Task | Instagram Backend API

I have successfully developed an Instagram Backend API using Go Language.

This API uses JSON as informed in the question.

Create New User

This API is capable of creating a user using a POST request, in which the name, email and password will be taken in JSON format in the body of the POST request. This information along with the ID will be stored in a MongoDB database.

```go
func newUser(w http.ResponseWriter, r *http.Request) {
    if r.Method == "POST" {
        lock.Lock()
        w.Header().Set("Content-Type", "application/json")
        var new user
        userCount++
        err := json.NewDecoder(r.Body).Decode(&new)
        if err != nil {
            fmt.Print(err)
        }
        new.Id = strconv.Itoa(userCount)
        fmt.Println(new)
        ct := encrypt([]byte(new.Password), new.Name)
        new.Password = string(ct)
        insertResult, err := userCollection.InsertOne(context.TODO(), new)
        if err != nil {
            log.Fatal(err)
        }
        lock.Unlock()
        fmt.Println("User created successfully, ID: ", new.Id)
        json.NewEncoder(w).Encode(insertResult)
    }
}
```

Search User

Next, searching a user using the ID. This is done using a GET request and the ID is present in the URL. This returns all the information of the user in JSON format.

```go
func getUser(w http.ResponseWriter, r *http.Request) {
    if r.Method == "GET" {
        lock.Lock()
        w.Header().Set("Content-Type", "application/json")
        ids := r.URL.Path
        id := path.Base(ids)
        var result primitive.M
        err := userCollection.FindOne(context.TODO(), bson.D{{"id",
id}}).Decode(&result)
        if err != nil {
            fmt.Println(err)
        }
        lock.Unlock()
        json.NewEncoder(w).Encode(result)

    }
}
```

Create New Post

Similar to creating a new user, a new post is created using the POST request and the caption, image URL and User ID are sent in the request body. This information along with post id and time is stored in a separate collection in MongoDB.

```
func newPost(w http.ResponseWriter, r *http.Request) {
    if r.Method == "POST" {
        lock.Lock()
        w.Header().Set("Content-Type", "application/json")
        var new post
        postCount++
        err := json.NewDecoder(r.Body).Decode(&new)
        if err != nil {
            fmt.Print(err)
        }
        t := time.Now()
        new.Time = t.String()
        new.Id = strconv.Itoa(postCount)
        fmt.Println(new)
        insertResult, err := postCollection.InsertOne(context.TODO(), new)
        if err != nil {
            log.Fatal(err)
        }
        lock.Unlock()
        fmt.Println("Post inserted successfully, ID:", new.Id)
        json.NewEncoder(w).Encode(insertResult)
    }
}
```

Search Post

Similar to searching a user, a post is searched using a GET request and the ID is present in the URL. All the information regarding that post is returned.

```go
func getPost(w http.ResponseWriter, r *http.Request) {
    if r.Method == "GET" {
        lock.Lock()
        w.Header().Set("Content-Type", "application/json")
        ids := r.URL.Path
        id := path.Base(ids)
        var result primitive.M
        err := postCollection.FindOne(context.TODO(), bson.D{{"id", 
id}}).Decode(&result)
        if err != nil {
            fmt.Println(err)
        }
        lock.Unlock()
        json.NewEncoder(w).Encode(result)
    }
}
```

All Posts

All posts of a particular user are returned using a GET request. The user-id is present in the URL.

```go
func allPost(w http.ResponseWriter, r *http.Request) {
    if r.Method == "GET" {
        lock.Lock()
        w.Header().Set("Content-Type", "application/json")
        ids := r.URL.Path
        l := strings.Split(ids, "/")
        id := l[3]
        page, _ := strconv.Atoi(l[4])
        limit := 2
        i := 0
        cur, err := postCollection.Find(context.TODO(), bson.D{{"uid", id}})
        if err != nil {
            fmt.Println(err)
        }
        var results []*post
        for cur.Next(context.TODO()) {
            var elem post
            err := cur.Decode(&elem)
            if err != nil {
                log.Fatal(err)
            }
            if i == (page-1)*limit || i == (page-1)*limit+1 {
                results = append(results, &elem)
            }
            i++
        }
        if err := cur.Err(); err != nil {
            log.Fatal(err)
        }
        cur.Close(context.TODO())
        lock.Unlock()
        json.NewEncoder(w).Encode(results)
    }
}
```

Encryption

The password of each user is encrypted using AES before storing in the data base. The plaintext is the password in the POST request body and the name present in the POST request is used as key phrase. Thus, each user has their own unique key for encryption. This stops anyone from reverse engineering the passwords.

```
func createHash(key string) string {
    hasher := md5.New()
    hasher.Write([]byte(key))
    return hex.EncodeToString(hasher.Sum(nil))
}

func encrypt(data []byte, passphrase string) []byte {
    block, _ := aes.NewCipher([]byte(createHash(passphrase)))
    gcm, err := cipher.NewGCM(block)
    if err != nil {
        panic(err.Error())
    }
    nonce := make([]byte, gcm.NonceSize())
    if _, err = io.ReadFull(rand.Reader, nonce); err != nil {
        panic(err.Error())
    }
    ciphertext := gcm.Seal(nonce, nonce, data, nil)
    return ciphertext
}

func decrypt(data []byte, passphrase string) []byte {
    key := []byte(createHash(passphrase))
    block, err := aes.NewCipher(key)
    if err != nil {
        panic(err.Error())
    }
    gcm, err := cipher.NewGCM(block)
    if err != nil {
        panic(err.Error())
    }
    nonceSize := gcm.NonceSize()
    nonce, ciphertext := data[:nonceSize], data[nonceSize:]
    plaintext, err := gcm.Open(nil, nonce, ciphertext, nil)
    if err != nil {
        panic(err.Error())
    }
    return plaintext
}
```

Thread Safety

The server is made thread safe so that multiple requests cannot be sent at once using sync.Mutex. The Lock and Unlock methods are implemented in each and every function.

```
var lock sync.Mutex
```

Pagination

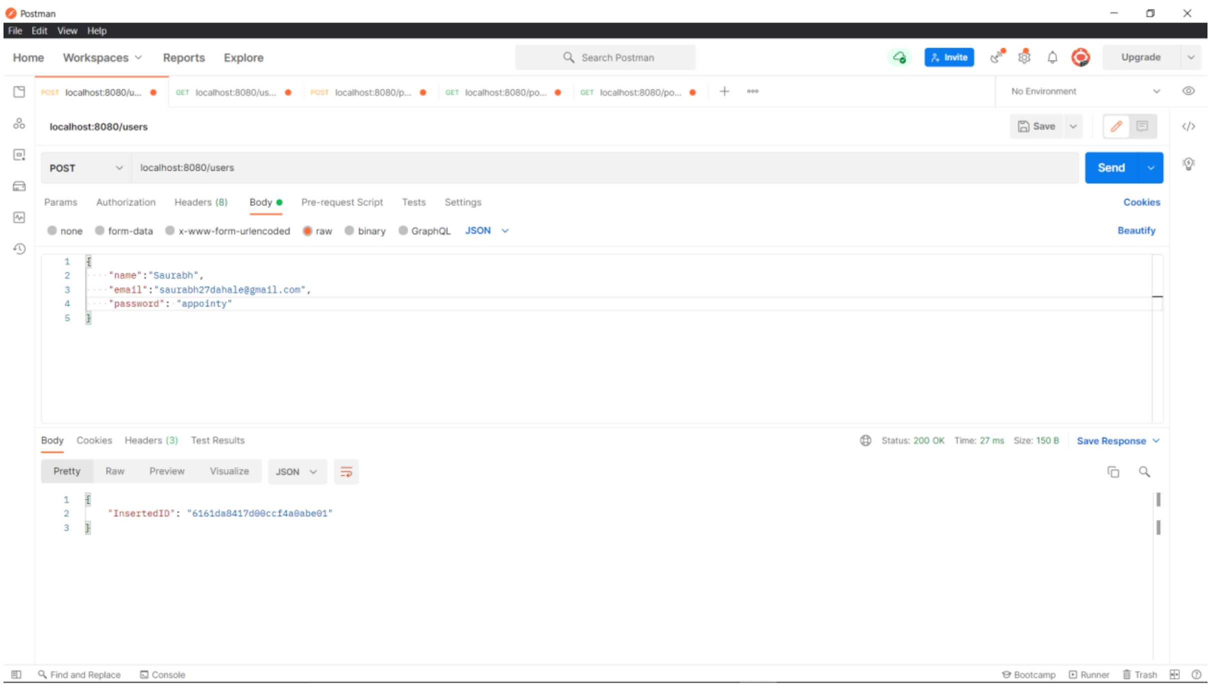
Pagination was required in the request for all the posts by a particular user. This was done by taking the page number in the URL and a simple if condition using the page number and the limit of each page.

```
if i == (page-1)*limit || i == (page-1)*limit+1 {
                results = append(results, &elem)
            }
```

Unit Tests

The unit tests were done using POSTMAN and each and every request was satisfied. Encryption, thread safety and pagination were implemented successfully.

New User:


Postman
File Edit View Help
Home Workspaces Reports Explore
Search Postman
Invite
Upgrade
No Environment
localhost:8080/users
Save
Send
Params Authorization Headers (8) Body Pre-request Script Tests Settings
Cookies
none form-data x-www-form-urlencoded raw binary GraphQL JSON
Beautify
Body Cookies Headers (3) Test Results
Status: 200 OK Time: 27 ms Size: 150 B
Save Response
Pretty Raw Preview Visualize JSON
Find and Replace Console
Bootcamp Runner Trash

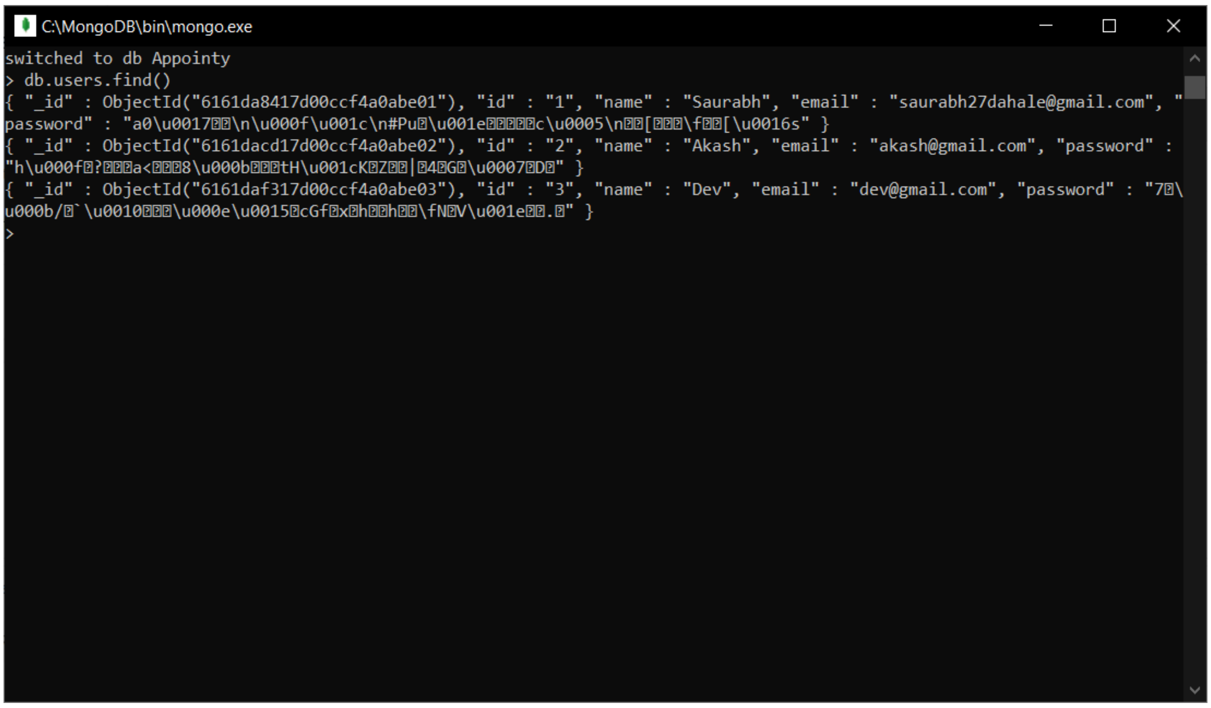
POST localhost:8080/users

```
{
    "name":"Saurabh",
    "email":"saurabh27dahale@gmail.com",
    "password": "appointy"
}
```

```
{
    "InsertedID": "6161da8417d00ccf4a0abe01"
}
```

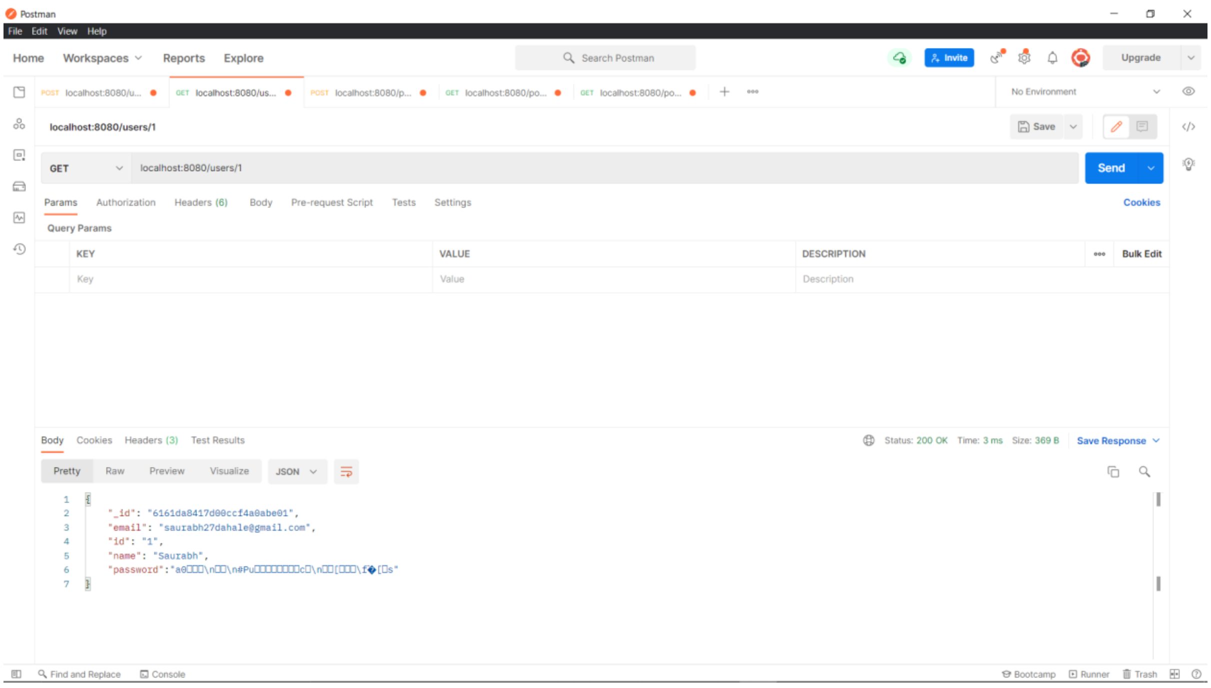
Mongodb:

New user was created along with 2 others.



```
switched to db Appointy
> db.users.find()
{ "_id" : ObjectId("6161da8417d00ccf4a0abe01"), "id" : "1", "name" : "Saurabh", "email" : "saurabh27dahale@gmail.com", "
password" : "a0\u0017��\n\u000f\u001c\n#Pu�\u001e�����c\u0005\n��[���\f��[\u0016s" }
{ "_id" : ObjectId("6161dacd17d00ccf4a0abe02"), "id" : "2", "name" : "Akash", "email" : "akash@gmail.com", "password" :
"h\u000f�?���a<���8\u000b���tH\u001cK�Z��|�4�G�\u0007�D�" }
{ "_id" : ObjectId("6161daf317d00ccf4a0abe03"), "id" : "3", "name" : "Dev", "email" : "dev@gmail.com", "password" : "7�\
u000b/�`\u0010���\u000e\u0015�cGf�x�h��h��\fN�V\u001e��.�" }
>
```

Search User:



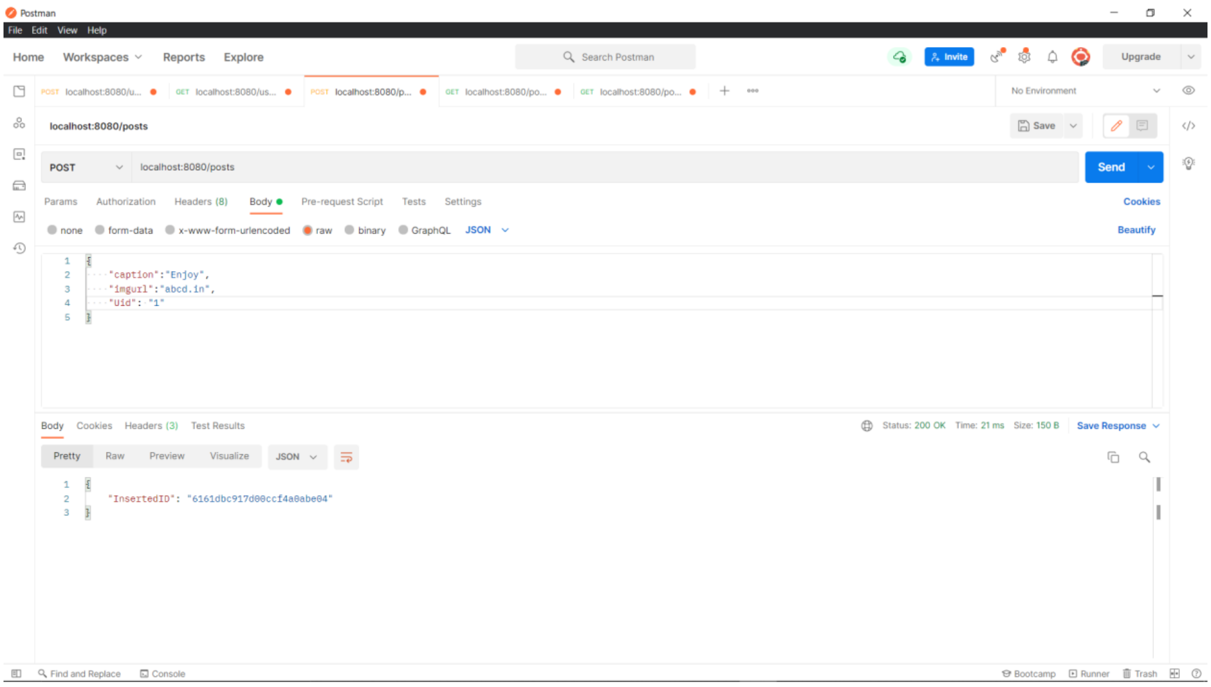
GET localhost:8080/users/1

Status: 200 OK Time: 3 ms Size: 369 B

```
{
    "_id": "6161da8417d00ccf4a0abe01",
    "email": "saurabh27dahale@gmail.com",
    "id": "1",
    "name": "Saurabh",
    "password":"a0□□□\n□□\n#Pu□□□□□□□□c□\n□□[□□□\f�[□s"
}
```

User 1 successfully found as seen in the body.

New Post



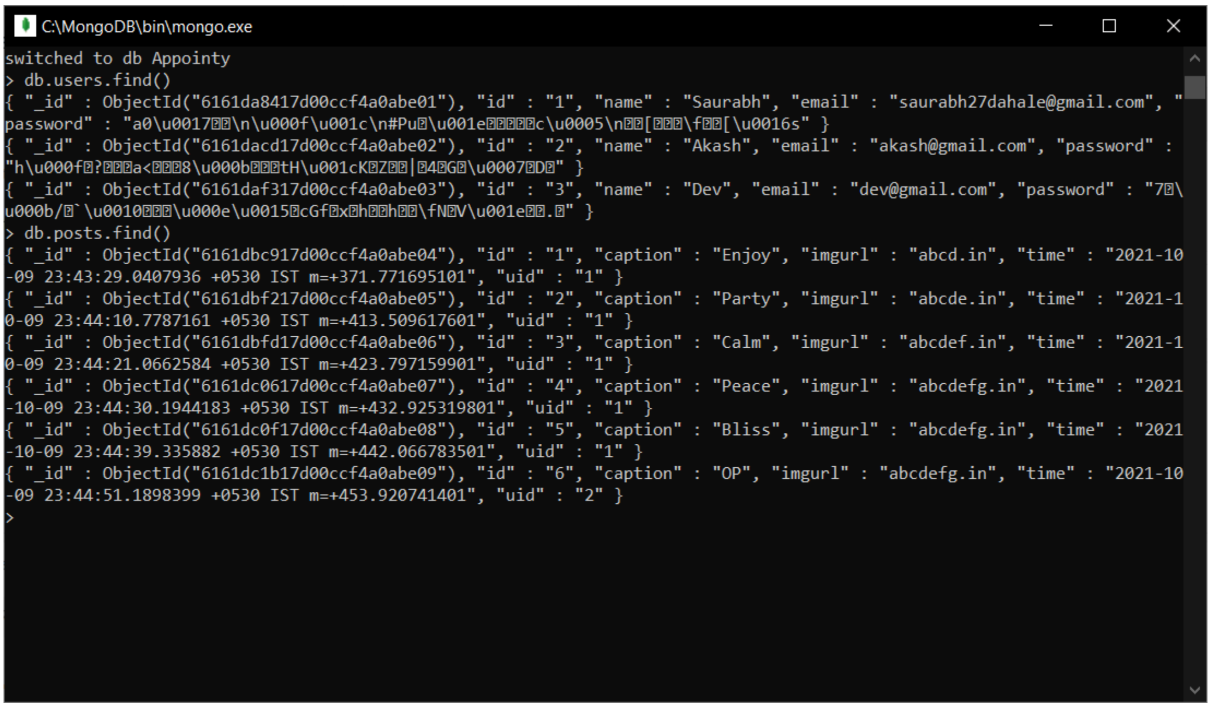
POST localhost:8080/posts

```
{
    "caption":"Enjoy",
    "imgurl":"abcd.in",
    "Uid": "1"
}
```

Status: 200 OK Time: 21 ms Size: 150 B

```
{
    "InsertedID": "6161dbc917d00ccf4a0abe04"
}
```

Mongodb

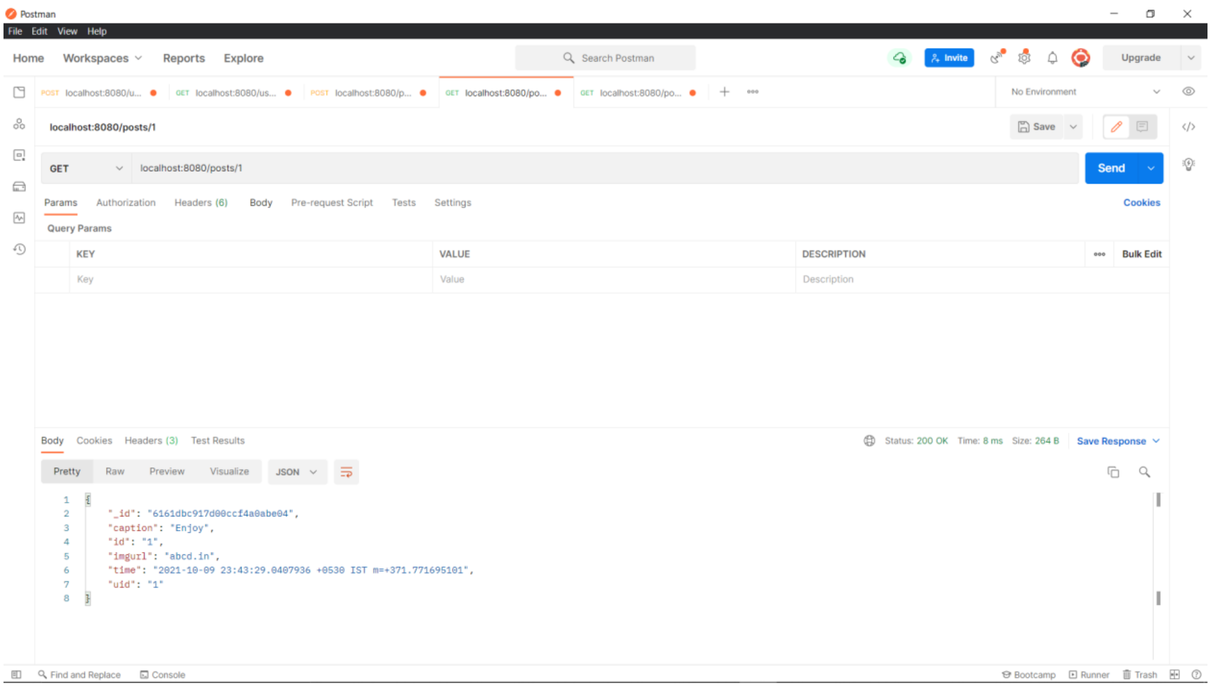


C:\MongoDB\bin\mongo.exe

```
switched to db Appointy
> db.users.find()
{ "_id" : ObjectId("6161da8417d00ccf4a0abe01"), "id" : "1", "name" : "Saurabh", "email" : "saurabh27dahale@gmail.com", "
password" : "a0\u0017??\n\u000f\u001c\n#Pu?\u001e?????c\u0005\n??[???\f??[\u0016s" }
{ "_id" : ObjectId("6161dacd17d00ccf4a0abe02"), "id" : "2", "name" : "Akash", "email" : "akash@gmail.com", "password" :
"h\u000f??????a<???8\u000b???tH\u001cK?Z??|?4?G?\u0007?D?" }
{ "_id" : ObjectId("6161daf317d00ccf4a0abe03"), "id" : "3", "name" : "Dev", "email" : "dev@gmail.com", "password" : "7?\
u000b/?`\u0010???\u000e\u0015?cGf?x?h??h??\fN?V\u001e??.?" }
> db.posts.find()
{ "_id" : ObjectId("6161dbc917d00ccf4a0abe04"), "id" : "1", "caption" : "Enjoy", "imgurl" : "abcd.in", "time" : "2021-10
-09 23:43:29.0407936 +0530 IST m=+371.771695101", "uid" : "1" }
{ "_id" : ObjectId("6161dbf217d00ccf4a0abe05"), "id" : "2", "caption" : "Party", "imgurl" : "abcde.in", "time" : "2021-1
0-09 23:44:10.7787161 +0530 IST m=+413.509617601", "uid" : "1" }
{ "_id" : ObjectId("6161dbfd17d00ccf4a0abe06"), "id" : "3", "caption" : "Calm", "imgurl" : "abcdef.in", "time" : "2021-1
0-09 23:44:21.0662584 +0530 IST m=+423.797159901", "uid" : "1" }
{ "_id" : ObjectId("6161dc0617d00ccf4a0abe07"), "id" : "4", "caption" : "Peace", "imgurl" : "abcdefg.in", "time" : "2021
-10-09 23:44:30.1944183 +0530 IST m=+432.925319801", "uid" : "1" }
{ "_id" : ObjectId("6161dc0f17d00ccf4a0abe08"), "id" : "5", "caption" : "Bliss", "imgurl" : "abcdefg.in", "time" : "2021
-10-09 23:44:39.335882 +0530 IST m=+442.066783501", "uid" : "1" }
{ "_id" : ObjectId("6161dc1b17d00ccf4a0abe09"), "id" : "6", "caption" : "OP", "imgurl" : "abcdefg.in", "time" : "2021-10
-09 23:44:51.1898399 +0530 IST m=+453.920741401", "uid" : "2" }
>
```

A new post along with a few others was created.

Search Post



localhost:8080/posts/1

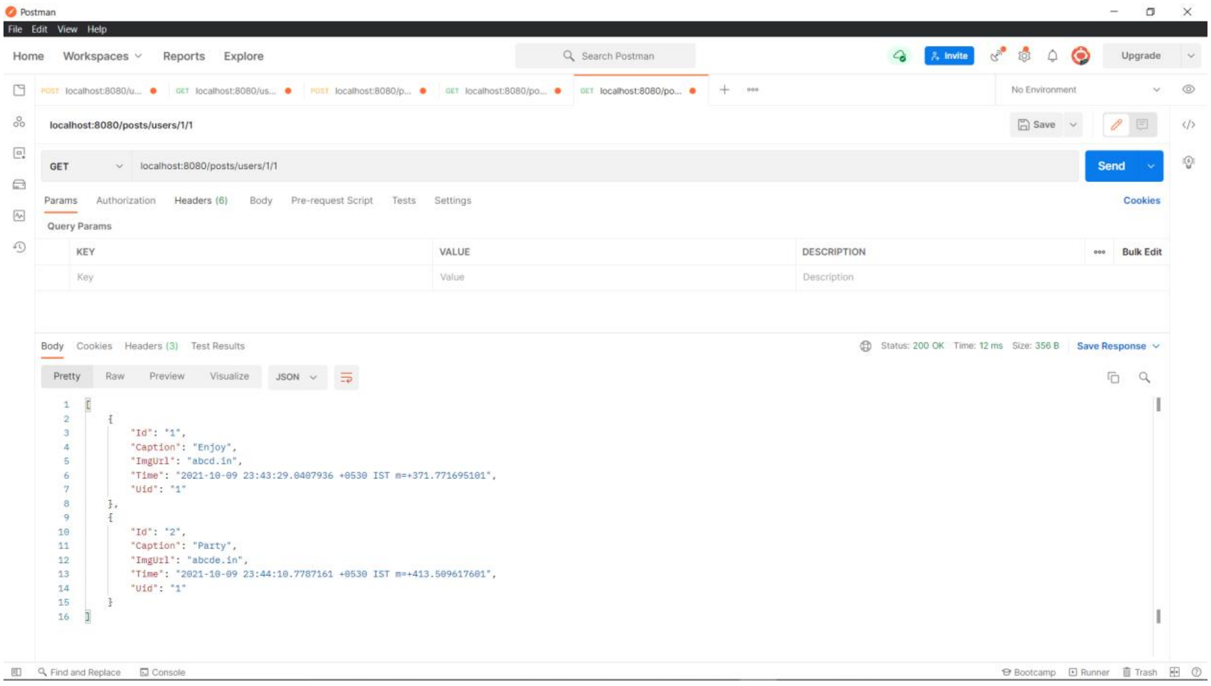
```
GET localhost:8080/posts/1
```

Status: 200 OK Time: 8 ms Size: 264 B

```
{
    "_id": "6161dbc917d00ccf4a0abe04",
    "caption": "Enjoy",
    "id": "1",
    "imgurl": "abcd.in",
    "time": "2021-10-09 23:43:29.0407936 +0530 IST m=+371.771695101",
    "uid": "1"
}
```

Post successfully found and information displayed in body.

Search All Posts



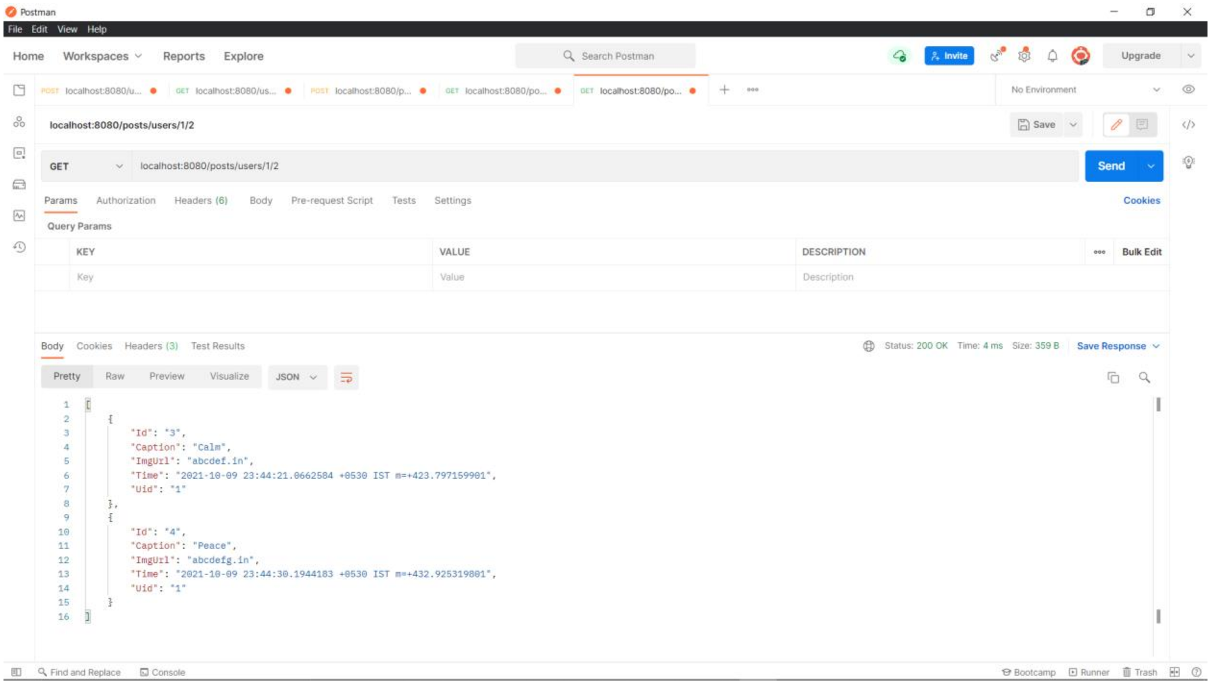
GET localhost:8080/posts/users/1/1

Status: 200 OK Time: 12 ms Size: 356 B

```
[
    {
        "Id": "1",
        "Caption": "Enjoy",
        "ImgUrl": "abcd.in",
        "Time": "2021-10-09 23:43:29.0407936 +0530 IST m=+371.771695101",
        "Uid": "1"
    },
    {
        "Id": "2",
        "Caption": "Party",
        "ImgUrl": "abcde.in",
        "Time": "2021-10-09 23:44:10.7787161 +0530 IST m=+413.509617601",
        "Uid": "1"
    }
]
```

Only 2 posts on page 1 show up as pagination limit is 2.

Page 2:



GET localhost:8080/posts/users/1/2

Status: 200 OK Time: 4 ms Size: 359 B

```
[
    {
        "Id": "3",
        "Caption": "Calm",
        "ImgUrl": "abcdef.in",
        "Time": "2021-10-09 23:44:21.0662584 +0530 IST m=+423.797159901",
        "Uid": "1"
    },
    {
        "Id": "4",
        "Caption": "Peace",
        "ImgUrl": "abcdefg.in",
        "Time": "2021-10-09 23:44:30.1944183 +0530 IST m=+432.925319801",
        "Uid": "1"
    }
]
```