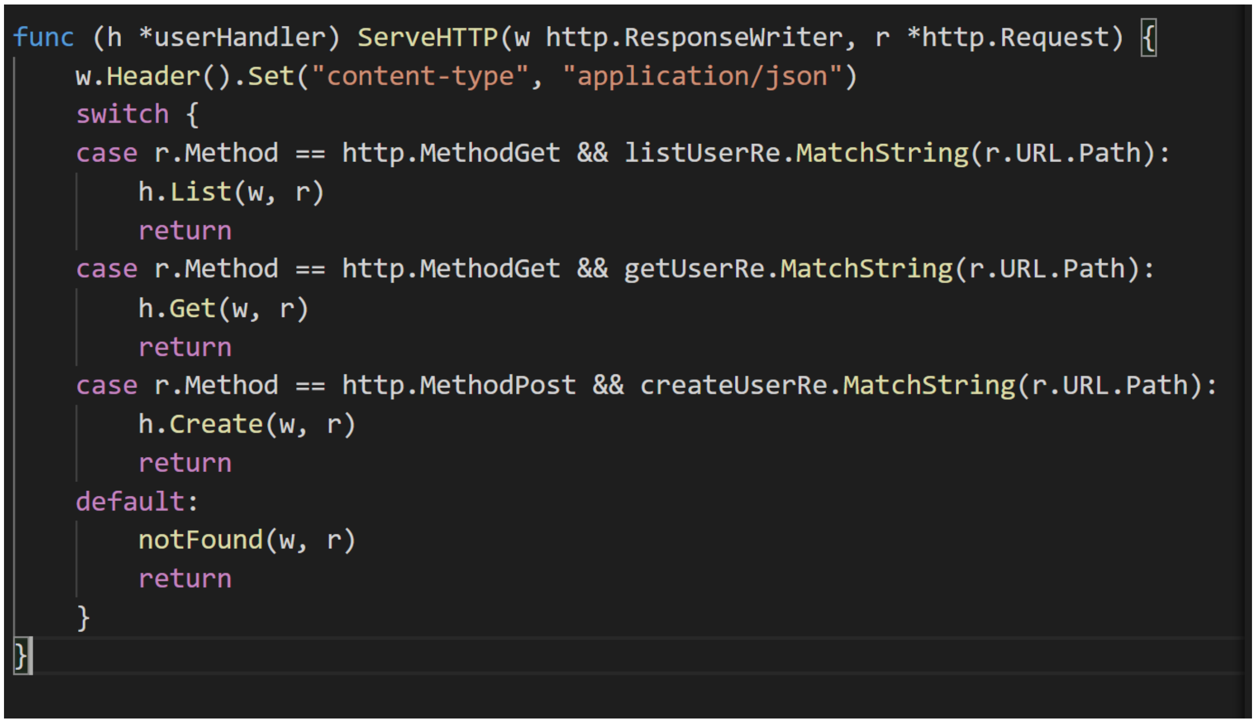
```go
type user struct {
    ID   string `json:"id"`
    Name string `json:"name"`
    Email string `json:"email"`
    Password string `json:"pass"`
}
```

(iv) Now defining the in-memory datastore for handling the records of the users and posts and also making the basing the userHandler onto the in-memory datastore:



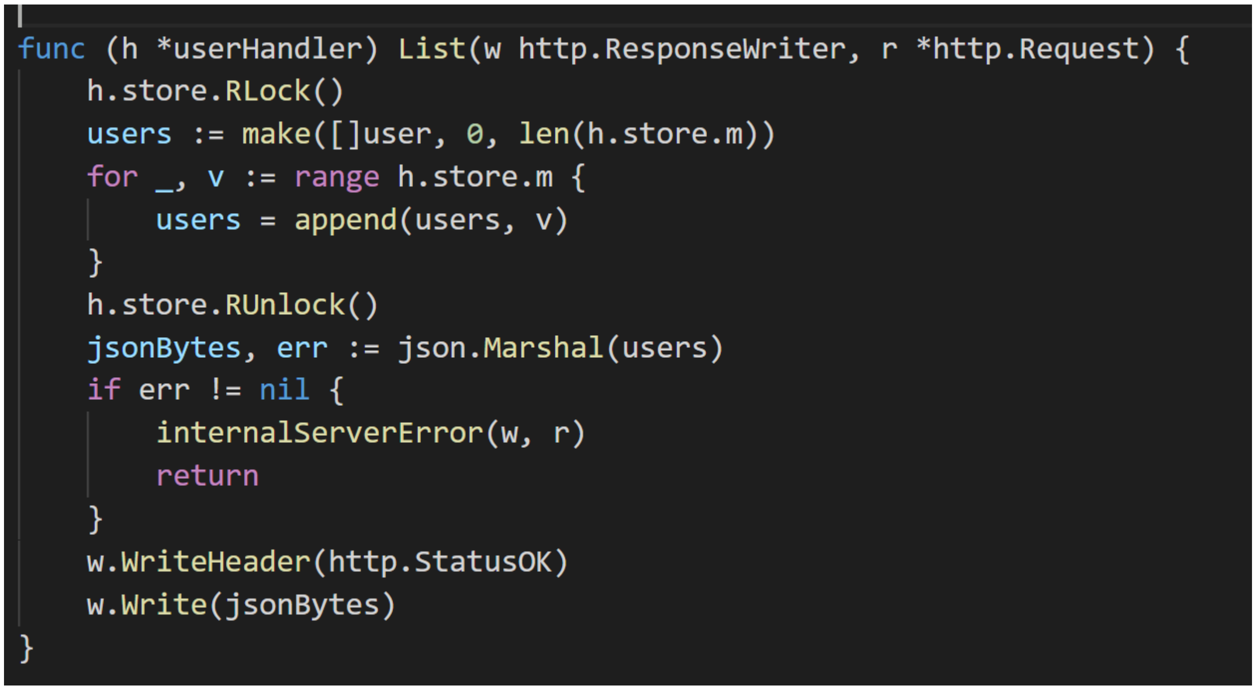
```go
type datastore struct {
    m map[string]user
    *sync.RWMutex
}

type userHandler struct {
    store *datastore
}
```

(v) Now defining the ServerHTTP for seeing the regex and selecting the methods to run accordingly as follows :



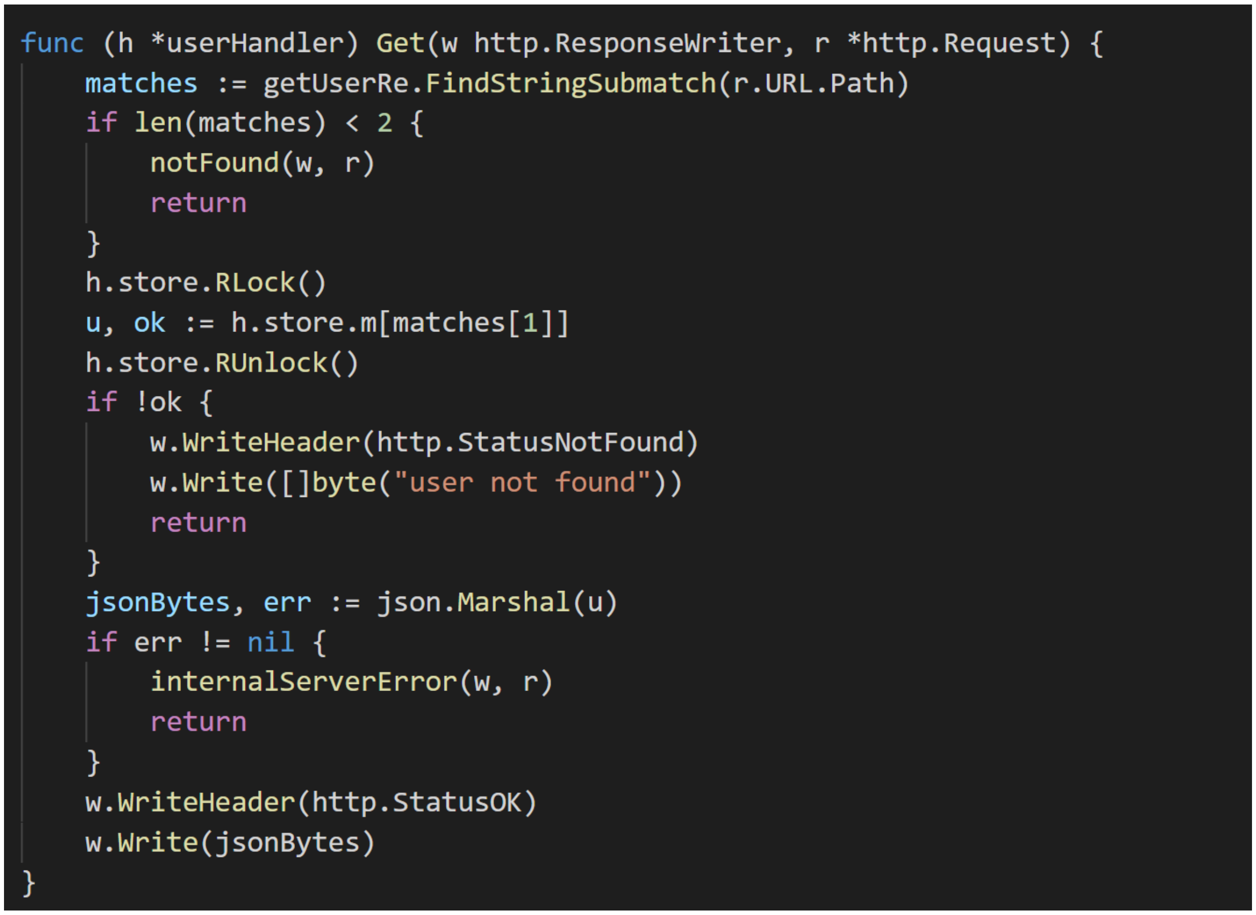
```go
func (h *userHandler) ServeHTTP(w http.ResponseWriter, r *http.Request) {
    w.Header().Set("content-type", "application/json")
    switch {
    case r.Method == http.MethodGet && listUserRe.MatchString(r.URL.Path):
        h.List(w, r)
        return
    case r.Method == http.MethodGet && getUserRe.MatchString(r.URL.Path):
        h.Get(w, r)
        return
    case r.Method == http.MethodPost && createUserRe.MatchString(r.URL.Path):
        h.Create(w, r)
        return
    default:
        notFound(w, r)
        return
    }
}
```

(vi) Making the List method for listing all the available user records:



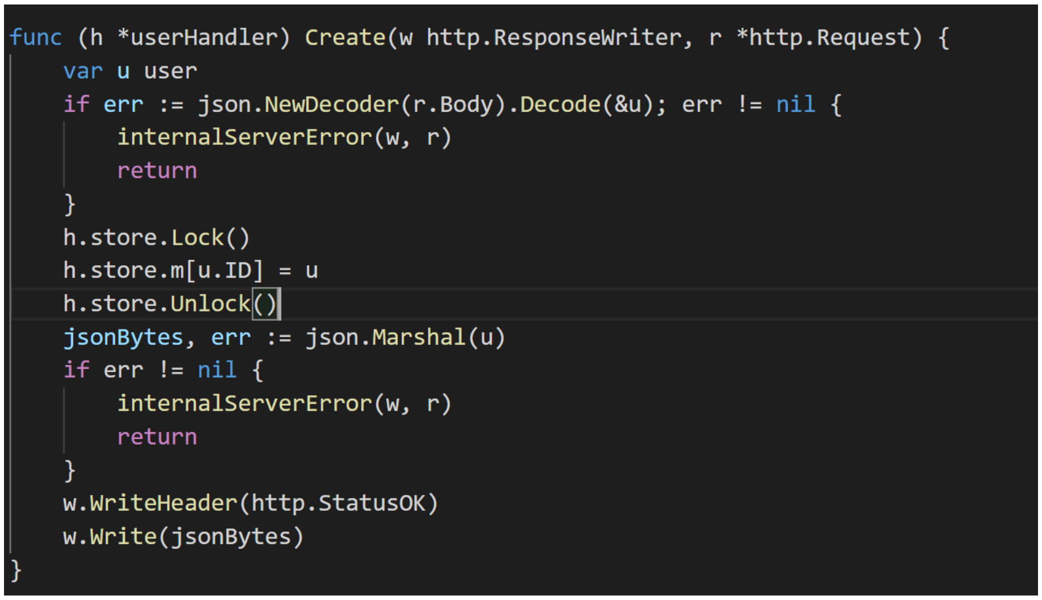
```go
func (h *userHandler) List(w http.ResponseWriter, r *http.Request) {
    h.store.RLock()
    users := make([]user, 0, len(h.store.m))
    for _, v := range h.store.m {
        users = append(users, v)
    }
    h.store.RUnlock()
    jsonBytes, err := json.Marshal(users)
    if err != nil {
        internalServerError(w, r)
        return
    }
    w.WriteHeader(http.StatusOK)
    w.Write(jsonBytes)
}
```

(vii) Making the Get method for retrieving records according to the ID:



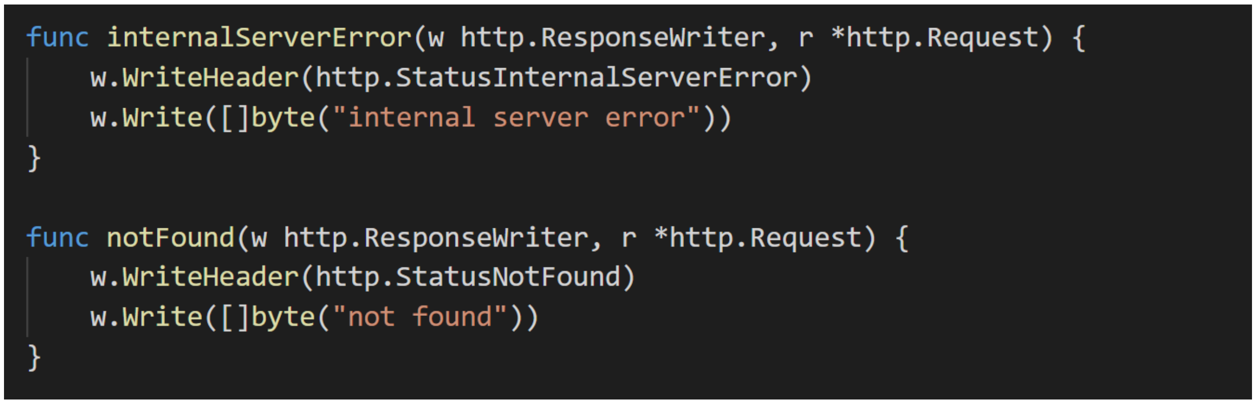
```go
func (h *userHandler) Get(w http.ResponseWriter, r *http.Request) {
    matches := getUserRe.FindStringSubmatch(r.URL.Path)
    if len(matches) < 2 {
        notFound(w, r)
        return
    }
    h.store.RLock()
    u, ok := h.store.m[matches[1]]
    h.store.RUnlock()
    if !ok {
        w.WriteHeader(http.StatusNotFound)
        w.Write([]byte("user not found"))
        return
    }
    jsonBytes, err := json.Marshal(u)
    if err != nil {
        internalServerError(w, r)
        return
    }
    w.WriteHeader(http.StatusOK)
    w.Write(jsonBytes)
}
```

(viii) Making the Create method:



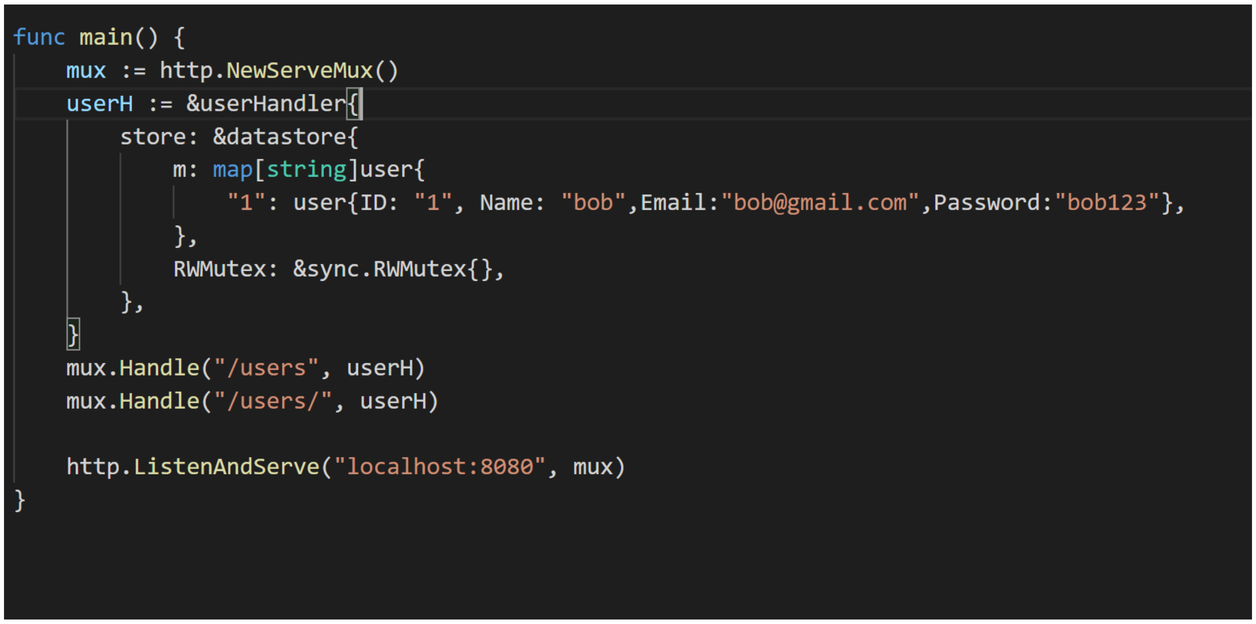
```go
func (h *userHandler) Create(w http.ResponseWriter, r *http.Request) {
    var u user
    if err := json.NewDecoder(r.Body).Decode(&u); err != nil {
        internalServerError(w, r)
        return
    }
    h.store.Lock()
    h.store.m[u.ID] = u
    h.store.Unlock()
    jsonBytes, err := json.Marshal(u)
    if err != nil {
        internalServerError(w, r)
        return
    }
    w.WriteHeader(http.StatusOK)
    w.Write(jsonBytes)
}
```

(ix) Defining the methods for handling the exceptions:



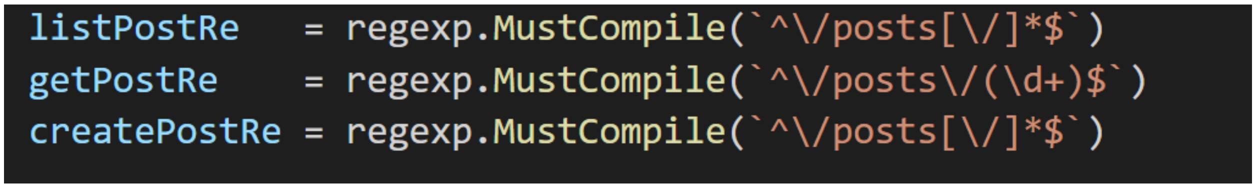
```go
func internalServerError(w http.ResponseWriter, r *http.Request) {
    w.WriteHeader(http.StatusInternalServerError)
    w.Write([]byte("internal server error"))
}

func notFound(w http.ResponseWriter, r *http.Request) {
    w.WriteHeader(http.StatusNotFound)
    w.Write([]byte("not found"))
}
```

(x) Making the main function and calling the functions. The entries have been done manually. The function is as follows:

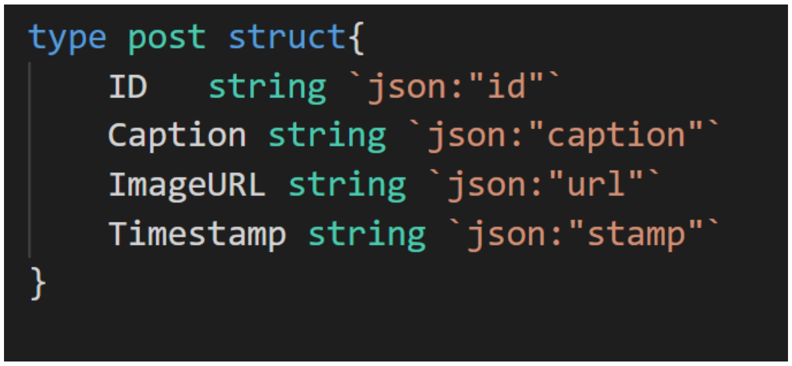


```go
func main() {
    mux := http.NewServeMux()
    userH := &userHandler{
        store: &datastore{
            m: map[string]user{
                "1": user{ID: "1", Name: "bob",Email:"bob@gmail.com",Password:"bob123"},
            },
            RWMutex: &sync.RWMutex{},
        },
    }
    mux.Handle("/users", userH)
    mux.Handle("/users/", userH)

    http.ListenAndServe("localhost:8080", mux)
}
```

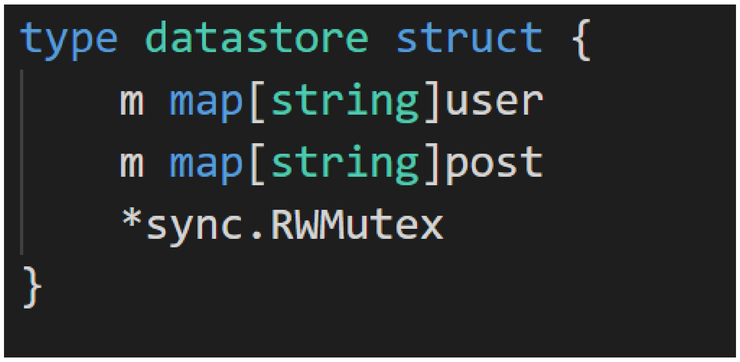
Similar workflow and methods have been defined for the Posts as follows and in the following workflow: But on combining this part with the users section, the code could not run.



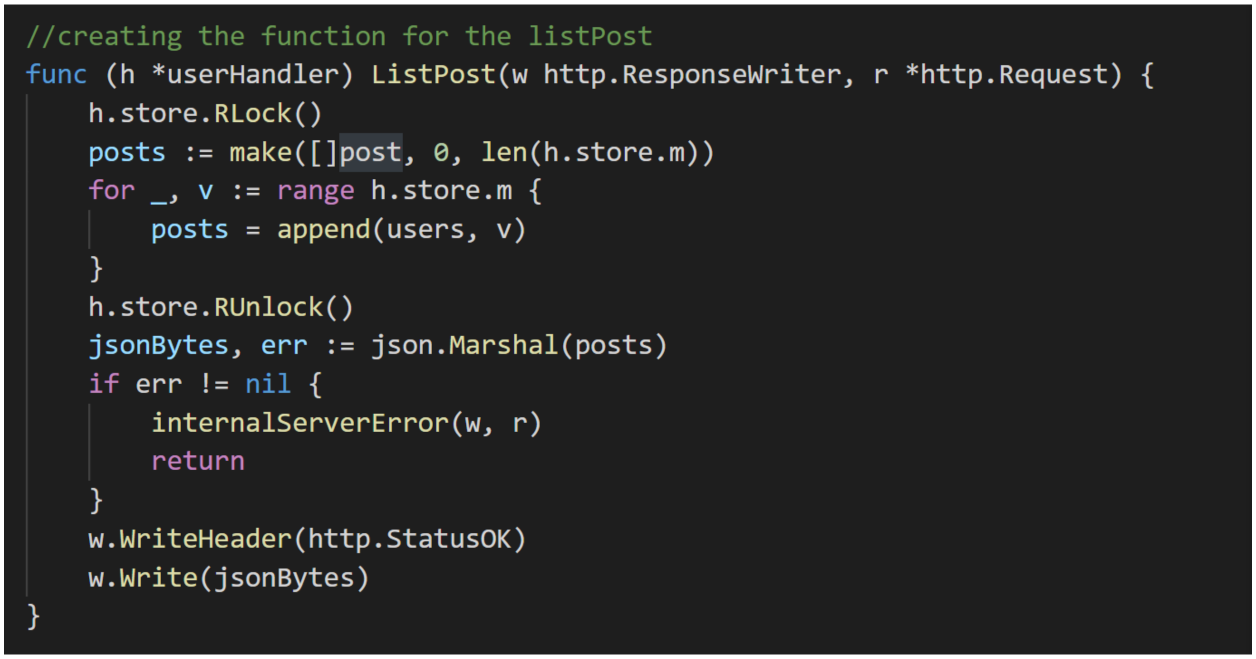
```
listPostRe   = regexp.MustCompile(`^\/posts[\/]*$`)
getPostRe    = regexp.MustCompile(`^\/posts\/(\d+)$`)
createPostRe = regexp.MustCompile(`^\/posts[\/]*$`)
```



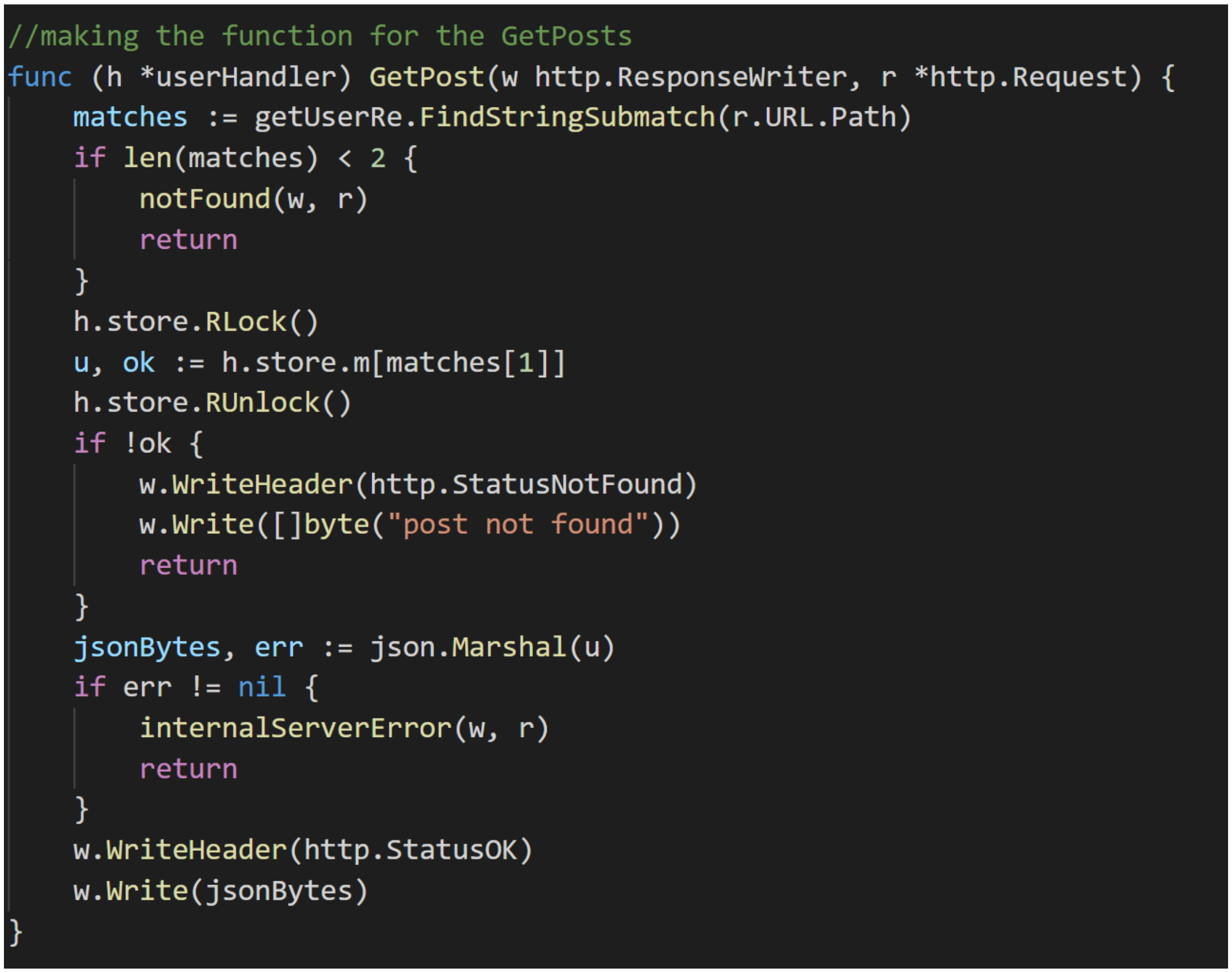
```
type post struct{
    ID   string `json:"id"`
    Caption string `json:"caption"`
    ImageURL string `json:"url"`
    Timestamp string `json:"stamp"`
}
```



```
type datastore struct {
    m map[string]user
    m map[string]post
    *sync.RWMutex
}
```

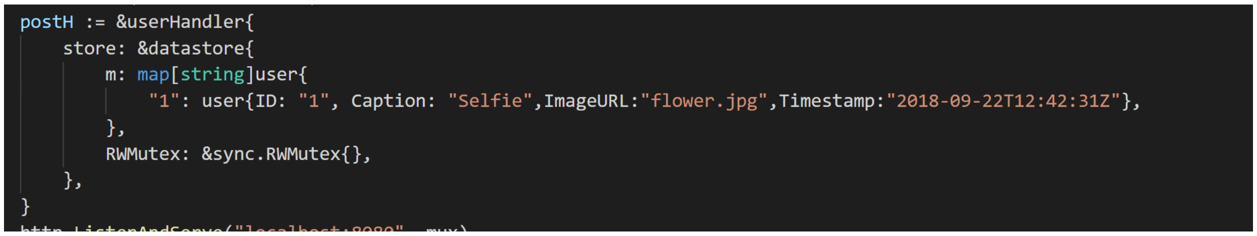


```go
//creating the function for the listPost
func (h *userHandler) ListPost(w http.ResponseWriter, r *http.Request) {
	h.store.RLock()
	posts := make([]post, 0, len(h.store.m))
	for _, v := range h.store.m {
		posts = append(users, v)
	}
	h.store.RUnlock()
	jsonBytes, err := json.Marshal(posts)
	if err != nil {
		internalServerError(w, r)
		return
	}
	w.WriteHeader(http.StatusOK)
	w.Write(jsonBytes)
}
```

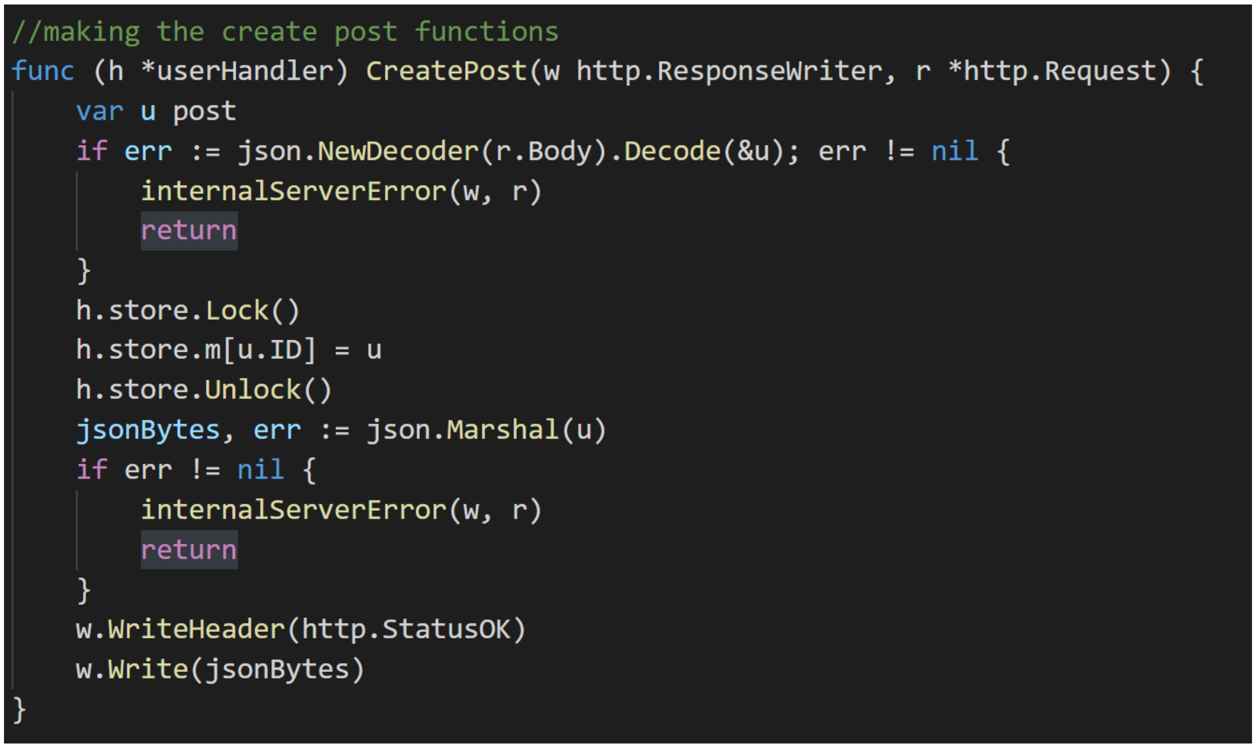


```go
//making the function for the GetPosts
func (h *userHandler) GetPost(w http.ResponseWriter, r *http.Request) {
	matches := getUserRe.FindStringSubmatch(r.URL.Path)
	if len(matches) < 2 {
		notFound(w, r)
		return
	}
	h.store.RLock()
	u, ok := h.store.m[matches[1]]
	h.store.RUnlock()
	if !ok {
		w.WriteHeader(http.StatusNotFound)
		w.Write([]byte("post not found"))
		return
	}
	jsonBytes, err := json.Marshal(u)
	if err != nil {
		internalServerError(w, r)
		return
	}
	w.WriteHeader(http.StatusOK)
	w.Write(jsonBytes)
}
```

Data entering:

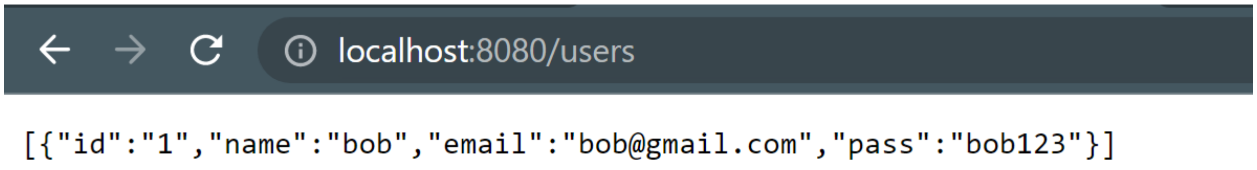


```
postH := &userHandler{
    store: &datastore{
        m: map[string]user{
            "1": user{ID: "1", Caption: "Selfie",ImageURL:"flower.jpg",Timestamp:"2018-09-22T12:42:31Z"},
        },
        RWMutex: &sync.RWMutex{},
    },
}
http.ListenAndServe("localhost:8080", mux)
```



```
//making the create post functions
func (h *userHandler) CreatePost(w http.ResponseWriter, r *http.Request) {
    var u post
    if err := json.NewDecoder(r.Body).Decode(&u); err != nil {
        internalServerError(w, r)
        return
    }
    h.store.Lock()
    h.store.m[u.ID] = u
    h.store.Unlock()
    jsonBytes, err := json.Marshal(u)
    if err != nil {
        internalServerError(w, r)
        return
    }
    w.WriteHeader(http.StatusOK)
    w.Write(jsonBytes)
}
```

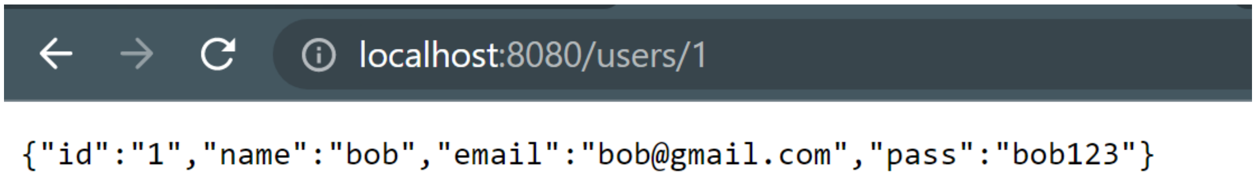
Output:
[For the users part]
Listing function :



localhost:8080/users

```
[{"id":"1","name":"bob","email":"bob@gmail.com","pass":"bob123"}]
```

Retrieving part:


localhost:8080/users/1

```
{"id":"1","name":"bob","email":"bob@gmail.com","pass":"bob123"}
```