Appointy - Technical Task | 19BCE1462

Statement:

To make a MongoDB/Go based Instagram Backend API with five basic endpoints, i.e. Create a User, Get User with ID, Create Post, Get Post using ID and Show all Posts from User.

test.go:

```
package main

import (
     "fmt"
     "net/http"
     "sync"
     "time"
     "context"
     "encoding/json"
     "github.com/gorilla/mux"
     "go.mongodb.org/mongo-driver/bson"
     "go.mongodb.org/mongo-driver/bson/primitive"
     "go.mongodb.org/mongo-driver/mongo"
     "go.mongodb.org/mongo-driver/mongo/options"
     "crypto/sha256"
)

var lock sync.Mutex
var client *mongo.Client
//json and binary json annotations
type Users struct {
     ID        primitive.ObjectID `json:"_id,omitempty"
bson:"_id,omitempty"`
     Name      string             `json:"name,omitempty"
bson:"name,omitempty"`
     Email     string             `json:"email,omitempty"
bson:"email,omitempty"`
```

```
     Password  string              `json: "password,omitempty"
bson:"password,omitempty"`
}

type Posts struct {
     ID        primitive.ObjectID `json:"_id,omitempty"
bson:"_id,omitempty"`
     Caption   string              `json:"caption,omitempty"
bson:"caption,omitempty"`
     ImageURL  string              `json:"imageurl,omitempty"
bson:"imageurl,omitempty"`
     Timestamp string              `json:"timestamp,omitempty"
bson:"timestamp,omitempty"`
}

//first endpoint - create a user (unique id generated as a
primitive
func CreateUserEndpoint(response http.ResponseWriter, request
*http.Request) {
     response.Header().Set("content-type", "application/json")
     var user Users
     _ = json.NewDecoder(request.Body).Decode(&user)
     collection :=
client.Database("InstagramDB").Collection("Users")
     ctx, _ := context.WithTimeout(context.Background(),
5*time.Second)
//sha256 encryption to generate cipher used in collection
     pass := []byte(user.Password)
     hash := sha256.Sum256(pass)
     user.Password = string(hash[:])
     result, _ := collection.InsertOne(ctx, user)
     json.NewEncoder(response).Encode(result)
     lock.Lock()
     defer lock.Unlock()
}
 //second endpoint - create a post with timestamp and metadata
 func CreatePostEndpoint(response http.ResponseWriter, request
*http.Request) {
     response.Header().Set("content-type", "application/json")
     var post Posts
     _ = json.NewDecoder(request.Body).Decode(&post)
```

```
     collection :=
client.Database("InstagramDB").Collection("Posts")
     ctx, _ := context.WithTimeout(context.Background(),
5*time.Second)
     dt := time.Now()
     post.Timestamp = dt.Format("01-02-2006 15:04:05")
     result, _ := collection.InsertOne(ctx, post)
     json.NewEncoder(response).Encode(result)
     lock.Lock()
     defer lock.Unlock()
}
//get user by id
func GetUserEndpoint(response http.ResponseWriter, request
*http.Request) {
     response.Header().Set("content-type", "application/json")
     params := mux.Vars(request)
     id, _ := primitive.ObjectIDFromHex(params["id"])
     var user Users
     collection :=
client.Database("InstagramDB").Collection("Users")
     ctx, _ := context.WithTimeout(context.Background(),
30*time.Second)
     err := collection.FindOne(ctx, Users{ID: id}).Decode(&user)
     if err != nil {
     response.WriteHeader(http.StatusInternalServerError)
     response.Write([]byte(`{ "message": "` + err.Error() + `"
}`))
     return
     }
     json.NewEncoder(response).Encode(user)
}
 //get post by id
func GetPostEndpoint(response http.ResponseWriter, request
*http.Request) {
     response.Header().Set("content-type", "application/json")
     params := mux.Vars(request)
     id, _ := primitive.ObjectIDFromHex(params["id"])
     var post Posts
     collection :=
client.Database("InstagramDB").Collection("Posts")
```

```
     ctx, _ := context.WithTimeout(context.Background(),
30*time.Second)
     err := collection.FindOne(ctx, Posts{ID: id}).Decode(&post)
     if err != nil {
     response.WriteHeader(http.StatusInternalServerError)
     response.Write([]byte(`{ "message": "` + err.Error() + `"
}`))
     return
     }
     json.NewEncoder(response).Encode(post)
}
 //get all posts from a particular user by id

func GetAllPostsEndpoint(response http.ResponseWriter, request
*http.Request) {
     response.Header().Set("content-type", "application/json")
     var posts []Posts
     collection :=
client.Database("InstagramDB").Collection("Posts")
     ctx, _ := context.WithTimeout(context.Background(),
30*time.Second)
     cursor, err := collection.Find(ctx, bson.M{})
     if err != nil {
     response.WriteHeader(http.StatusInternalServerError)
     response.Write([]byte(`{ "message": "` + err.Error() + `"
}`))
     return
     }

     defer cursor.Close(ctx)
     for cursor.Next(ctx) {
     var post Posts
     cursor.Decode(&post)
     posts = append(posts, post)
     }
     if err := cursor.Err(); err != nil {
     response.WriteHeader(http.StatusInternalServerError)
     response.Write([]byte(`{ "message": "` + err.Error() + `"
}`))
     return
     }
```

```
	json.NewEncoder(response).Encode(posts)
}

func main() {

	fmt.Println("Loading backend... Success!/n")
	fmt.Println("API Ready to use... System time: ")
	fmt.Println(time.Now())

	ctx, _ := context.WithTimeout(context.Background(),
10*time.Second)
	clientOptions :=
options.Client().ApplyURI("mongodb://localhost:27017")
	client, _ = mongo.Connect(ctx, clientOptions)

	router := mux.NewRouter()
	router.HandleFunc("/users",
CreateUserEndpoint).Methods("POST")
	router.HandleFunc("/users/{id}",
GetUserEndpoint).Methods("GET")
	router.HandleFunc("/posts",
CreatePostEndpoint).Methods("POST")
	router.HandleFunc("/posts/{id}",
GetPostEndpoint).Methods("GET")
	router.HandleFunc("/posts/users/{id}",
GetAllPostsEndpoint).Methods("GET")
	http.ListenAndServe(":12345", router)
}
```

External Dependencies used - mongo-driver

Endpoints:

/api/posts

creates a post using data from the POST request's body
/api/posts/<id>

fetches post details for the given id

/api/posts/users/<id>

fetches all posts from given user id
/api/users

creates a user and encrypts the password using ciphers before storing
/api/users/<id>

fetches user details of given id

Testing:

This API has been successfully tested using Postman with POST and GET requests