Golang-Backend

Introduction

This RESTful API is designed to interact with a MongoDB database to **get/create/update/delete** user data from the db. In addition to the following features have also been implemented:

- Basic user auth using jwt

- Logging
- Follow other users (following)
  Have followers
- Get nearby users who they are following

Tech Stack Used

Database: MongoDB hosted in Atlas

Server (&client for auth): Go

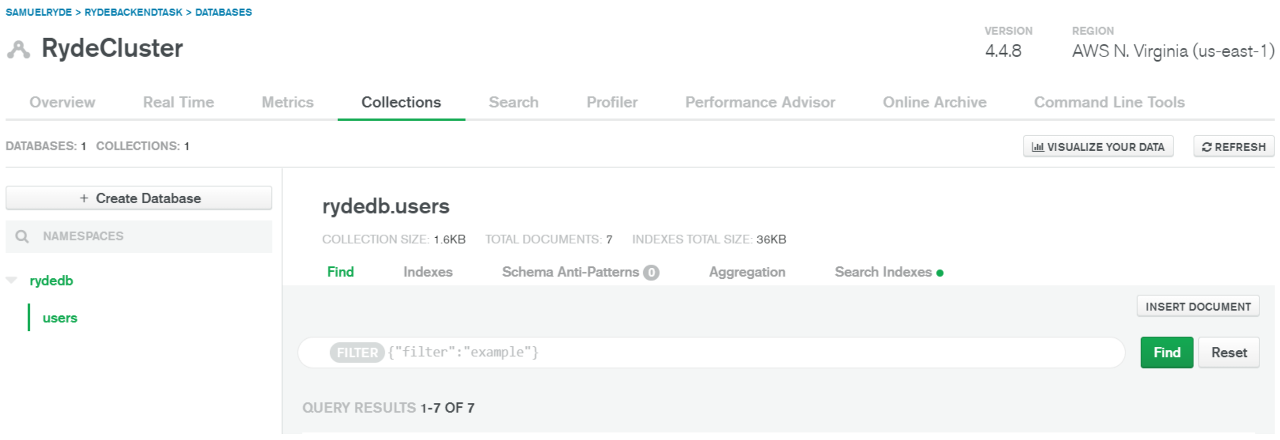
Setting Up

There are two main parts to the set up - Accessing Database and tetting up the server.

Accessing Database

MongoDB was selected as the database as **NoSQL DB was the preferred database** in the project specs. To avoid additional set up hassles, I simply hosted the MongoDB under the free tier in Atlas by creating a dummy email account. Outlined are the steps to accessing the database:

1. Visit the following link: https://www.mongodb.com
2. Click Sign In
3. Select Log in with Google
4. Gmail for login: samuel.david.ryde@gmail.com
5. Password for login: RydePassword!
6. Click **RydeCluster** in the center of the page
7. Click **Collections** in the menu options



8. You will be presented with the interface shown above with the collection details

Setting up the API

Go was used to develop this project as I've been wanting to learn Go and this was the perfect opportunity for me to pick it up. The code for the project can be found in this link: https://github.com/Samuel787/Golang-Backend

1. Git clone the project into a local directory
2. **Go** must be installed in your system. Installation instructions can be found here: https://golang.org/doc/install
3. The following dependencies have to be installed before the project can be run
   a. `"github.com/gorilla/mux"`
   b. `"go.mongodb.org/mongo-driver/bson"`
   c. `"go.mongodb.org/mongo-driver/bson/primitive"`
   d. `"go.mongodb.org/mongo-driver/mongo"`
   e. `"github.com/dgrijalva/jwt-go"`

   f. `"github.com/sirupsen/logrus"`
   g. `"go.mongodb.org/mongo-driver/mongo/options"`
   h. `"go.mongodb.org/mongo-driver/mongo/readpref"`
   i. `"github.com/stretchr/testify/assert"`
   j. `"github.com/stretchr/testify/mock"`
4. To install all the dependencies above, simply run the following:

```
$ go get -d ./...
```

in the project root directory terminal

Running the server

1. Run the following command inside the `server directory`

```
$ go run main.go
```

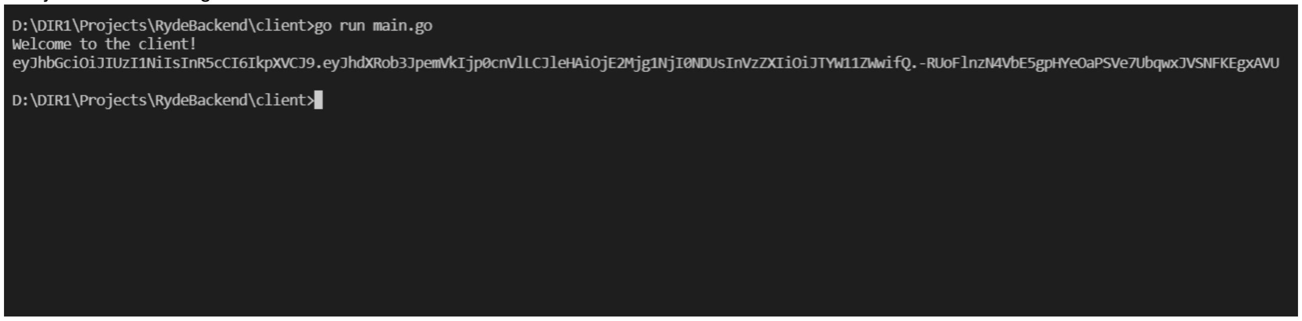
Running the client (just to get auth token)

Only simple auth is implemented for this project. In this project, only the client can access the API. To mock this, a jwt token with a **30 minute validity** is generated in `client/main.go`

1. Run the following command inside the `client directory`

```
$ go run main.go
```

2. The jwt token will be generated as shown below

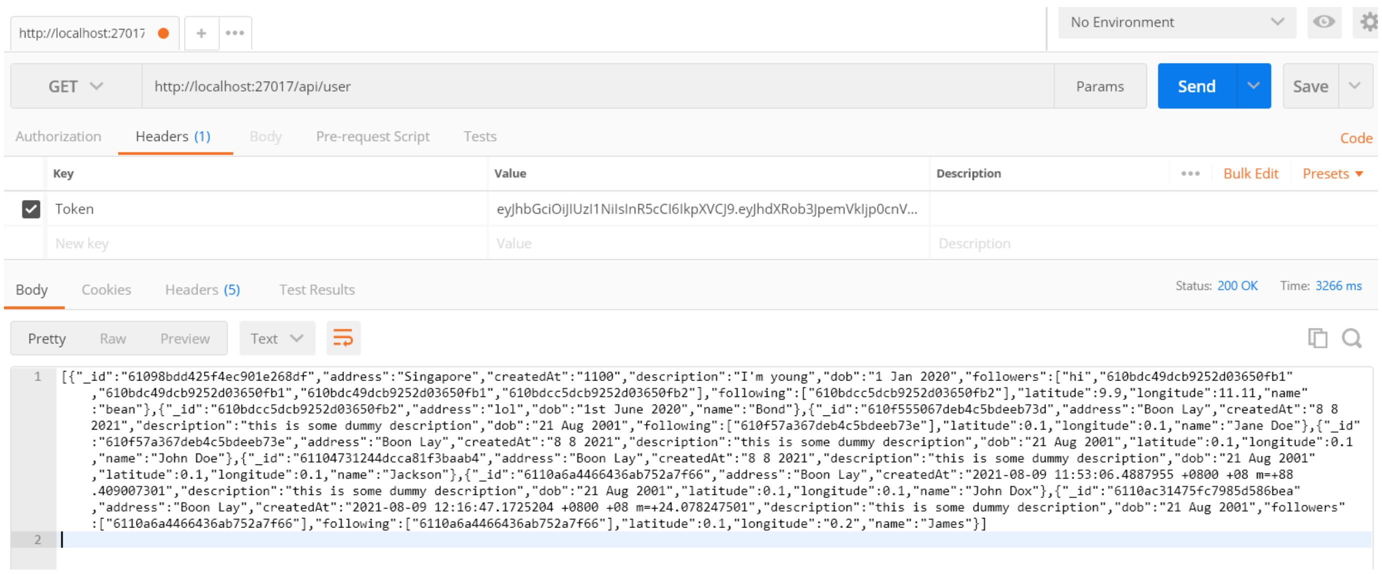


```
D:\DIR1\Projects\RydeBackend\client>go run main.go
Welcome to the client!
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJhdXRob3JpemVkIjp0cnVlLCJleHAiOjE2Mjg1NjI0NDUsInVzZXIiOiJTYW11ZWwifQ.-RUoFlnzN4VbE5gpHYeOaPSVe7UbqwxJVSNFKEgxAVU

D:\DIR1\Projects\RydeBackend\client>
```

Consuming API

Postman can be used to consume the API. For every request, the jwt token generated from above is very important. Postman can be either installed on the local system or used as a chrome browser extension. The browser extension will look something as shown below. For all API calls, the jwt token has to be included in the header with key `Token`.



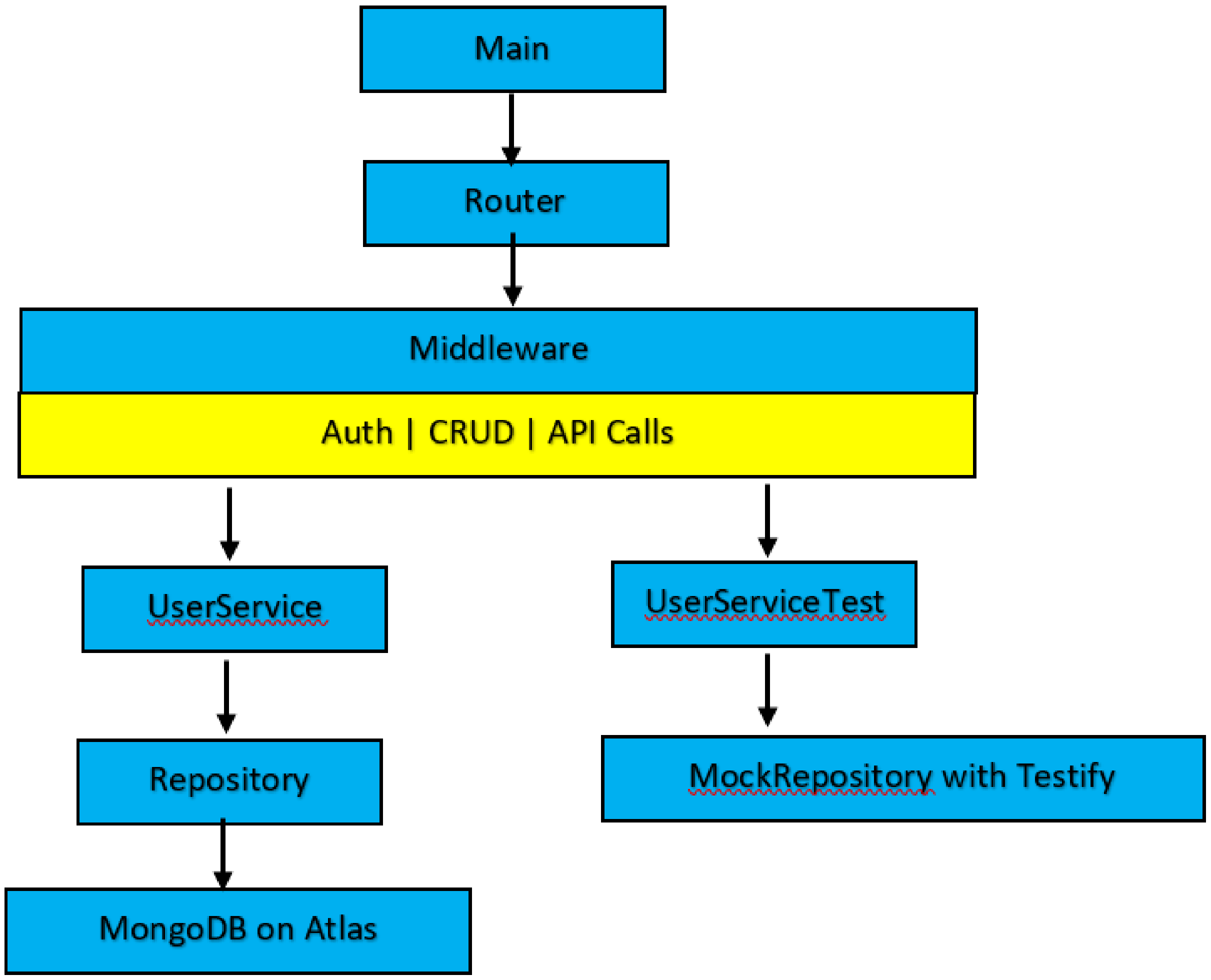
GET http://localhost:27017/api/user

Key	Value	Description
✓ Token	eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJhdXRob3JpemVkIjp0cnV...	
New key	Value	Description

Status: 200 OK Time: 3266 ms

```
[{"_id":"61098bdd425f4ec901e268df","address":"Singapore","createdAt":"1100","description":"I'm young","dob":"1 Jan 2020","followers":["hi","610bdc49dcb9252d03650fb1"
    ,"610bdc49dcb9252d03650fb1","610bdc49dcb9252d03650fb1","610bdcc5dcb9252d03650fb2"],"following":["610bdcc5dcb9252d03650fb2"],"latitude":9.9,"longitude":11.11,"name"
    :"bean"},{"_id":"610bdcc5dcb9252d03650fb2","address":"lol","dob":"1st June 2020","name":"Bond"},{"_id":"610f555067deb4c5bdeeb73d","address":"Boon Lay","createdAt":"8 8
    2021","description":"this is some dummy description","dob":"21 Aug 2001","following":["610f57a367deb4c5bdeeb73e"],"latitude":0.1,"longitude":0.1,"name":"Jane Doe"},{"_id"
    :"610f57a367deb4c5bdeeb73e","address":"Boon Lay","createdAt":"8 8 2021","description":"this is some dummy description","dob":"21 Aug 2001","latitude":0.1,"longitude":0.1
    ,"name":"John Doe"},{"_id":"61104731244dcca81f3baab4","address":"Boon Lay","createdAt":"8 8 2021","description":"this is some dummy description","dob":"21 Aug 2001"
    ,"latitude":0.1,"longitude":0.1,"name":"Jackson"},{"_id":"6110a6a4466436ab752a7f66","address":"Boon Lay","createdAt":"2021-08-09 11:53:06.4887955 +0800 +08 m=+88
    .409007301","description":"this is some dummy description","dob":"21 Aug 2001","latitude":0.1,"longitude":0.1,"name":"John Dox"},{"_id":"6110ac31475fc7985d586bea"
    ,"address":"Boon Lay","createdAt":"2021-08-09 12:16:47.1725204 +0800 +08 m=+24.078247501","description":"this is some dummy description","dob":"21 Aug 2001","followers"
    :["6110a6a4466436ab752a7f66"],"following":["6110a6a4466436ab752a7f66"],"latitude":0.1,"longitude":"0.2","name":"James"}]
```

High Level Architecture



Main contains the driver code. It listens for HTTP requests on `port 27017`

Router routes the various different API requests to the correct end points in the middleware. It first routes the request through an Auth middleware to ensure that the request is authorized

Middleware the middleware handles Auth, CRUD and API calls

UserService contains the main logic for the various requests

Repository contains database API to interact with the MongoDB on Atlas

UserServiceTest tests the logic in UserService with `testify`

MockRepository creates a dummy database for testing purposes

User Model

```
type User struct {
    ID primitive.ObjectID `bson:"_id,omitempty"`
    Name string `bson:"name,omitempty"`
    DOB string `bson:"dob,omitempty"`
    Address string `bson:"address,omitempty"`
    Description string `bson:"description,omitempty"`
    CreatedAt string `bson:"createdAt,omitempty"`
    Followers []string `bson:"followers,omitempty"`
    Following []string `bson:"following,omitempty"`
    Latitude float64 `bson:"latitude,omitempty"`
    Longitude float64 `bson:"longitude,omitempty"`
}
```

ID is uniquely given to every user by MongoDB to uniquely identify user

For this project, I've set the user ID as the unique primary key. This will be used for the API calls as well to uniquely identify users. Not user's name. The simple reason for this is that multiple users can have the same names. For example, Facebook adopts this.

Features

This section details out the various different API calls that can be made and their implementation details.

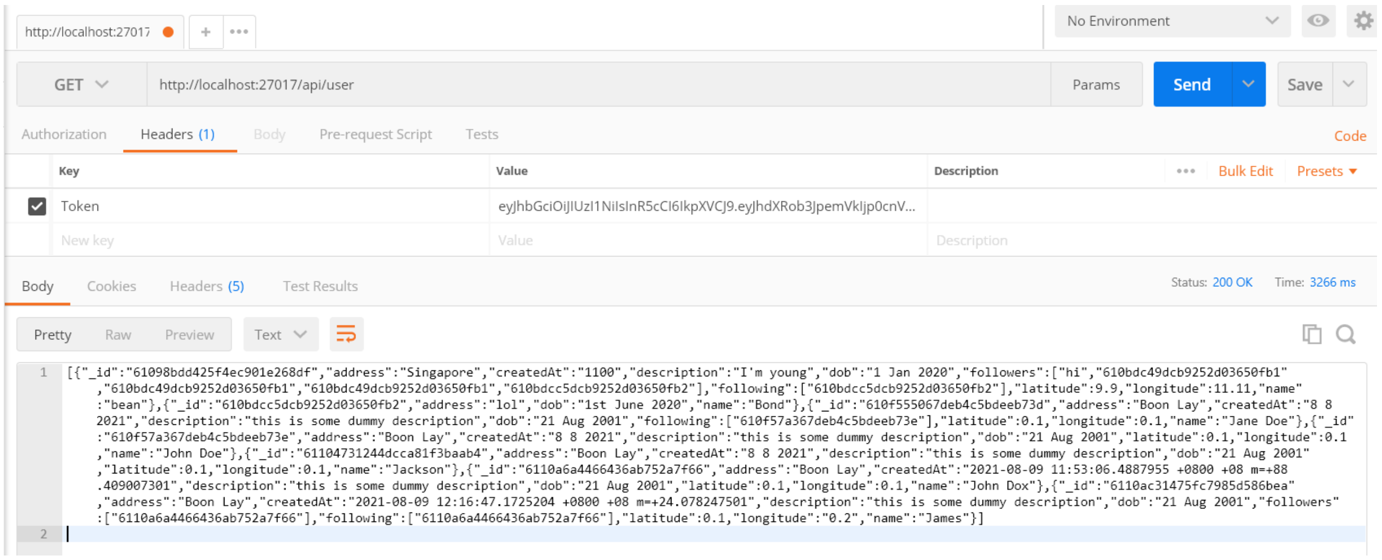
Auth

Auth is implemented with `jwt "github.com/dgrijalva/jwt-go"`

Both the server and the user shares a SigningKey. The client then generates a token with a 30-minute validity. The server verifies this in `AuthorizeUser` in middleware.

When making API requests, this token has to be included in the Header of the requests.

Valid Token



URL: http://localhost:27017/api/user

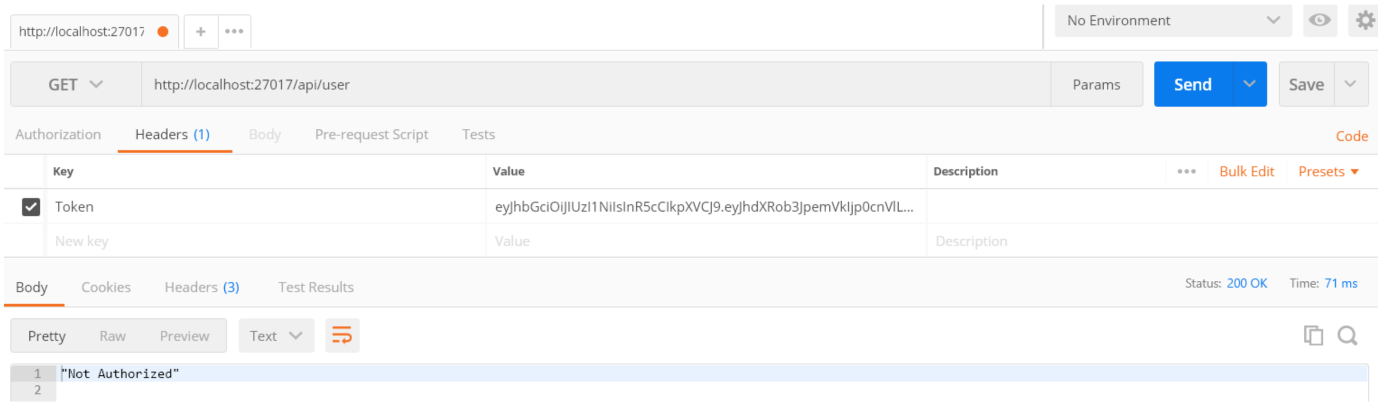
Key	Value	Description
✓ Token	eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJhdXRob3JpemVkIjp0cnV...	
New key	Value	Description

Status: 200 OK Time: 3266 ms

```
[{"_id":"61098bdd425f4ec901e268df","address":"Singapore","createdAt":"1100","description":"I'm young","dob":"1 Jan 2020","followers":["hi","610bdc49dcb9252d03650fb1"
    ,"610bdc49dcb9252d03650fb1","610bdc49dcb9252d03650fb1","610bdcc5dcb9252d03650fb2"],"following":["610bdcc5dcb9252d03650fb2"],"latitude":9.9,"longitude":11.11,"name"
    :"bean"},{"_id":"610bdcc5dcb9252d03650fb2","address":"lol","dob":"1st June 2020","name":"Bond"},{"_id":"610f555067deb4c5bdeeb73d","address":"Boon Lay","createdAt":"8 8
    2021","description":"this is some dummy description","dob":"21 Aug 2001","following":["610f57a367deb4c5bdeeb73e"],"latitude":0.1,"longitude":0.1,"name":"Jane Doe"},{"_id"
    :"610f57a367deb4c5bdeeb73e","address":"Boon Lay","createdAt":"8 8 2021","description":"this is some dummy description","dob":"21 Aug 2001","latitude":0.1,"longitude":0.1
    ,"name":"John Doe"},{"_id":"61104731244dcca81f3baab4","address":"Boon Lay","createdAt":"8 8 2021","description":"this is some dummy description","dob":"21 Aug 2001"
    ,"latitude":0.1,"longitude":0.1,"name":"Jackson"},{"_id":"6110a6a4466436ab752a7f66","address":"Boon Lay","createdAt":"2021-08-09 11:53:06.4887955 +0800 +08 m=+88
    .409007301","description":"this is some dummy description","dob":"21 Aug 2001","latitude":0.1,"longitude":0.1,"name":"John Dox"},{"_id":"6110ac31475fc7985d586bea"
    ,"address":"Boon Lay","createdAt":"2021-08-09 12:16:47.1725204 +0800 +08 m=+24.078247501","description":"this is some dummy description","dob":"21 Aug 2001","followers"
    :["6110a6a4466436ab752a7f66"],"following":["6110a6a4466436ab752a7f66"],"latitude":0.1,"longitude":"0.2","name":"James"}]
```

When a valid token is used in the header to make the request, the API call will be successful.

Invalid Token



URL: http://localhost:27017/api/user

Key	Value	Description
✓ Token	eyJhbGciOiJIUzI1NiIsInR5cCIkpXVCJ9.eyJhdXRob3JpemVkIjp0cnVlL...	
New key	Value	Description

Status: 200 OK Time: 71 ms

```
"Not Authorized"
```

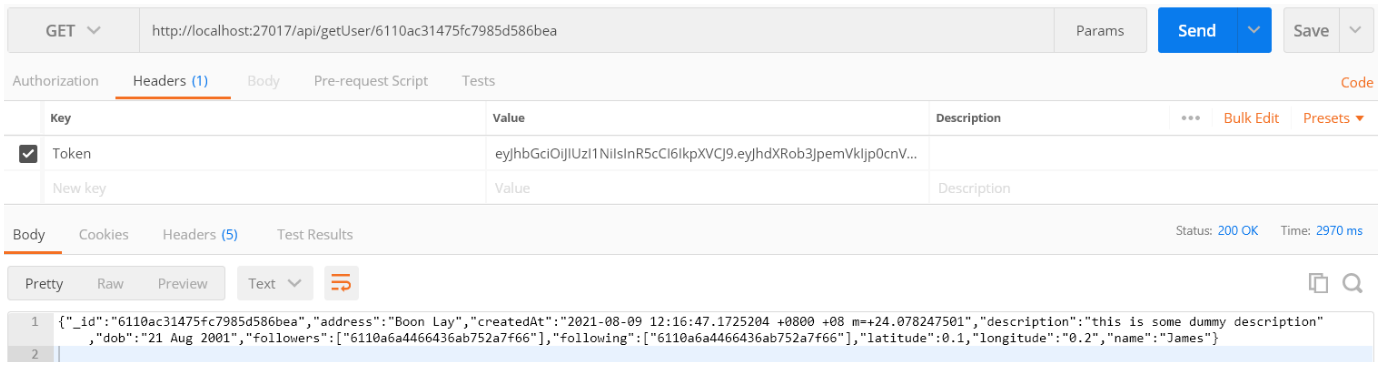
When an invalid token is used in the header to make the request, the user will be notified that he’s not authorized.

GET

Used to retrieve a specific user’s information using the user’s ID.

```
GET http://localhost:27017/api/getUser/{id}
```

Successful Request



GET http://localhost:27017/api/getUser/6110ac31475fc7985d586bea

Headers (1)

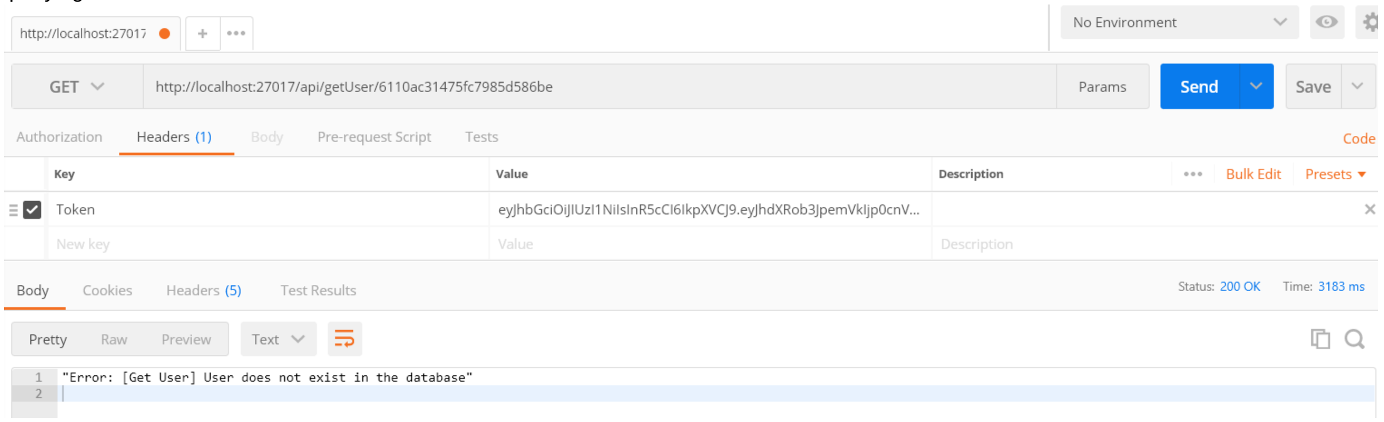
Key	Value	Description
Token	eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJhdXRob3JpemVkIjp0cnV...	

Status: 200 OK Time: 2970 ms

```
{"_id":"6110ac31475fc7985d586bea","address":"Boon Lay","createdAt":"2021-08-09 12:16:47.1725204 +0800 +08 m=+24.078247501","description":"this is some dummy description"
    ,"dob":"21 Aug 2001","followers":["6110a6a4466436ab752a7f66"],"following":["6110a6a4466436ab752a7f66"],"latitude":0.1,"longitude":"0.2","name":"James"}
```

Unsuccessful Request

querying an invalid user id



GET http://localhost:27017/api/getUser/6110ac31475fc7985d586be

Headers (1)

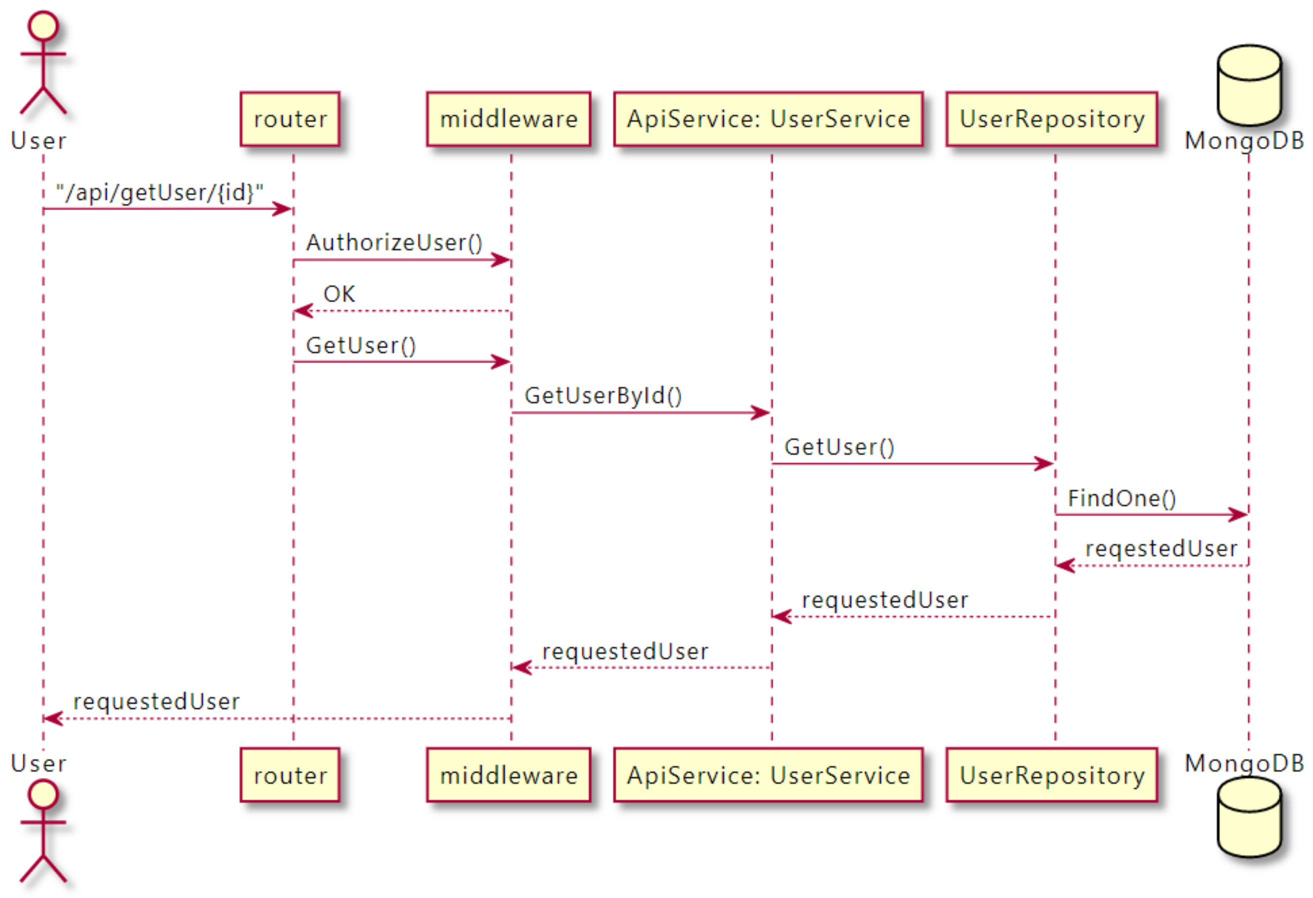
Key	Value	Description
Token	eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJhdXRob3JpemVkIjp0cnV...	

Status: 200 OK Time: 3183 ms

```
"Error: [Get User] User does not exist in the database"
```

Implementation Details

For success scenario



Edge Cases Handled

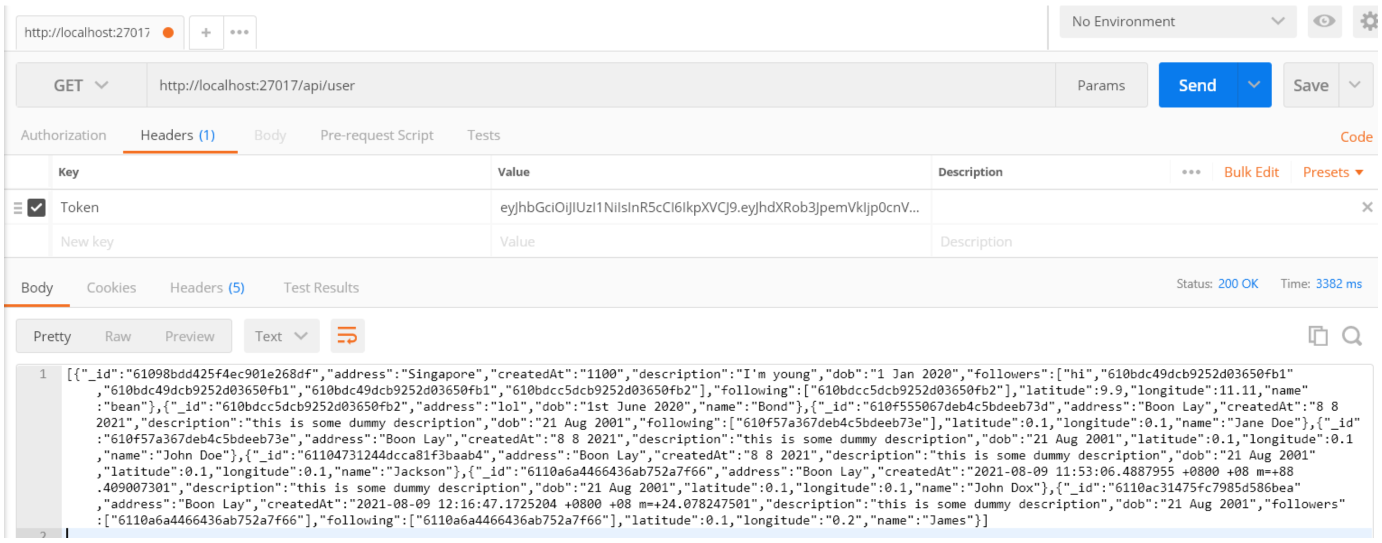
1. Invalid user id
2. User is unauthorized
3. Connection to DB fails

GET ALL USERS

Used to retrieve all the users in the database

```
GET http://localhost:27017/api/user
```

Successful Request



GET http://localhost:27017/api/user

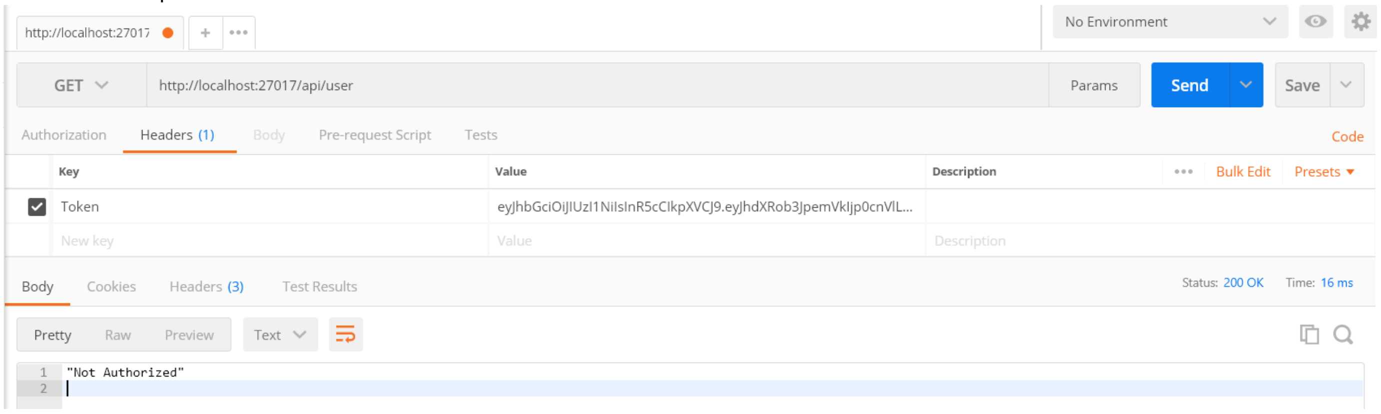
Key	Value	Description
Token	eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJhdXRob3JpemVkIjp0cnV...	
New key	Value	Description

Status: 200 OK Time: 3382 ms

```
[{"_id":"61098bdd425f4ec901e268df","address":"Singapore","createdAt":"1100","description":"I'm young","dob":"1 Jan 2020","followers":["hi","610bdc49dcb9252d03650fb1"
    ,"610bdc49dcb9252d03650fb1","610bdc49dcb9252d03650fb1","610bdcc5dcb9252d03650fb2"],"following":["610bdcc5dcb9252d03650fb2"],"latitude":9.9,"longitude":11.11,"name"
    :"bean"},{"_id":"610bdcc5dcb9252d03650fb2","address":"lol","dob":"1st June 2020","name":"Bond"},{"_id":"610f555067deb4c5bdeeb73d","address":"Boon Lay","createdAt":"8 8
    2021","description":"this is some dummy description","dob":"21 Aug 2001","following":["610f57a367deb4c5bdeeb73e"],"latitude":0.1,"longitude":0.1,"name":"Jane Doe"},{"_id"
    :"610f57a367deb4c5bdeeb73e","address":"Boon Lay","createdAt":"8 8 2021","description":"this is some dummy description","dob":"21 Aug 2001","latitude":0.1,"longitude":0.1
    ,"name":"John Doe"},{"_id":"61104731244dcca81f3baab4","address":"Boon Lay","createdAt":"8 8 2021","description":"this is some dummy description","dob":"21 Aug 2001"
    ,"latitude":0.1,"longitude":0.1,"name":"Jackson"},{"_id":"6110a6a4466436ab752a7f66","address":"Boon Lay","createdAt":"2021-08-09 11:53:06.4887955 +0800 +08 m=+88
    .409007301","description":"this is some dummy description","dob":"21 Aug 2001","latitude":0.1,"longitude":0.1,"name":"John Dox"},{"_id":"6110ac31475fc7985d586bea"
    ,"address":"Boon Lay","createdAt":"2021-08-09 12:16:47.1725204 +0800 +08 m=+24.078247501","description":"this is some dummy description","dob":"21 Aug 2001","followers"
    :["6110a6a4466436ab752a7f66"],"following":["6110a6a4466436ab752a7f66"],"latitude":0.1,"longitude":"0.2","name":"James"}]
```

Unsuccessful Request

Unauthorized request



GET http://localhost:27017/api/user

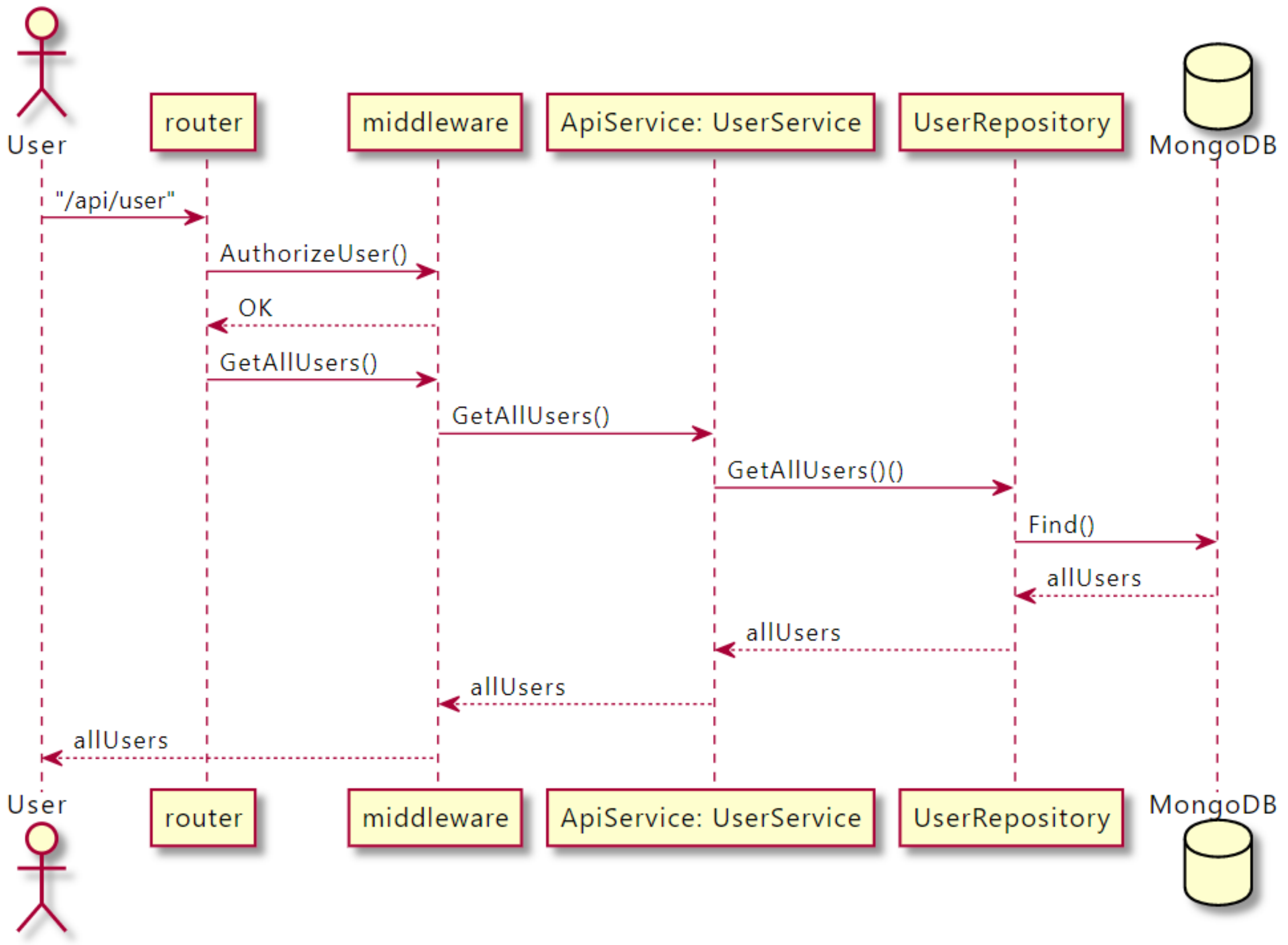
Key	Value	Description
Token	eyJhbGciOiJIUzI1NiIsInR5cCIkpXVCJ9.eyJhdXRob3JpemVkIjp0cnVlL...	
New key	Value	Description

Status: 200 OK Time: 16 ms

```
"Not Authorized"
```

Implementation Details

For success scenario



Edge Cases Handled

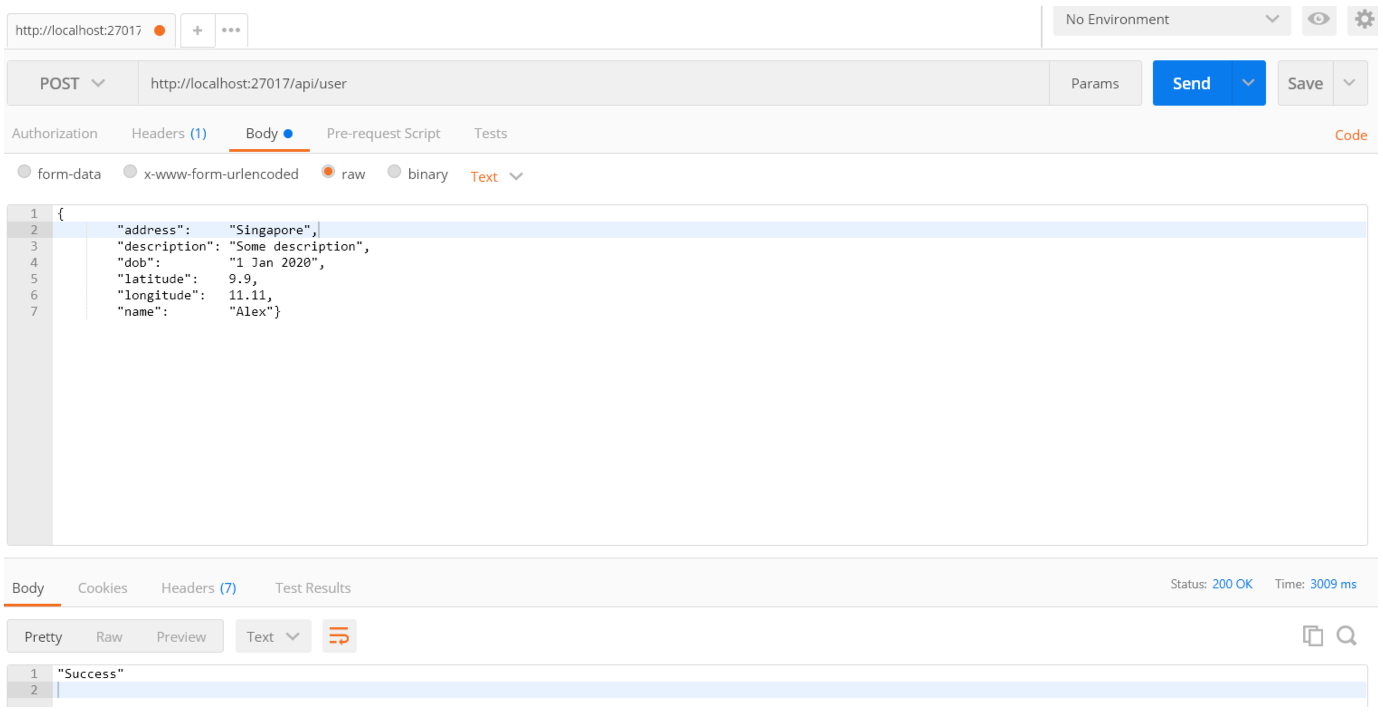
1. Unauthorized user
2. DB errors

CREATE

Used to create user in the database

```
POST http://localhost:27017/api/user
```

Successful Request



POST http://localhost:27017/api/user

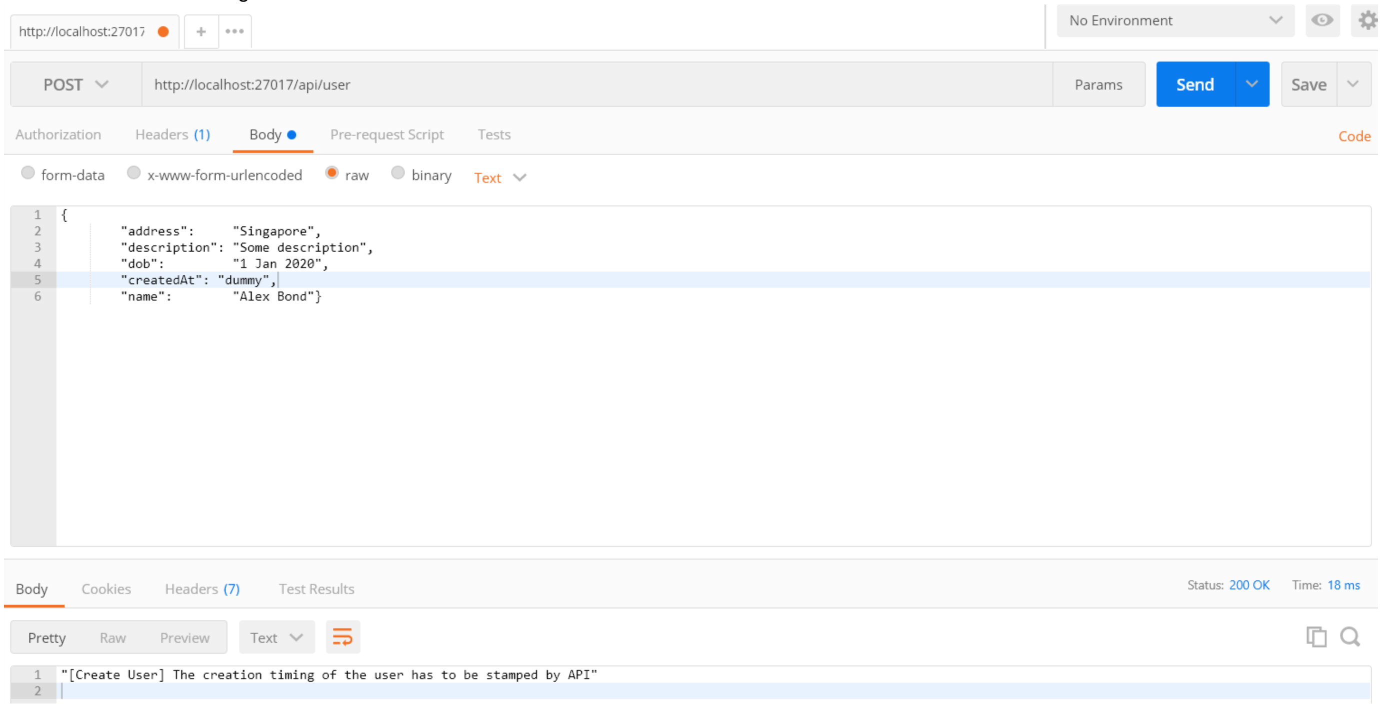
```
{
        "address":     "Singapore",
        "description": "Some description",
        "dob":         "1 Jan 2020",
        "latitude":    9.9,
        "longitude":   11.11,
        "name":        "Alex"}
```

Status: 200 OK Time: 3009 ms

```
"Success"
```

Unsuccessful Request

User should not be setting CreatedAt



POST http://localhost:27017/api/user

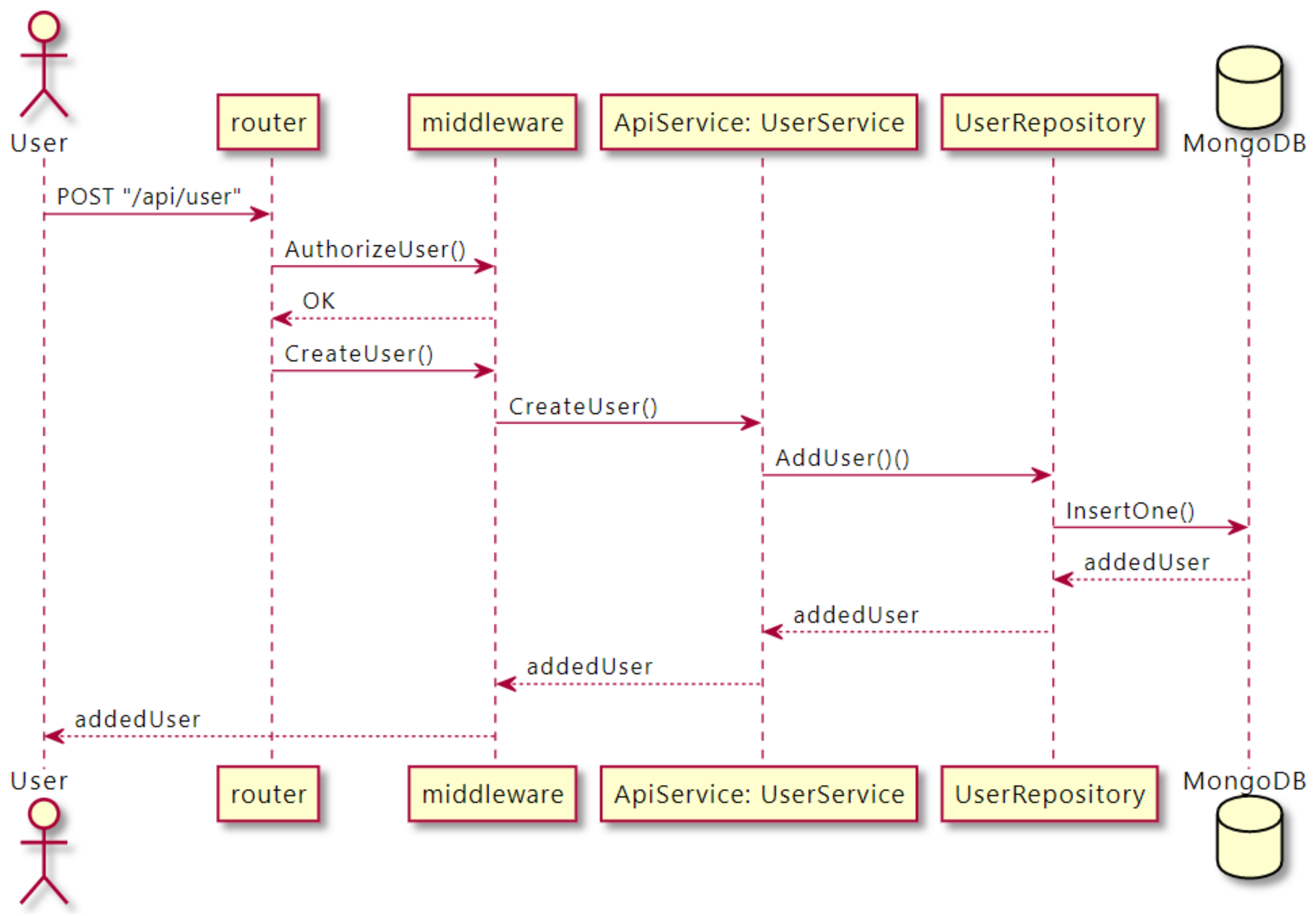
```
{
        "address":     "Singapore",
        "description": "Some description",
        "dob":         "1 Jan 2020",
        "createdAt": "dummy",
        "name":        "Alex Bond"}
```

Status: 200 OK Time: 18 ms

```
"[Create User] The creation timing of the user has to be stamped by API"
```

Implementation Details

For success scenario



Edge Cases Handled

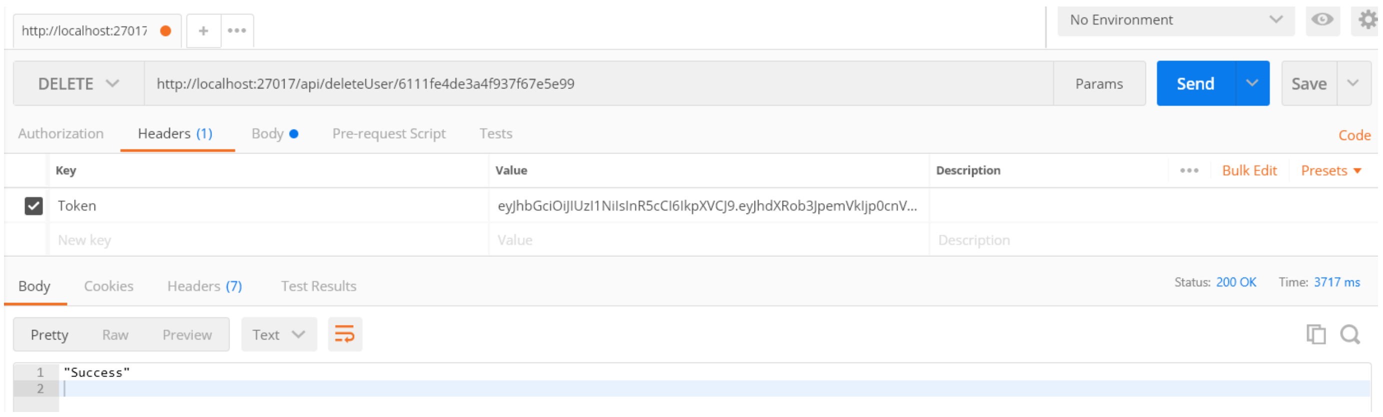
1. Unauthorized user
2. CreatedAt being set by user
3. Compulsory fields not present
4. DB errors

DELETE

Used to delete user in database by user ID

```
POST http://localhost:27017/api/deleteUser/{id}
```

Successful Request



DELETE http://localhost:27017/api/deleteUser/6111fe4de3a4f937f67e5e99

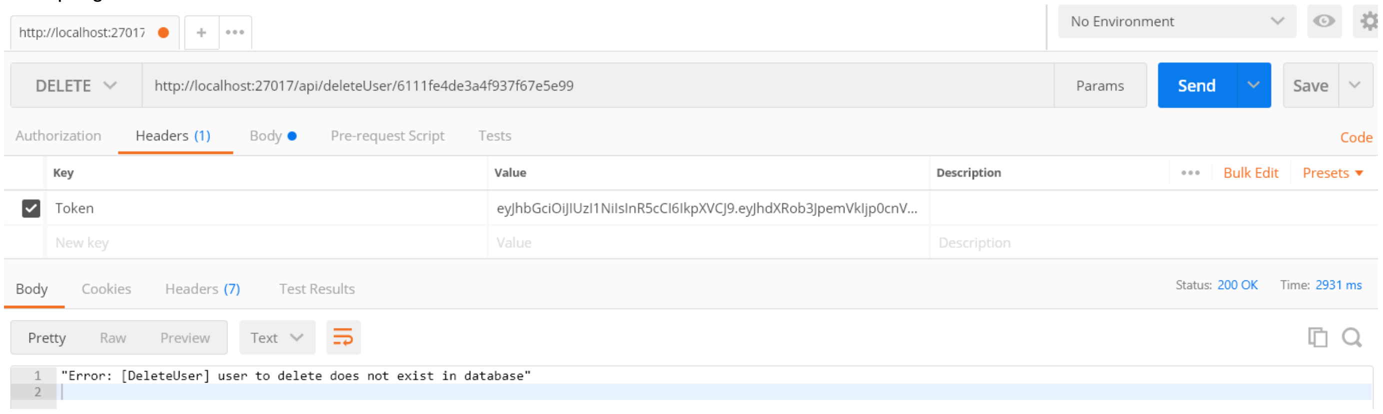
Key	Value	Description
Token	eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJhdXRob3JpemVkIjp0cnV...	

Status: 200 OK Time: 3717 ms

```
"Success"
```

Unsuccessful Request

Attempting to delete user who doesn’t exist



DELETE http://localhost:27017/api/deleteUser/6111fe4de3a4f937f67e5e99

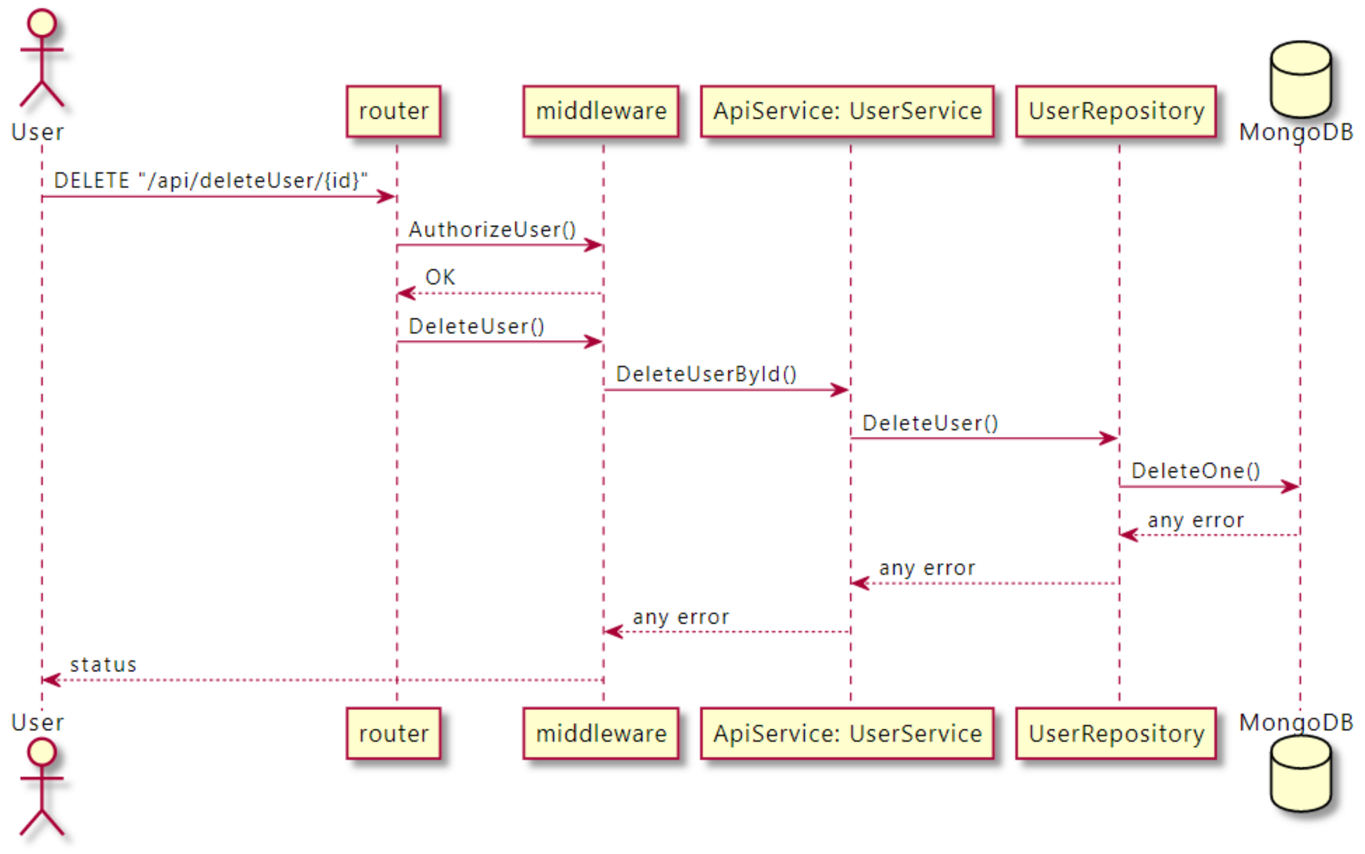
Key	Value	Description
Token	eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJhdXRob3JpemVkIjp0cnV...	

Status: 200 OK Time: 2931 ms

```
"Error: [DeleteUser] user to delete does not exist in database"
```

Implementation Details

For success scenario



Edge Cases Handled

1. Unauthorized user
2. Deleting user who does not exist in database

UPDATE

Used to update user's information in the database

```
PUT http://localhost:27017/api/updateUser
```

User's attributes that can be updated are listed in the table below

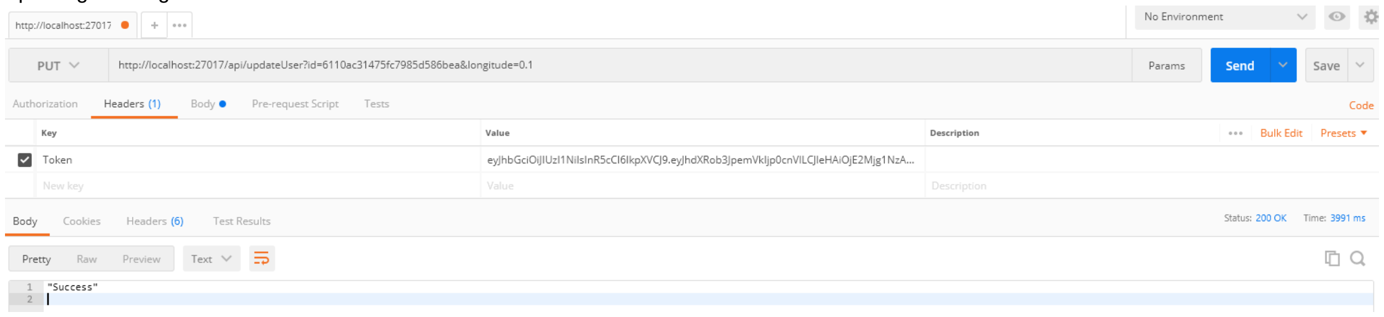
Key	**Value type**	**Optional?**
"id"	string	Required
"name"	string	Optional
"dob"	string	Optional
"address"	string	Optional
"description"	string	Optional
"latitude"	float64	Optional
"longitude"	float64	Optional

Example request:

```
PUT http://localhost:27017/api/updateUser?
id=6110ac31475fc785d586bea&longitude=0.1
```

Successful Request

Updating the longitude of the user



PUT http://localhost:27017/api/updateUser?id=6110ac31475fc7985d586bea&longitude=0.1

Headers (1)

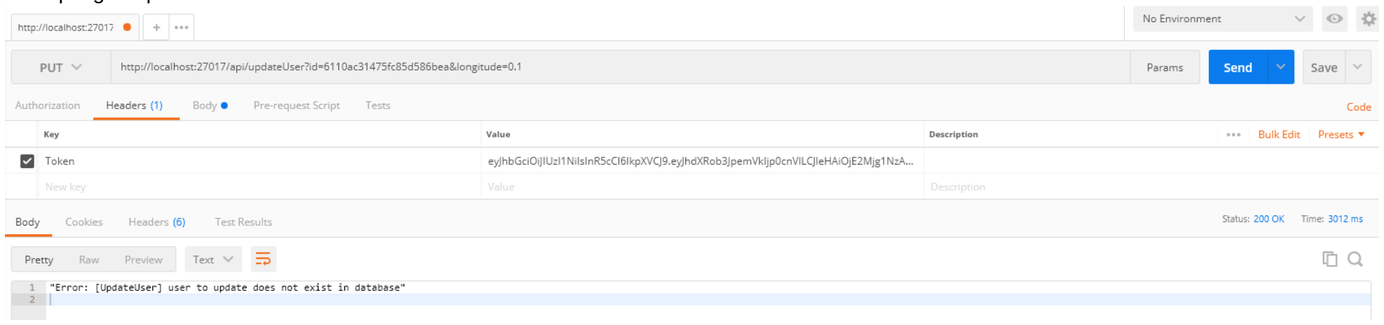
Key	Value	Description
Token	eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJhdXRob3JpemVkIjp0cnVlLCJleHAiOjE2Mjg1NzA...	

Status: 200 OK Time: 3991 ms

```
"Success"
```

Unsuccessful Request

Attempting to update user who doesn’t exist



PUT http://localhost:27017/api/updateUser?id=6110ac31475fc85d586bea&longitude=0.1

Headers (1)

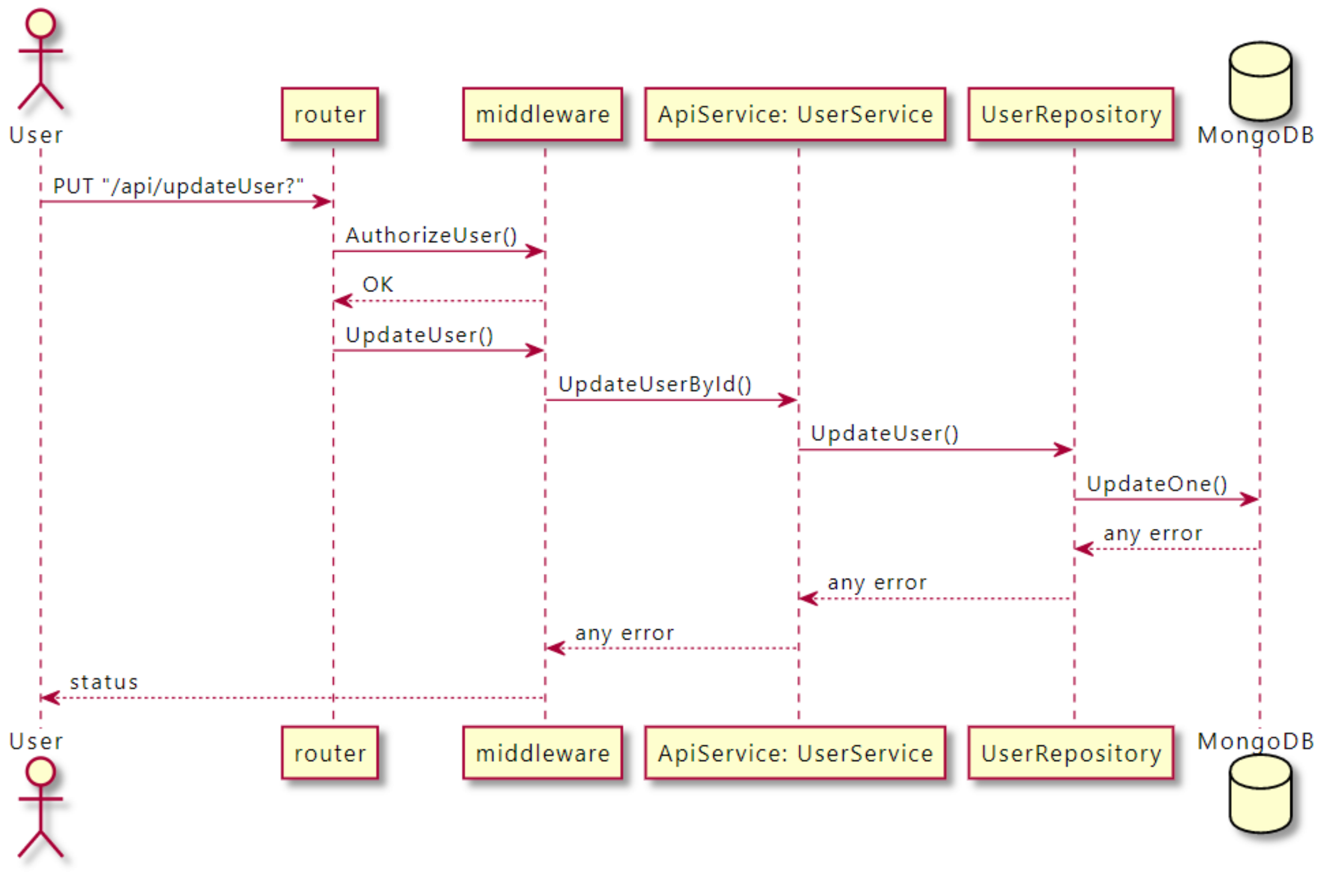
Key	Value	Description
Token	eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJhdXRob3JpemVkIjp0cnVlLCJleHAiOjE2Mjg1NzA...	

Status: 200 OK Time: 3012 ms

```
"Error: [UpdateUser] user to update does not exist in database"
```

Implementation Details

For success scenario



Edge Cases Handled

1. Unauthorized user
2. User does not exist in database
3. Database errors

ADD FOLLOWER

Used to add follower to a user

```
PUT http://localhost:27017/api/addFollower
```

API Parameters

Key	Value type	Optional?
"userId"	string	Required
"followerId"	string	Required

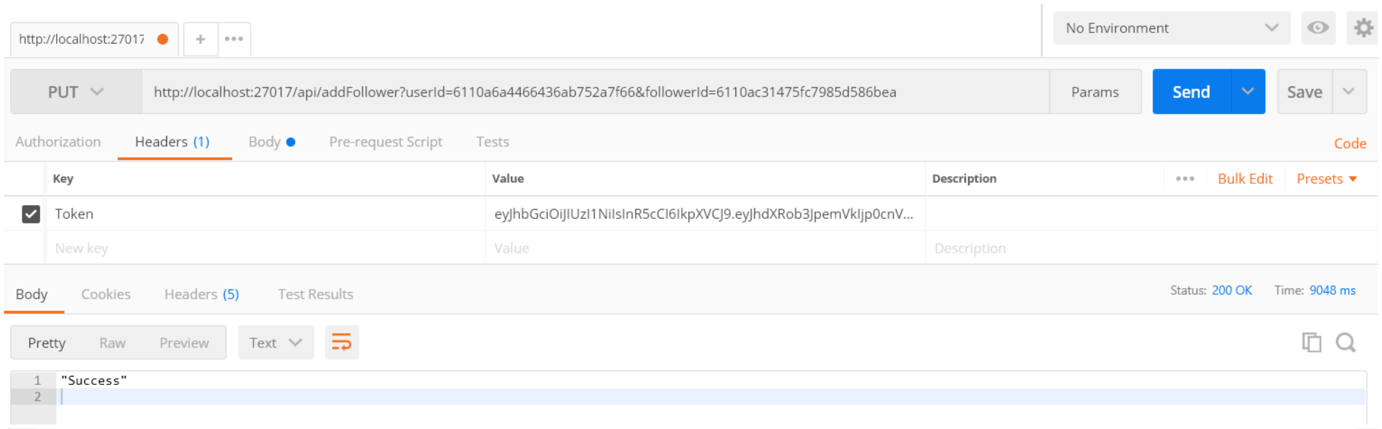
Example request:

```
PUT http://localhost:27017/api/addFollower?
userId=6110a6a4466436ab752a7f66&followerId=6110ac31475fc7985d586bea
```

Successful Request

Adding `6110ac31475fc7985d586bea` as follower for `6110a6a4466436ab752a7f66`

This means that `6110ac31475fc7985d586bea` is following `6110a6a4466436ab752a7f66`

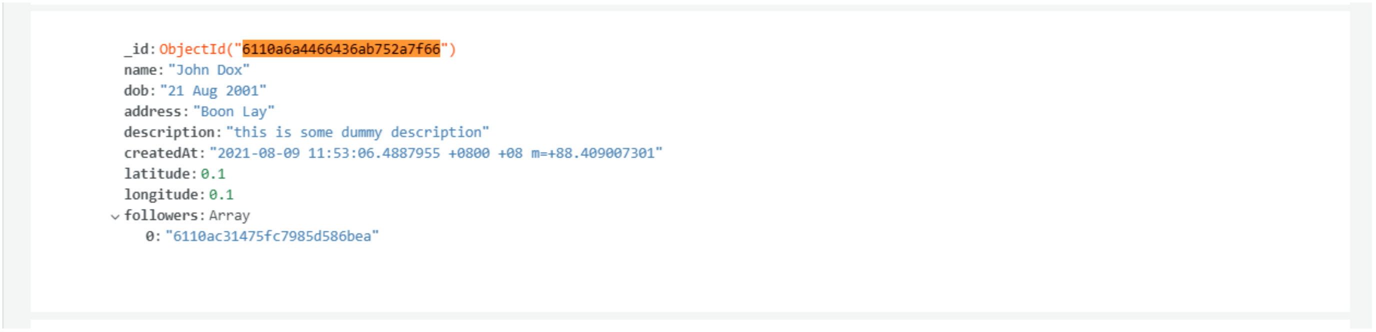


PUT http://localhost:27017/api/addFollower?userId=6110a6a4466436ab752a7f66&followerId=6110ac31475fc7985d586bea

Key	Value	Description
Token	eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJhdXRob3JpemVkIjp0cnV...	

Status: 200 OK Time: 9048 ms

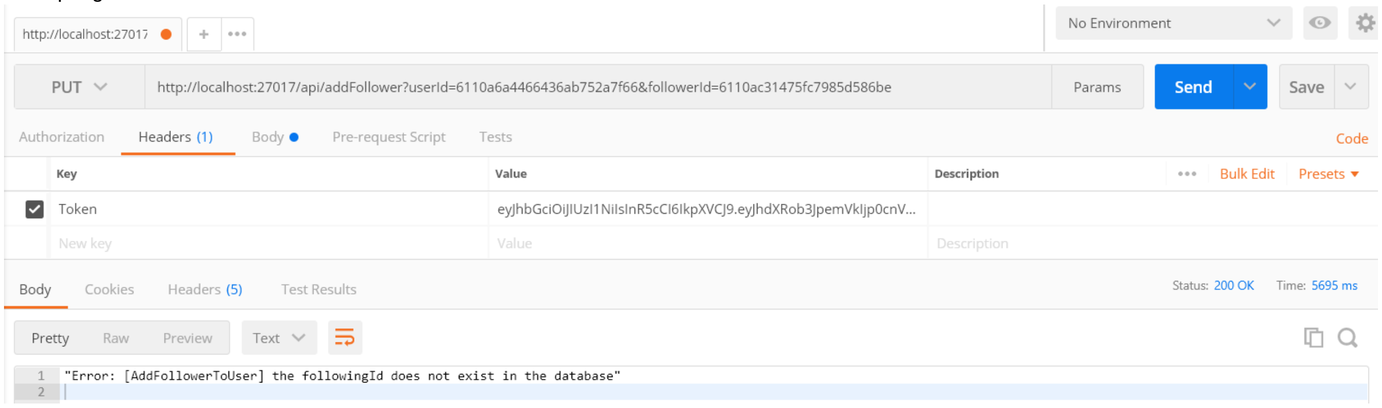
```
"Success"
```



```
_id: ObjectId("6110a6a4466436ab752a7f66")
name: "John Dox"
dob: "21 Aug 2001"
address: "Boon Lay"
description: "this is some dummy description"
createdAt: "2021-08-09 11:53:06.4887955 +0800 +08 m=+88.409007301"
latitude: 0.1
longitude: 0.1
followers: Array
    0: "6110ac31475fc7985d586bea"
```

Unsuccessful Request

Attempting to add someone who doesn’t exist as a follower



PUT http://localhost:27017/api/addFollower?userId=6110a6a4466436ab752a7f66&followerId=6110ac31475fc7985d586be

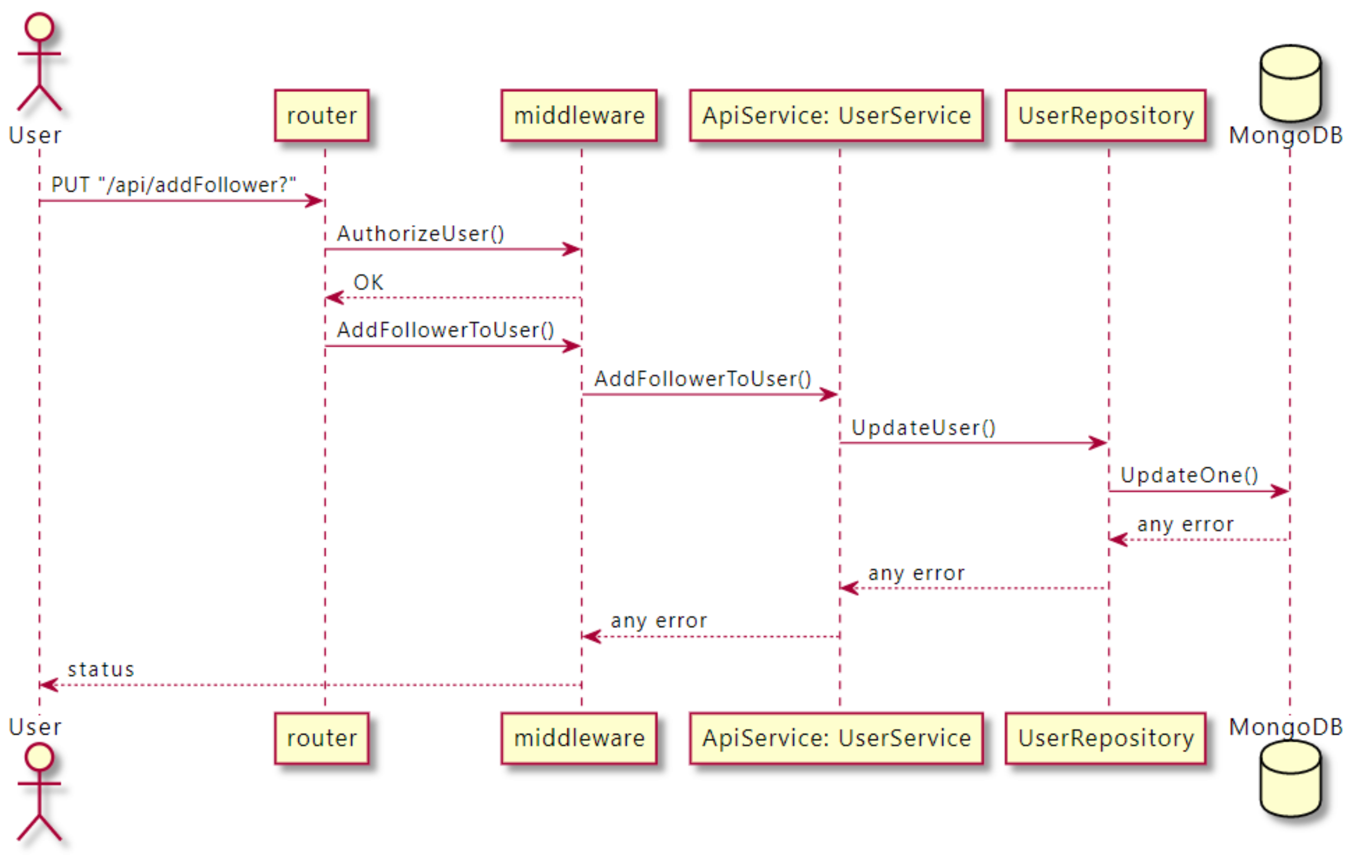
Key	Value	Description
Token	eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJhdXRob3JpemVkIjp0cnV...	

Status: 200 OK Time: 5695 ms

```
"Error: [AddFollowerToUser] the followingId does not exist in the database"
```

Implementation Details

For success scenario



Edge Cases Handled

1. Unauthorized user
2. User does not exist in database
3. followerId is already in the followers list for userId
4. Database errors

ADD FOLLOWING

Used to add following to a user

```
PUT http://localhost:27017/api/addFollowing
```

API Parameters

Key	Value type	Optional?
“userId”	string	Required
“followingId”	string	Required

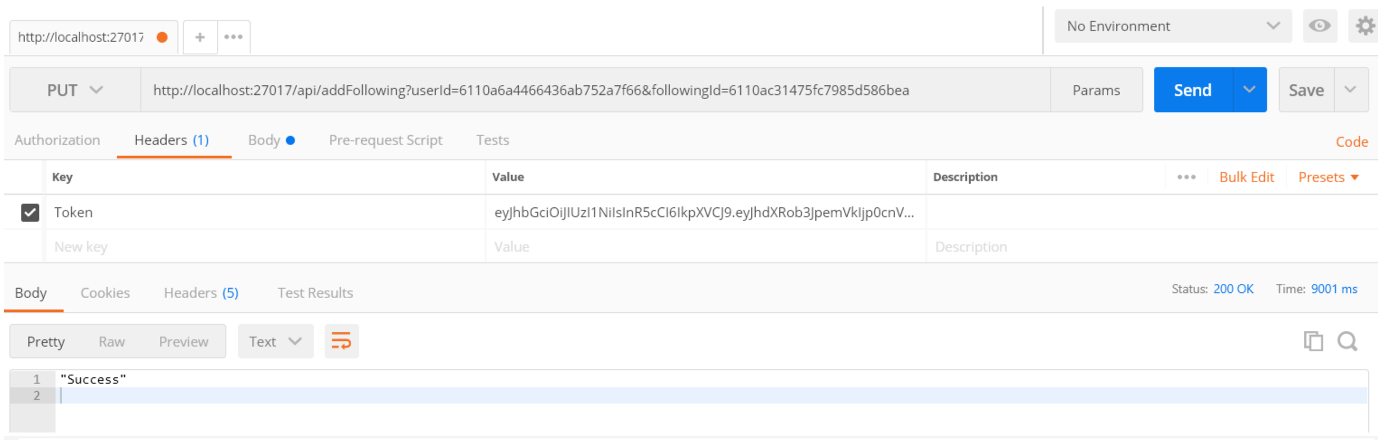
Example request:

```
PUT http://localhost:27017/api/addFollower?
userId=6110a6a4466436ab752a7f66&followerId=6110ac31475fc7985d586bea
```

Successful Request

Adding `6110ac31475fc7985d586bea` as following for `6110a6a4466436ab752a7f66`

This means that `6110a6a4466436ab752a7f66`is following `6110ac31475fc7985d586bea`

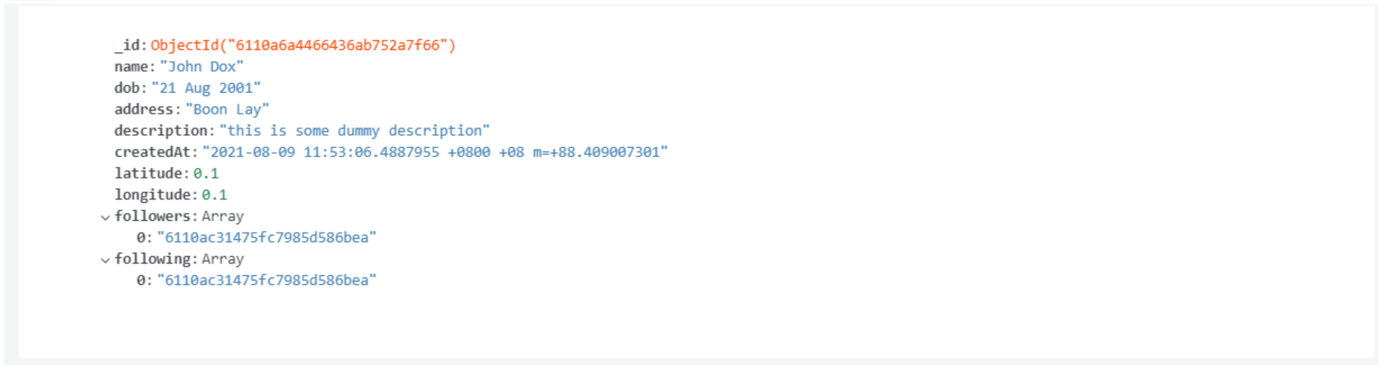


PUT http://localhost:27017/api/addFollowing?userId=6110a6a4466436ab752a7f66&followingId=6110ac31475fc7985d586bea

Key	Value	Description
Token	eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJhdXRob3JpemVkIjp0cnV...	

Status: 200 OK Time: 9001 ms

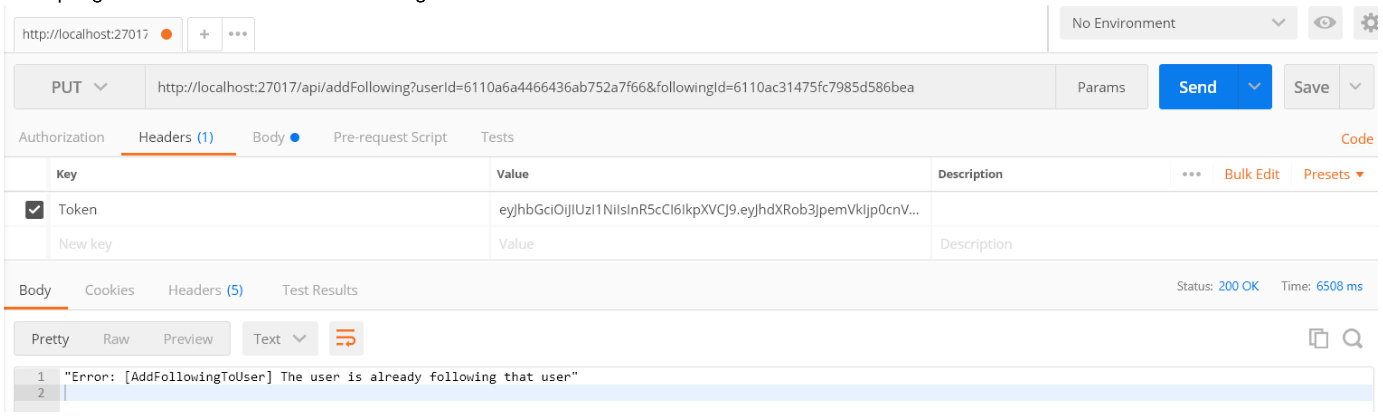
```
"Success"
```



```
_id: ObjectId("6110a6a4466436ab752a7f66")
name: "John Dox"
dob: "21 Aug 2001"
address: "Boon Lay"
description: "this is some dummy description"
createdAt: "2021-08-09 11:53:06.4887955 +0800 +08 m=+88.409007301"
latitude: 0.1
longitude: 0.1
followers: Array
   0: "6110ac31475fc7985d586bea"
following: Array
   0: "6110ac31475fc7985d586bea"
```

Unsuccessful Request

Attempting to add the same user as following



PUT http://localhost:27017/api/addFollowing?userId=6110a6a4466436ab752a7f66&followingId=6110ac31475fc7985d586bea

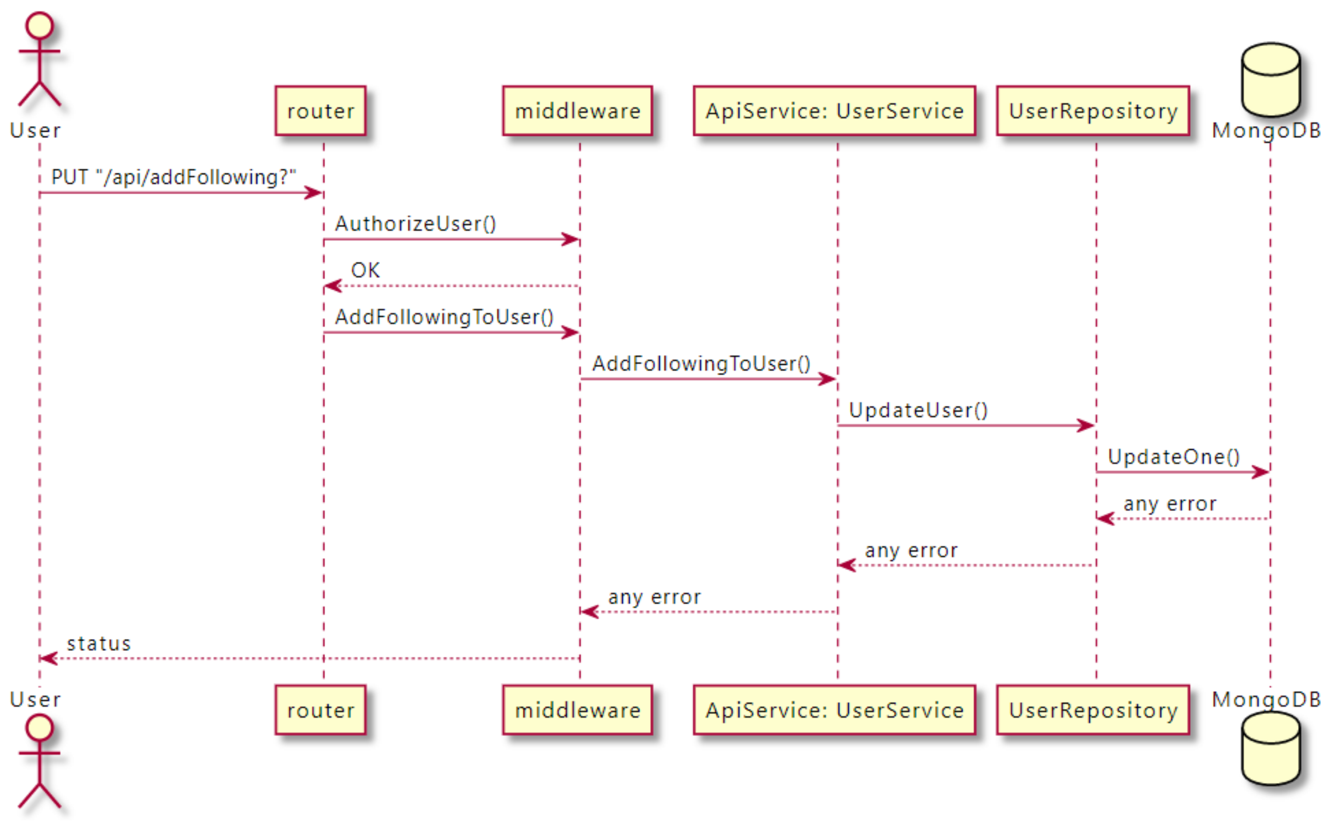
Key	Value	Description
Token	eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJhdXRob3JpemVkIjp0cnV...	

Status: 200 OK Time: 6508 ms

```
"Error: [AddFollowingToUser] The user is already following that user"
```

Implementation Details

For success scenario



Edge Cases Handled

1. Unauthorized user
2. User does not exist in database
3. followingId is already in the following list for userId
4. Database errors

FIND NEARBY USERS

This allows a user to retrieve other users (who must be in the following list) who are near them. The rationale for only looking in the following' list is because these are the users that the user is following.

```
PUT http://localhost:27017/api/nearByFollowing?
```

API Parameters

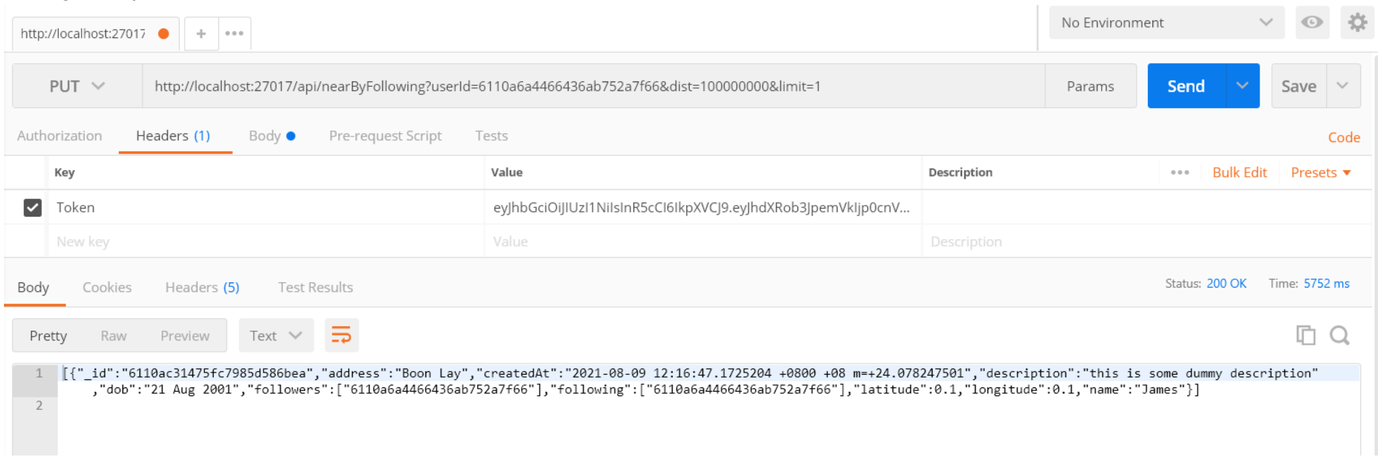
Key	Value type	Optional?	Description
“userId”	string	Required	userId of the user in interest
“dist”	float64	Required	Within what distance radius?
“limit”	int	Required	The maximum number of results to return

Example request:

```
PUT http://localhost:27017/api/nearByFollowing?
userId=6110a6a4466436ab752a7f66&dist=100000000&limit=1
```

Successful Request

Finding nearby users for `6110a6a4466436ab752a7f66`



PUT http://localhost:27017/api/nearByFollowing?userId=6110a6a4466436ab752a7f66&dist=100000000&limit=1

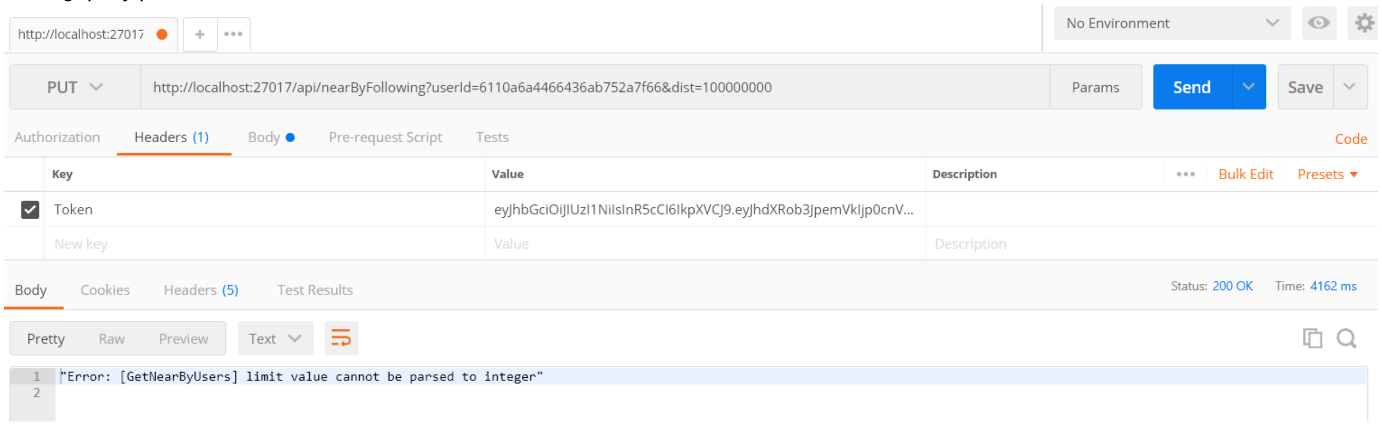
Key	Value	Description
✓ Token	eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJhdXRob3JpemVkIjp0cnV...	
New key	Value	Description

Status: 200 OK Time: 5752 ms

```
[{"_id":"6110ac31475fc7985d586bea","address":"Boon Lay","createdAt":"2021-08-09 12:16:47.1725204 +0800 +08 m=+24.078247501","description":"this is some dummy description"
    ,"dob":"21 Aug 2001","followers":["6110a6a4466436ab752a7f66"],"following":["6110a6a4466436ab752a7f66"],"latitude":0.1,"longitude":0.1,"name":"James"}]
```

Unsuccessful Request

Missing query parameter



PUT http://localhost:27017/api/nearByFollowing?userId=6110a6a4466436ab752a7f66&dist=100000000

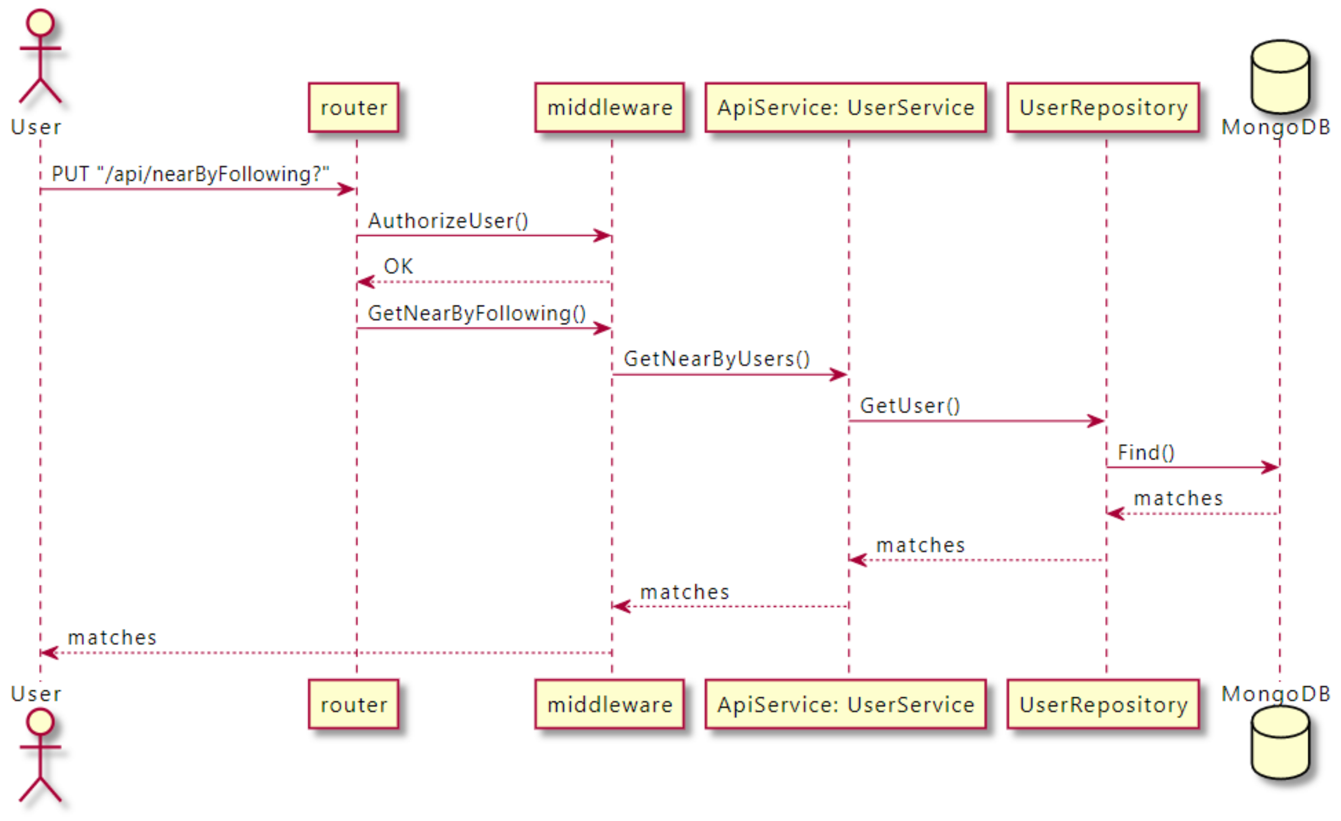
Key	Value	Description
✓ Token	eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJhdXRob3JpemVkIjp0cnV...	
New key	Value	Description

Status: 200 OK Time: 4162 ms

```
"Error: [GetNearByUsers] limit value cannot be parsed to integer"
```

Implementation Details

For success scenario



Code for calculating the distance between 2 lat-long coordinates

```
/**
helper method to get dist in metres between two location points
https://www.nhc.noaa.gov/gccalc.shtml
*/
func getDist(lat1 float64, long1 float64, lat2 float64, long2 float64)
float64 {
        var distX = (lat1 - lat2) * 111000
        var distY = (long1 - long2) * 111000
        var hyptotenuse = math.Sqrt((distX*distX + distY*distY))
        var distMetres = hyptotenuse * 111000
        return distMetres
}
```

Edge Cases Handled

1. Unauthorized user
2. User does not exist in database
3. Missing parameter
4. Database errors

Logging Strategy

The following library was used for logging: `logrus "github.com/sirupsen/logrus"`

Logging is done at a high level where results and errors are received. Hence, logging is mostly implemented in `middleware.go` unless there was a need to do lower-level logging.

Log results are printed to the server stdout in JSON format for easy parsing of logs if needed in the future.

All requests will be logged in the Auth layer as well (even for failed requests). This serves to be a complete logging strategy.

Setting up the logger

```
func enableLogging(flag bool) {
	if flag {
		logrus.SetOutput(os.Stdout)
		logrus.SetFormatter(&logrus.JSONFormatter{})
		logLevel, err := logrus.ParseLevel("debug")
		if err != nil {
			logLevel = logrus.InfoLevel
		}
		logrus.SetLevel(logLevel)
	}
}
```

This method is called with `true` in `init()` method in `middleware.go`

Testing

The following libraries were used for testing:

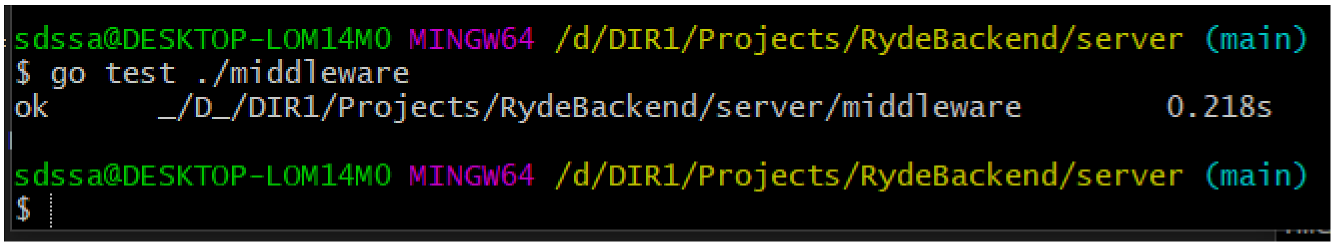
```
    "github.com/stretchr/testify/assert"
    "github.com/stretchr/testify/mock"
```

During testing, it was important that test data was not pushed to production database. "`github.com/stretchr/testify/mock`" library allowed for the creation of a mock repository which served as a test layer which interacts between the code logic and a mock database.

Unit Tests are implemented in `server/middleware/user-service_test.go`.

To run all the unit tests, the following can be run in the `server` directory:

```
$ go test ./middleware
```



```
sdssa@DESKTOP-LOM14M0 MINGW64 /d/DIR1/Projects/RydeBackend/server (main)
$ go test ./middleware
ok      _/D_/DIR1/Projects/RydeBackend/server/middleware        0.218s

sdssa@DESKTOP-LOM14M0 MINGW64 /d/DIR1/Projects/RydeBackend/server (main)
$
```