Technical Documentation

Abstract

The purpose of this document is to provide technical details for the JSON API project. It contains the information about the technology used to create JSON API, deployment procedure and architecture diagram.

Keywords: JSON API, Rails 5, AWS

1. Introduction

1.1 Purpose of the document

The document contains the technical and behavioural information about the JSON API created in Ruby on Rails. This document is created to provide a complete overview of the product to its reader. This will also help new developers to get the insight about the REST API.

1.2 Target Audience

The target audience are all the developers associated with the product, technical project manager, team leader, CTO and CEO

2. Project Description

2.1 Technology Used

Language: Ruby 2.2.5
Framework: Rails 5
Database: Postgres
Version Control: Github
Environment: Development Ubuntu 14 & Production Ubuntu 16 (AWS)
Text Editor: Sublime Text
Diagram: Creately
Documentation: Google Docs.
API Testing: Postman

2.2 Setup Process

2.2.1 Development

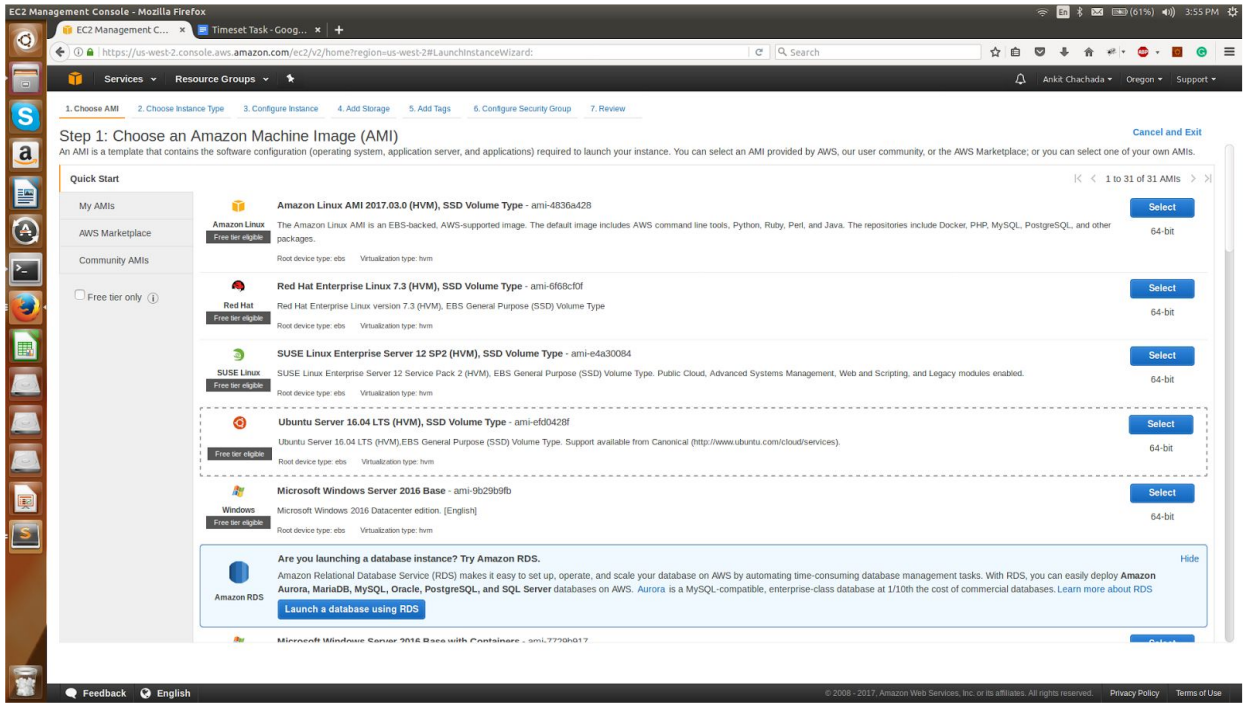
rails new blog --api -no-sprockets -d postgresql
Added few gems and ran bundle install
rails g model post user_id:integer content:text
rails g devise:install
rails g devise User
rake db:migrate
Created appropriate controllers under controllers/api/v1

2.2.2 Production

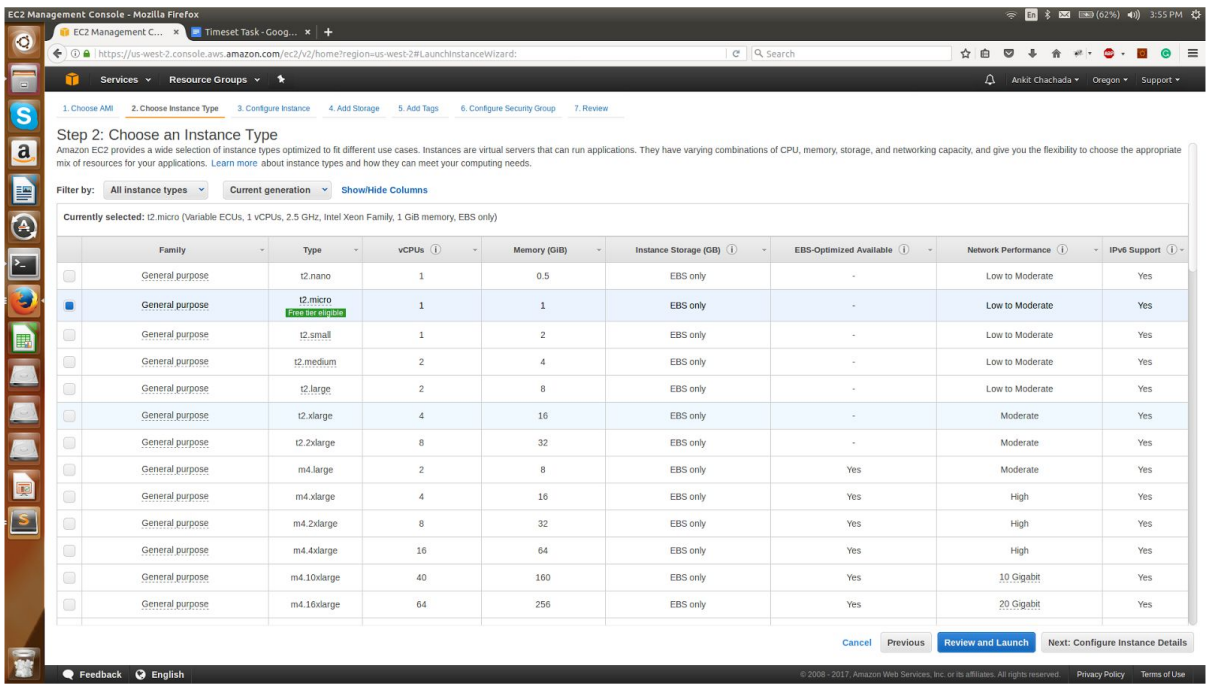
Created free account on Amazon.

1. Choose an Amazon Machine Image (AMI): I choose Ubuntu as my development

machine was also ubuntu

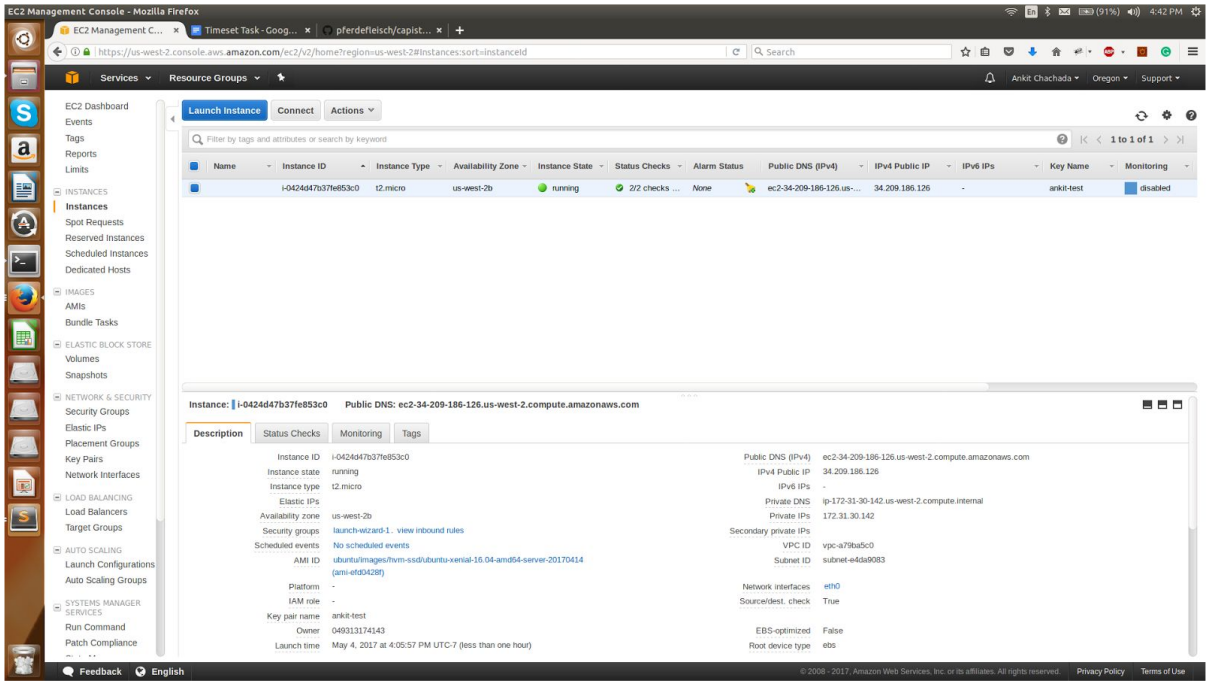


2. Instance Type: Selected t2.micro as it comes for free

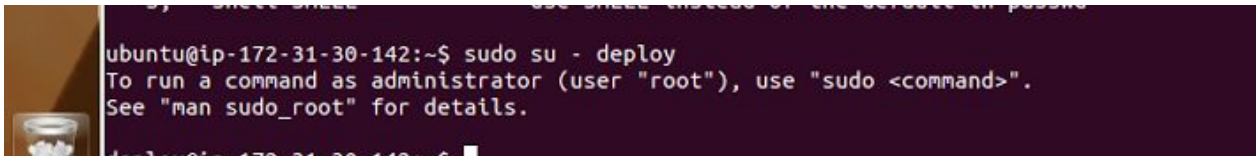


Family	Type	vCPUs	Memory (GiB)	Instance Storage (GB)	EBS-Optimized Available	Network Performance	IPv6 Support
General purpose	t2.nano	1	0.5	EBS only	-	Low to Moderate	Yes
General purpose	t2.micro (Free tier eligible)	1	1	EBS only	-	Low to Moderate	Yes
General purpose	t2.small	1	2	EBS only	-	Low to Moderate	Yes
General purpose	t2.medium	2	4	EBS only	-	Low to Moderate	Yes
General purpose	t2.large	2	8	EBS only	-	Low to Moderate	Yes
General purpose	t2.xlarge	4	16	EBS only	-	Moderate	Yes
General purpose	t2.2xlarge	8	32	EBS only	-	Moderate	Yes
General purpose	m4.large	2	8	EBS only	Yes	Moderate	Yes
General purpose	m4.xlarge	4	16	EBS only	Yes	High	Yes
General purpose	m4.2xlarge	8	32	EBS only	Yes	High	Yes
General purpose	m4.4xlarge	16	64	EBS only	Yes	High	Yes
General purpose	m4.10xlarge	40	160	EBS only	Yes	10 Gigabit	Yes
General purpose	m4.16xlarge	64	256	EBS only	Yes	20 Gigabit	Yes

3. Did the remaining process, created pem file and launched the instance.



4. Open terminal and ran ssh -i "ankit-test.pem" ubuntu@34.209.186.126, created new user deploy. (Note: need to give permission to ankit-test.pem file by chmod 400)



.

5. sudo useradd -d /home/deploy -m deploy
6. Added my local ssh key to the server and server ssh key to my github account
7. Install RVM, Ruby, Rails, Postgres, Node.js(Execjs error), Git, Ngnix.
8. Created postgres user and database for the project
9. Created github repository and linked to my local repository
10. Added following gems to Gemfile

```
group :development do
  gem 'capistrano', '3.5.0'
  gem 'capistrano-rvm'
  gem 'capistrano-nginx'
  gem 'capistrano3-puma'
  gem 'capistrano-rails'
  gem 'capistrano-rails-db'
  gem 'capistrano-rails-console'
  gem 'capistrano-upload-config'
  gem 'sshkit-sudo'
end
```

11. bundle install

12. ‘cap install STAGES=production’ for setting production environment
13. Added setting in Capfile, deploy.rb and config/deploy/production.rb
14. Added settings in Ngnix sites available default file
15. Created a shared folder with config under home/deploy/blog(project directory)
16. ‘cap production deploy’ to deploy project from my github repo to server machine

2.3 Gems and Libraries used

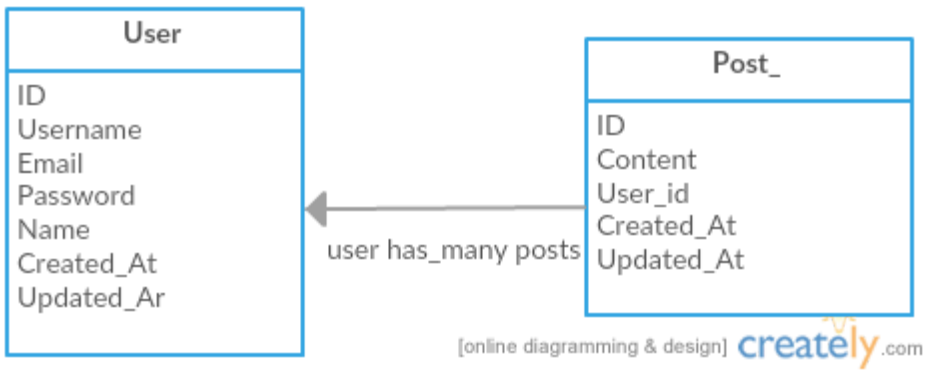
1. Devise: For user sign up/sign in I have used devise.
2. Simple_token_authentication: For auth token
3. Kaminari: For pagination
4. Pundit: For authorization
5. Pg : For postgres
6. Active_model_serializers: For serializing the objects
7. For deployment

```
gem 'capistrano', '3.5.0'
 gem 'capistrano-rvm'
 gem 'capistrano-nginx'
 gem 'capistrano3-puma'
 gem 'capistrano-rails'
 gem 'capistrano-rails-db'
 gem 'capistrano-rails-console'
 gem 'capistrano-upload-config'
```

8. Rack-cors for cors

2.4 Rails Model

User & Post model



We have a relationship between user and post. I have also added dependent destroy. If a user is deleted all the post associated with him will also get deleted.

2.5 Schema

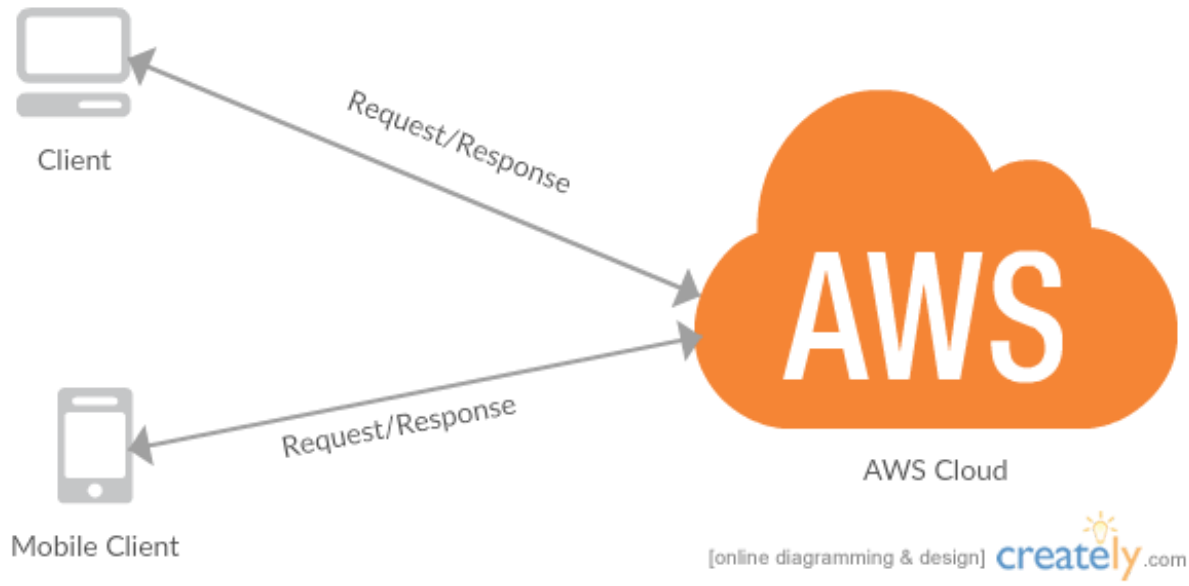
```
create_table "posts", force: :cascade do |t|
```

```
      t.integer "user_id"
      t.text "content"
      t.datetime "created_at", null: false
      t.datetime "updated_at", null: false
  end

  create_table "users", force: :cascade do |t|
      t.string "name", default: "", null: false
      t.string "username", default: "", null: false
      t.string "email", default: "", null: false
      t.string "encrypted_password", default: "", null: false
      t.string "reset_password_token"
      t.datetime "reset_password_sent_at"
      t.datetime "remember_created_at"
      t.integer "sign_in_count", default: 0, null: false
      t.datetime "current_sign_in_at"
      t.datetime "last_sign_in_at"
      t.inet "current_sign_in_ip"
      t.inet "last_sign_in_ip"
      t.datetime "created_at", null: false
      t.datetime "updated_at", null: false
      t.string "authentication_token", limit: 30
      t.index ["authentication_token"], name:
"index_users_on_authentication_token", unique: true
      t.index ["email"], name: "index_users_on_email", unique: true
      t.index ["reset_password_token"], name:
"index_users_on_reset_password_token", unique: true
  end
```

2.6 Architecture

2.6.1 Abstract View

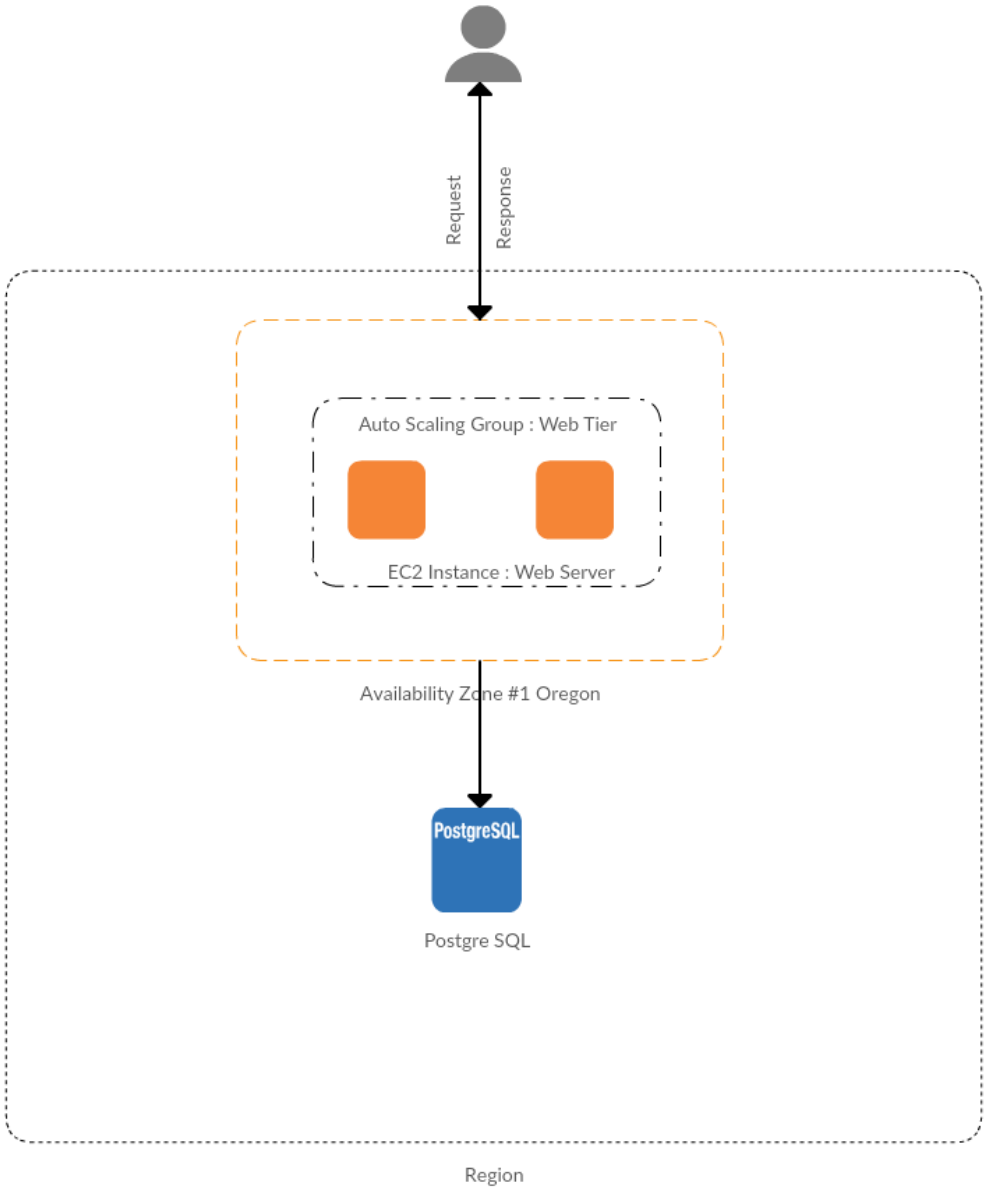


Above is the abstract view of the service used: Clients can request and get response from the AWS cloud. The application and the database are residing on AWS Cloud.

2.6.2 Full Architecture

Currently I have created only a single instance, since the application is very small. The application code is residing on that instance. I have also setup postgresql on the server and created local database for this project. A user here is any type of a client which will request and will get response from the server. The server will fetch the data from the database and render the response in a JSON format.

Currently I am using t2.micro and there are no graphics involved. So it depends how much data is being used. T2.micro comes with 1GiB memory, so assume that we have 900 MB and one user takes 1 mb then 900/1 = 900 users can be served concurrently.

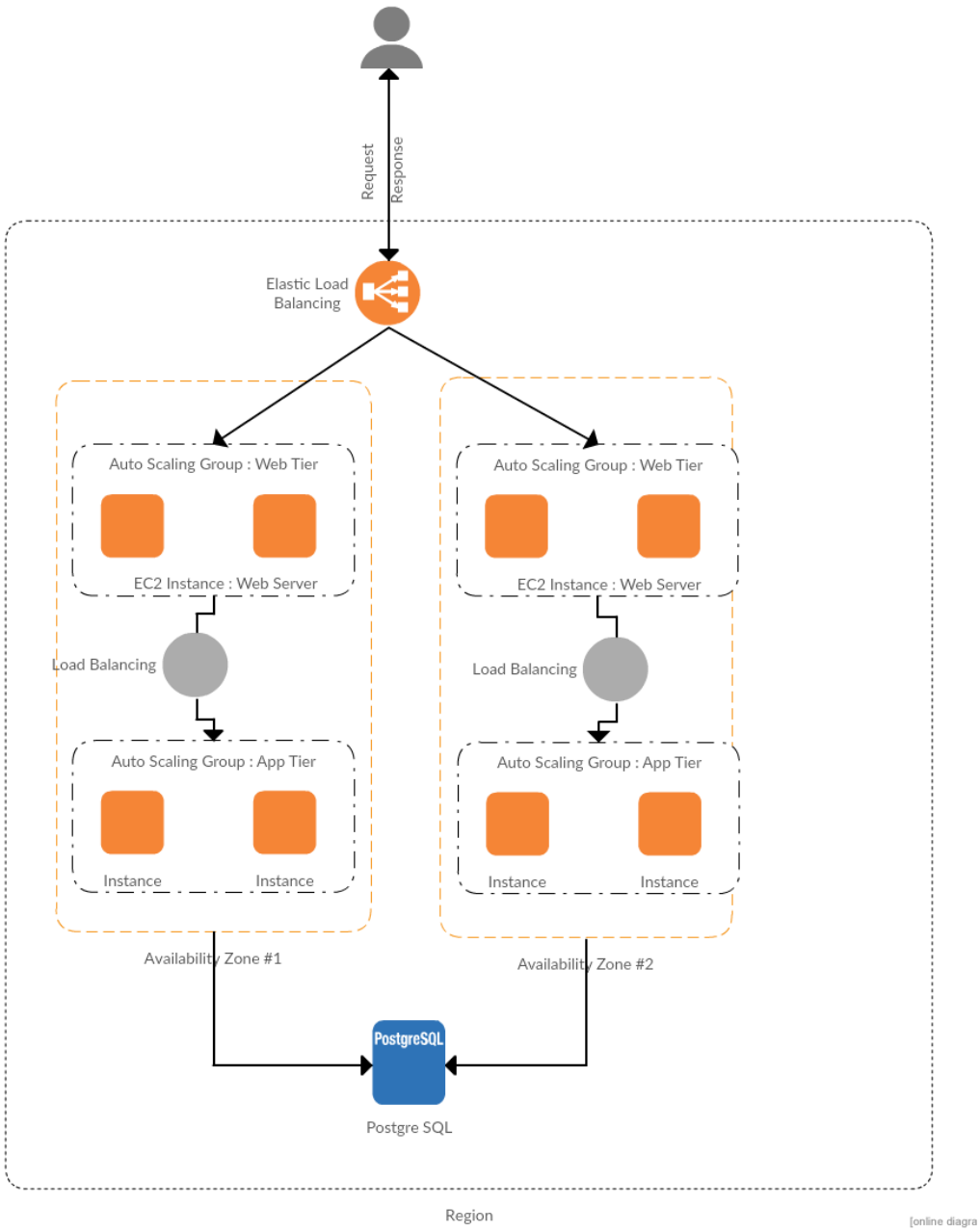


2.6.3 Future Scope

Amazon provides us Elastic Load Balancer. “Elastic Load Balancing distributes incoming application traffic across multiple EC2 instances, in multiple Availability Zones. This increases the fault tolerance of your applications. ”[1]

Suppose in future, if we have a media uploading feature and traffic on the API increase dramatically we need to introduce elastic load balancer. Following is the prototype architecture for this kind of feature.

We also need to introduce Amazon RDS to handle such requests.

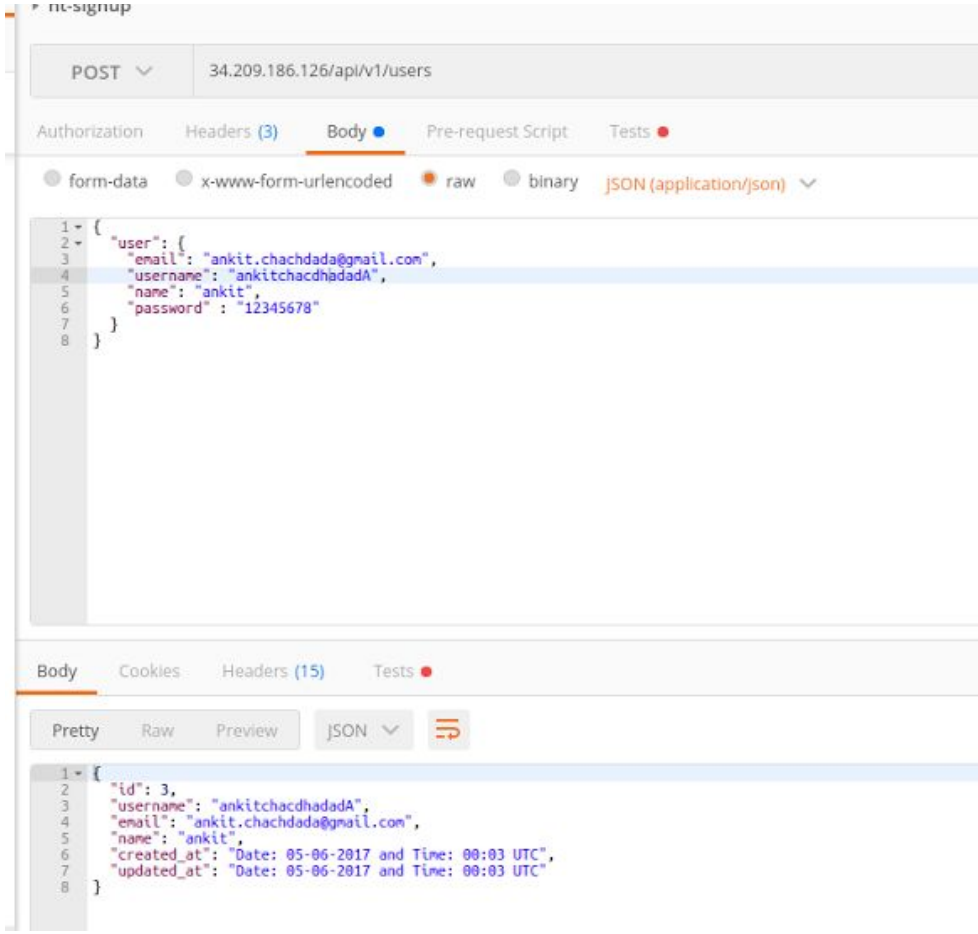


3. Screens and API:

[1] http://docs.aws.amazon.com/elasticloadbalancing/latest/userguide/what-is-load-balancing.html

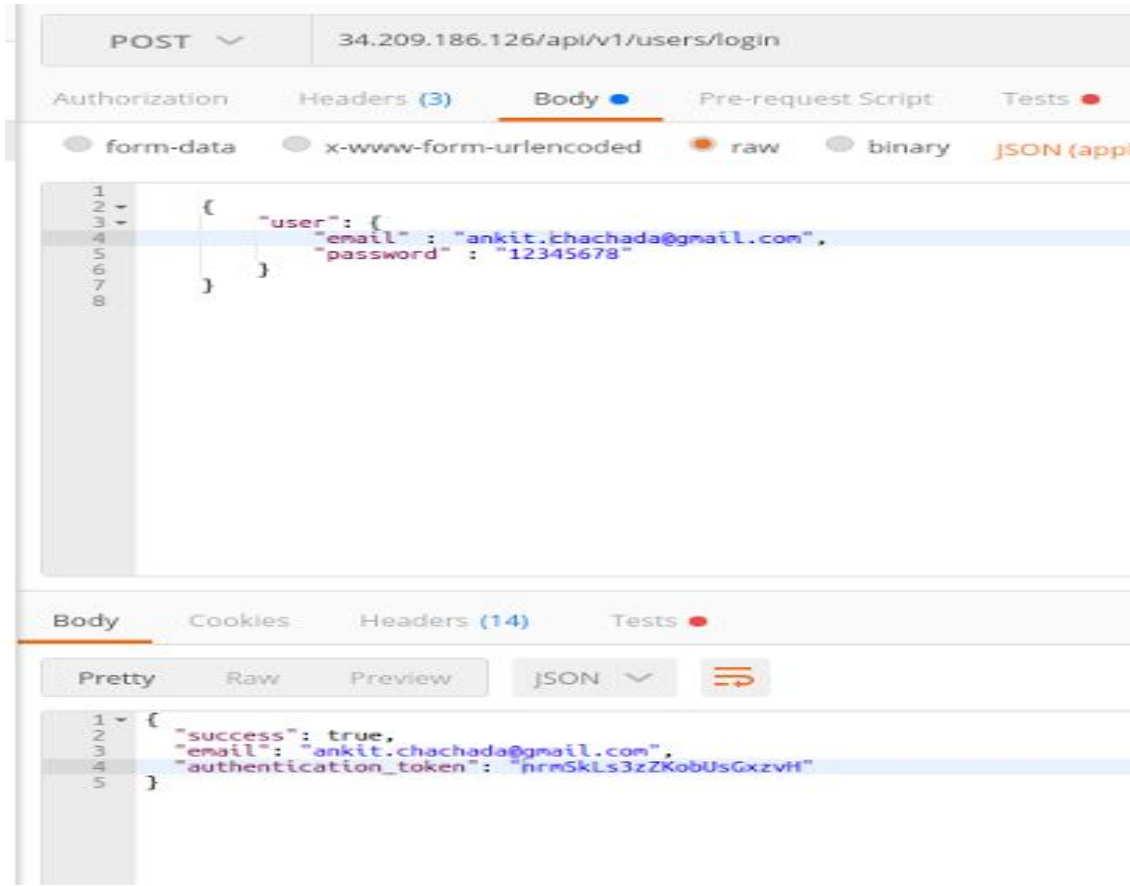
This section contains screenshots of API using Postman. It shows all the CRUD operations.

1. User create



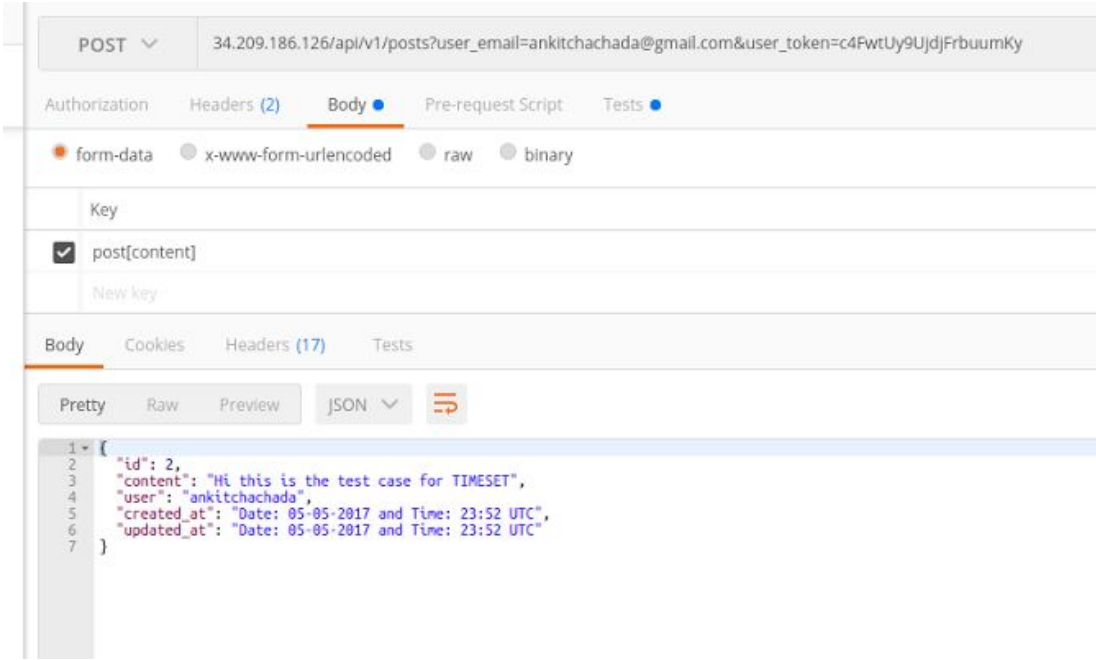
API Endpoint:
api/v1/users
HTTP Method:
Post
JSON Response:
Email, Username, Name, Created At and Updated At

2. Login Screen



API Endpoint:
api/v1/users/login
HTTP Method:
Post
JSON Response:
Email and Authentication Token. These details are used for creating, editing or deleting posts and users.

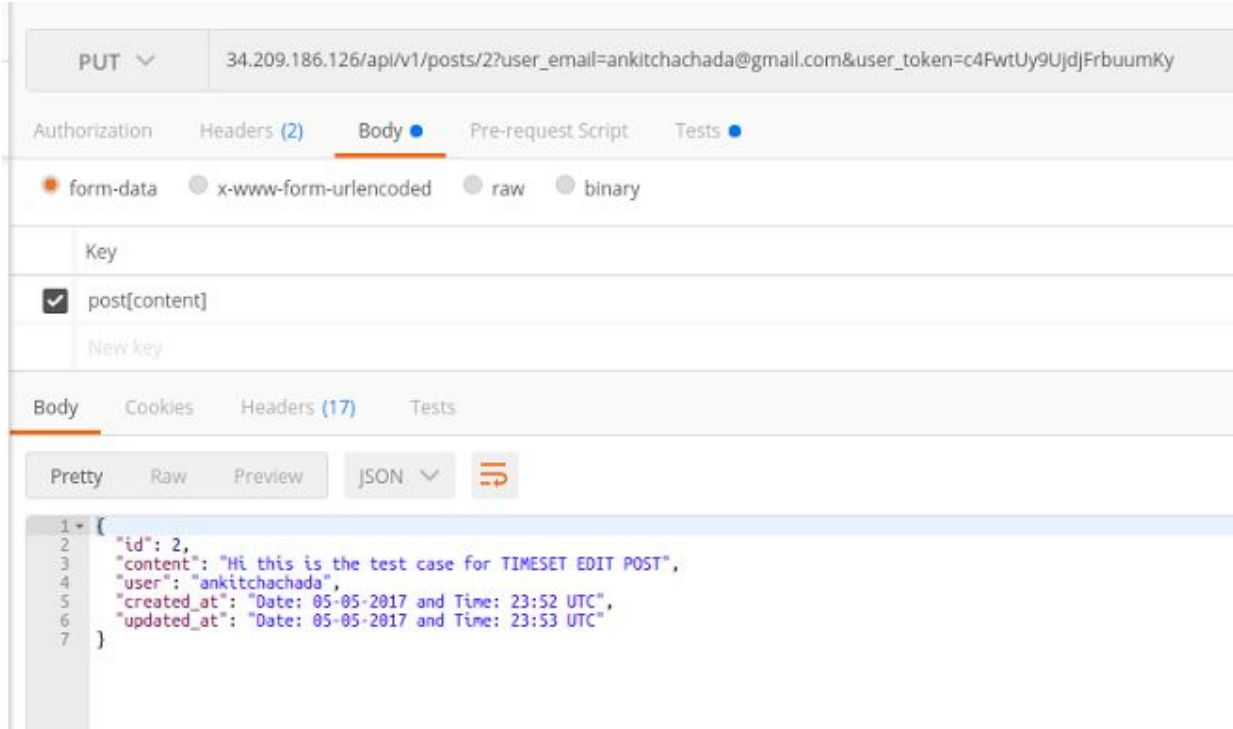
3. Create Post



API Endpoint:
api/v1/posts?user_email=<EMAIL>&user_token=<AUTH TOKEn>
HTTP Method:
Post
JSON Response:
Id, content, user, created_at and updated_at

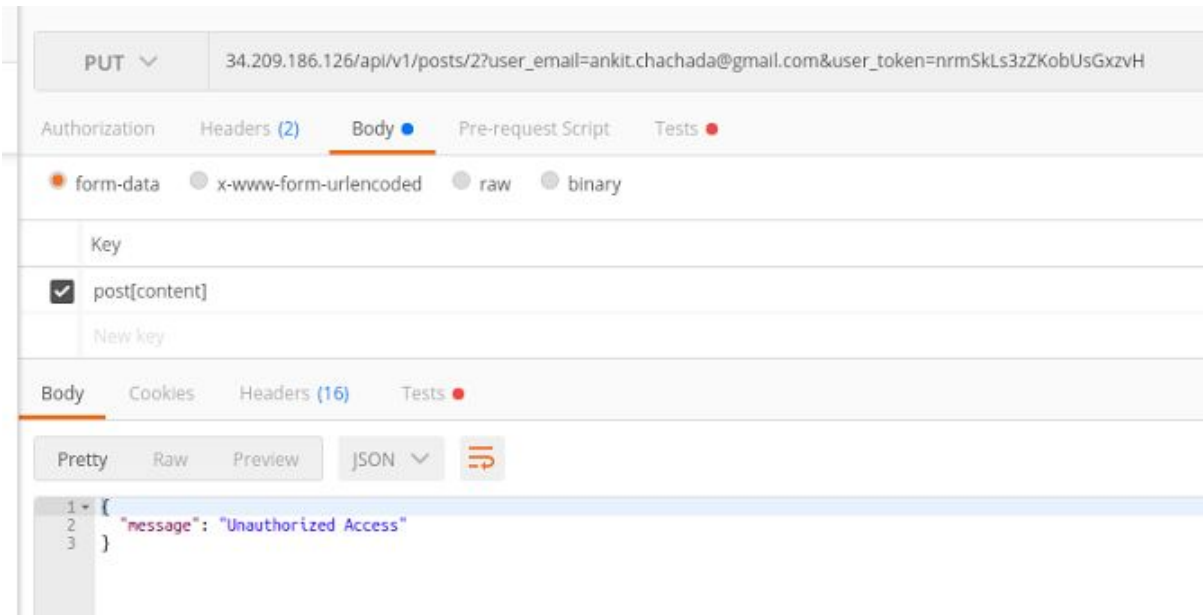
The request for creating a post is authenticated using email and auth token which is received during the sign in process. This mechanism is implemented to ensure user's authenticity using 'simple_token_authentication' gem.

4. Edit Post



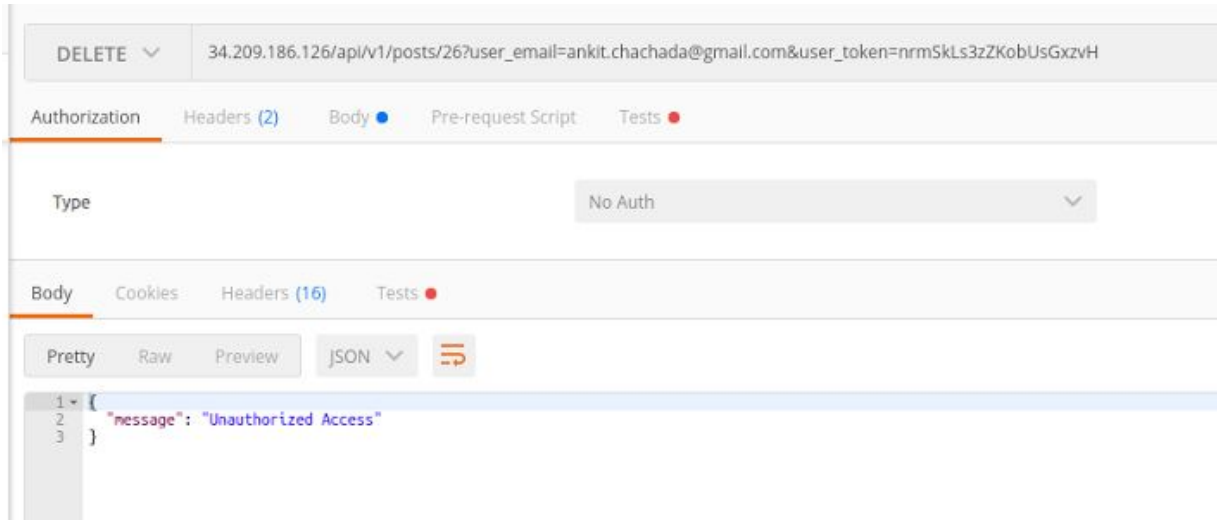
API Endpoint:
api/v1/posts/<ID>?user_email=<EMAIL>&user_token=<AUTH TOKEN>
HTTP Method:
Put
JSON Response:
Id, content, user, created_at and updated_at
The request for editing a post is authenticated using email and auth token which is received during the sign in process. This mechanism is implemented to ensure user's authenticity using 'simple_token_authentication' gem.

5. Unauthorized Access: Edit



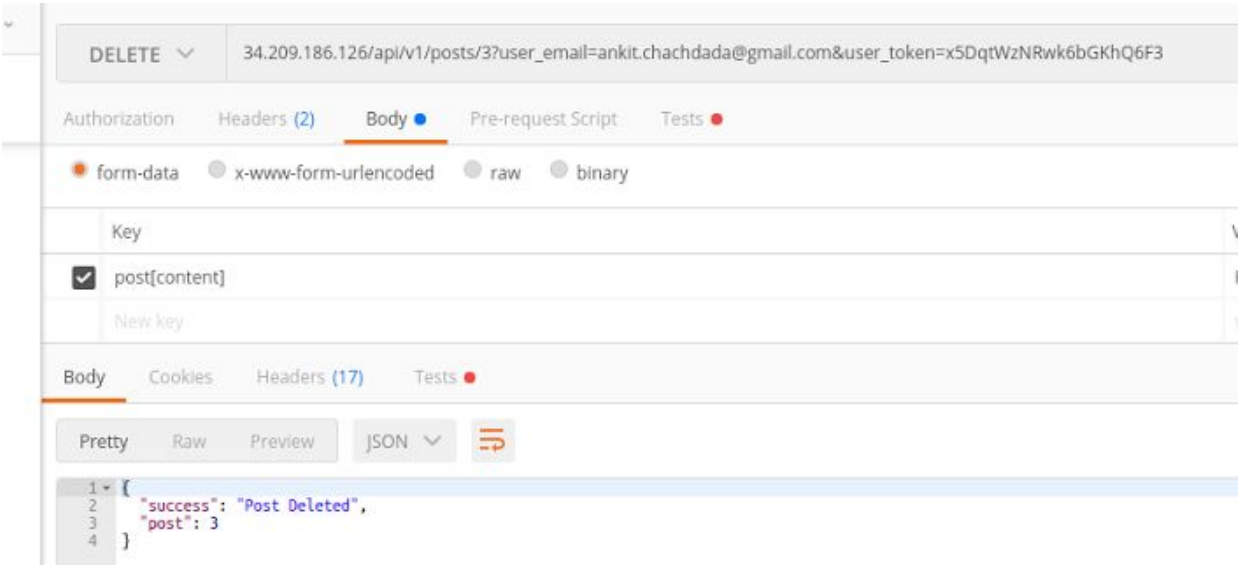
If a user tries to edit a post which he/she has not created will get Unauthorized Access error. For authorization rules I have used Pundit gem and wrote policy for authorization.

6. Unauthorized Access: Delete



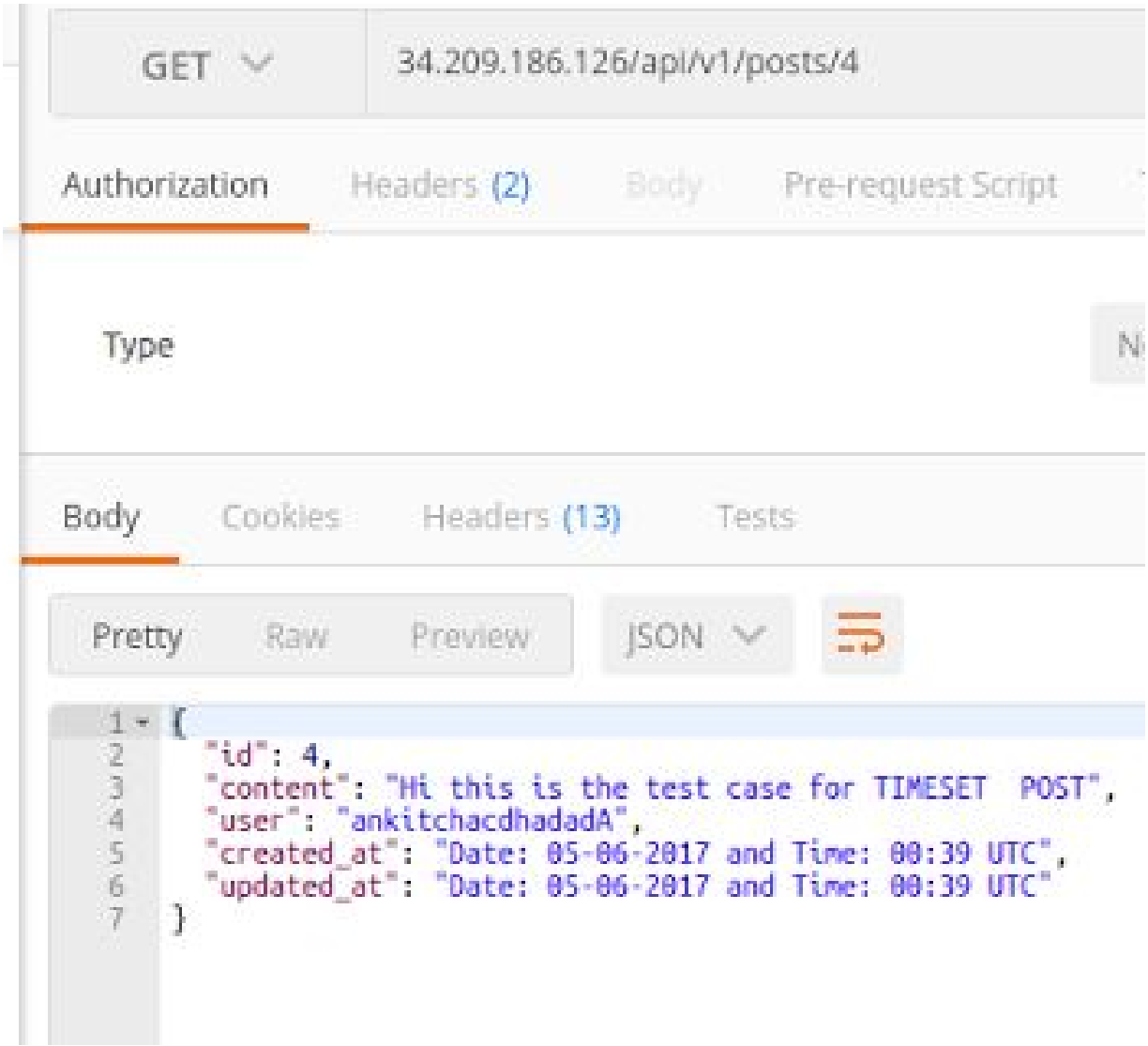
If a user tries to delete a post which he/she has not created will get Unauthorized Access error. For authorization rules I have used Pundit gem and wrote policy for authorization.

7. Delete Post



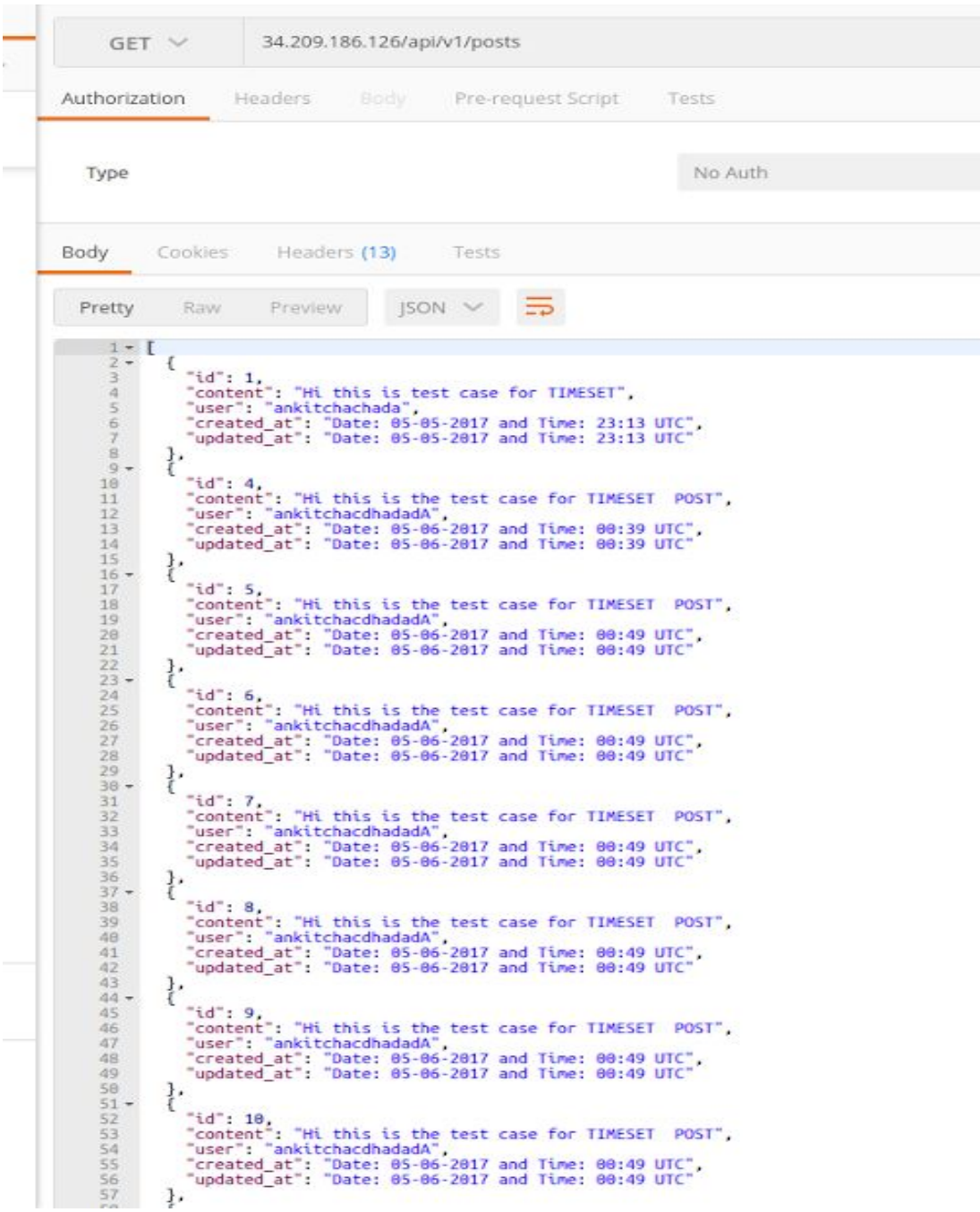
API Endpoint:
api/v1/posts/<ID>?user_email=<EMAIL>&user_token=<AUTH TOKEN>
HTTP Method:
Delete
JSON Response:
Success and Post ID
The request for deleting a post is authenticated using email and auth token which is received during the sign in process. This mechanism is implemented to ensure user's authenticity using 'simple_token_authentication' gem.

8. View Post



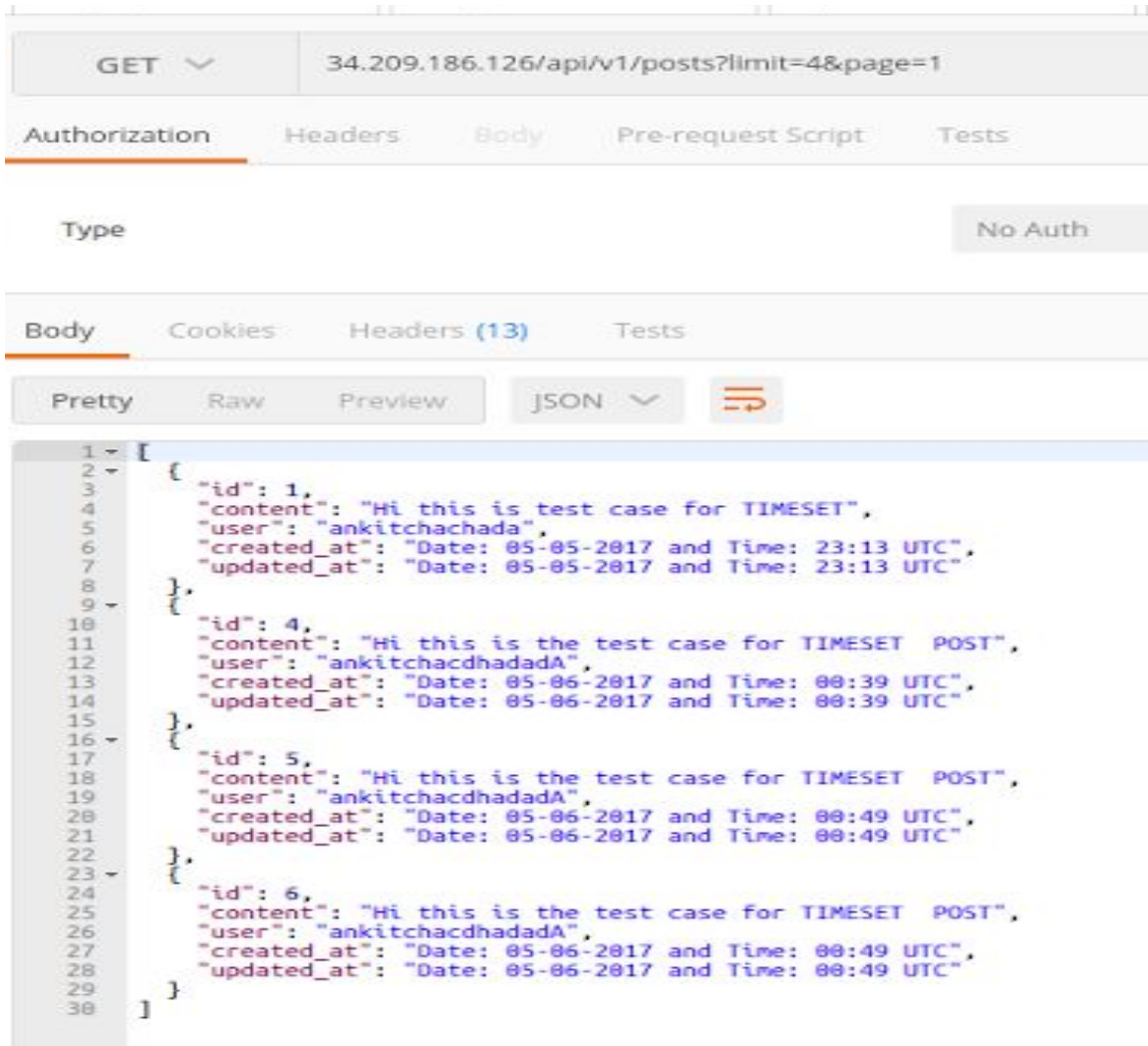
API Endpoint:
api/v1/posts/<ID>
HTTP Method:
Get
JSON Response:
Id, content, user, created_at and updated_at
The request for viewing a post is not authenticated.

9. View All Posts
   9.1 Without Pagination



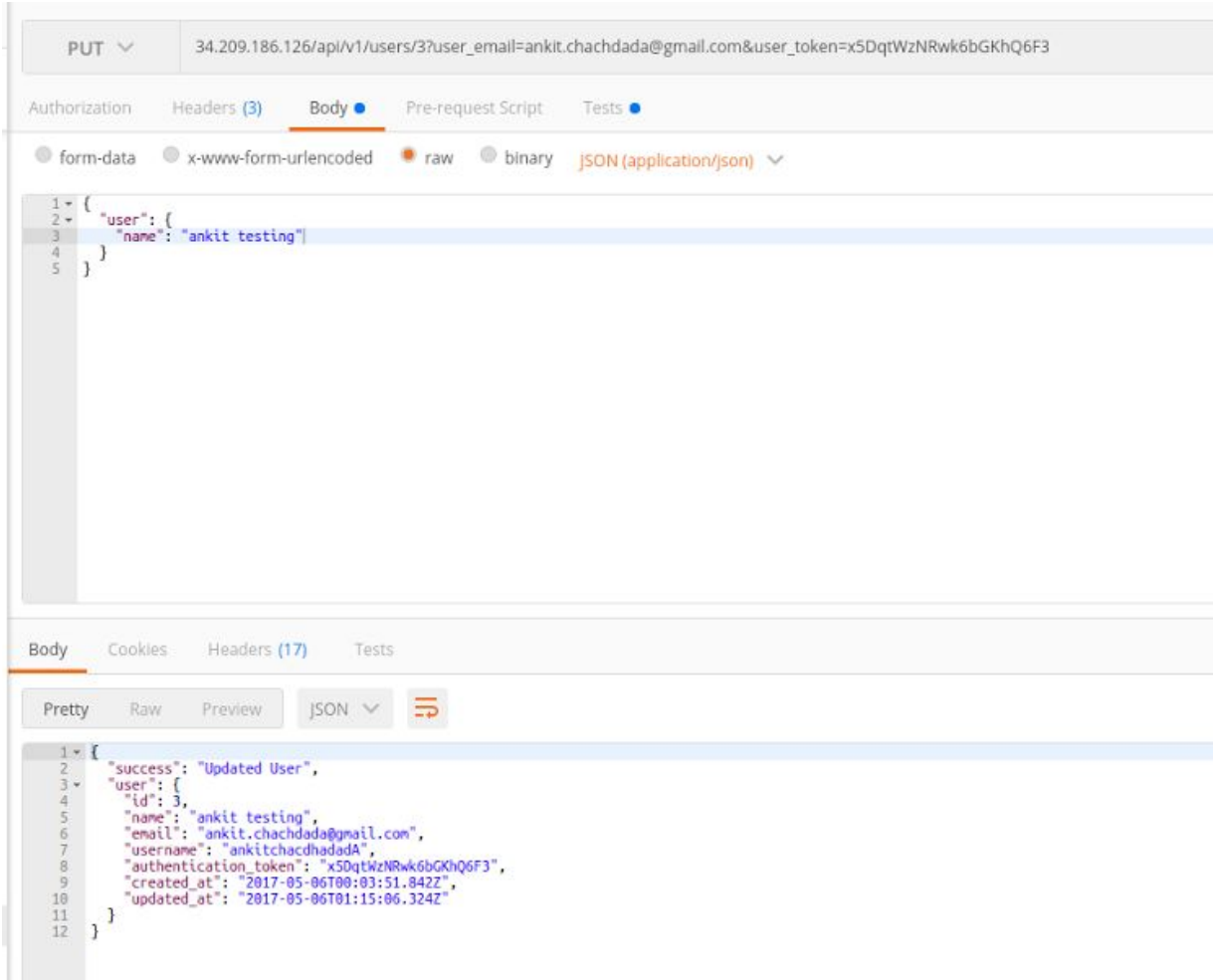
API Endpoint:
api/v1/posts
HTTP Method:
Get
JSON Response:
Id, content, user, created_at and updated_at
This returns all the posts(no authentication)

9.2 With Pagination

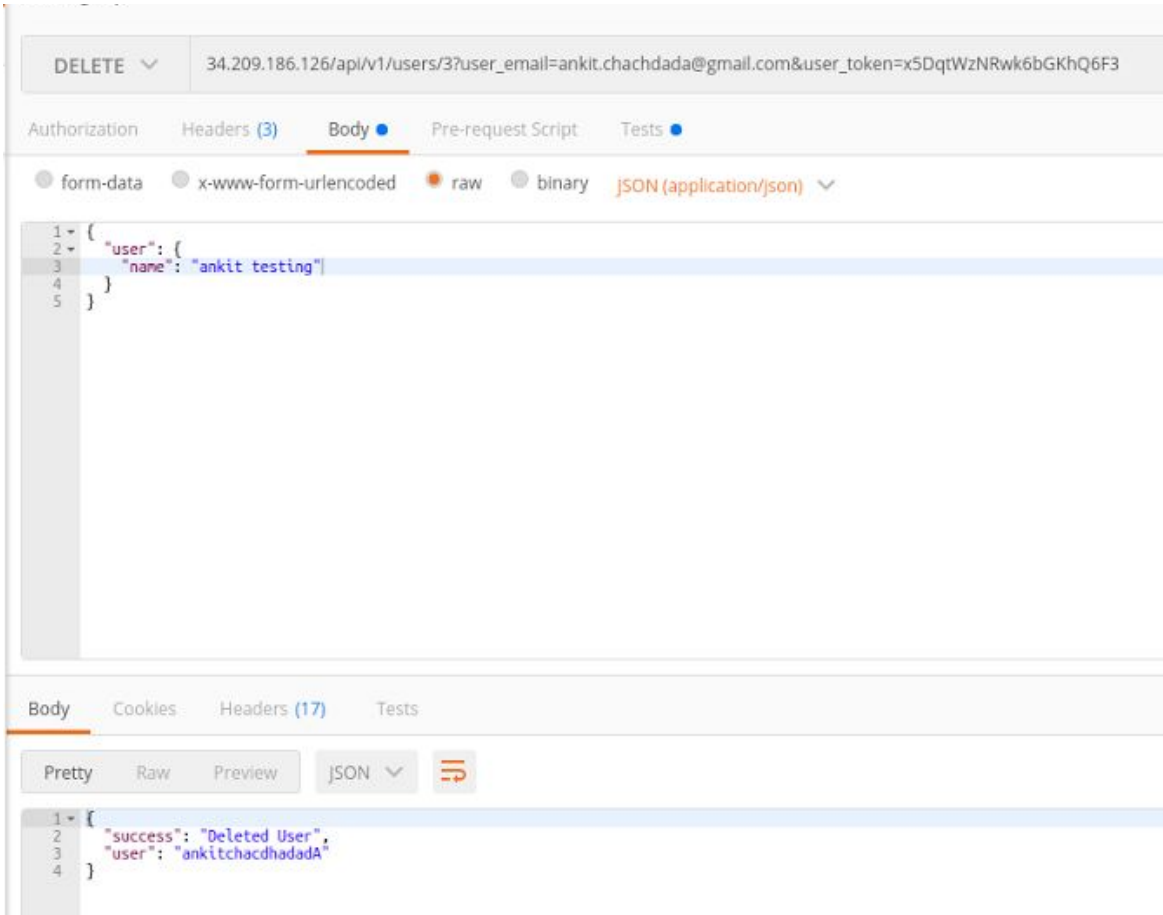


API Endpoint:
api/v1/posts/?limit=<NUMBER OF POSTS>&page=<PAGE NUMBER>
HTTP Method:
Get
JSON Response:
Id, content, user, created_at and updated_at
The returns 4 posts from page 1(no authentication), if limit = 4 and page = 1

10. Edit User


PUT
34.209.186.126/api/v1/users/3?user_email=ankit.chachdada@gmail.com&user_token=x5DqtWzNRwk6bGKhQ6F3
Authorization
Headers (3)
Body
Pre-request Script
Tests
form-data
x-www-form-urlencoded
raw
binary
JSON (application/json)
{
"user": {
"name": "ankit testing"
}
}
Body
Cookies
Headers (17)
Tests
Pretty
Raw
Preview
JSON
{
"success": "Updated User",
"user": {
"id": 3,
"name": "ankit testing",
"email": "ankit.chachdada@gmail.com",
"username": "ankitchacdhadadA",
"authentication_token": "x5DqtWzNRwk6bGKhQ6F3",
"created_at": "2017-05-06T00:03:51.842Z",
"updated_at": "2017-05-06T01:15:06.324Z"
}
}

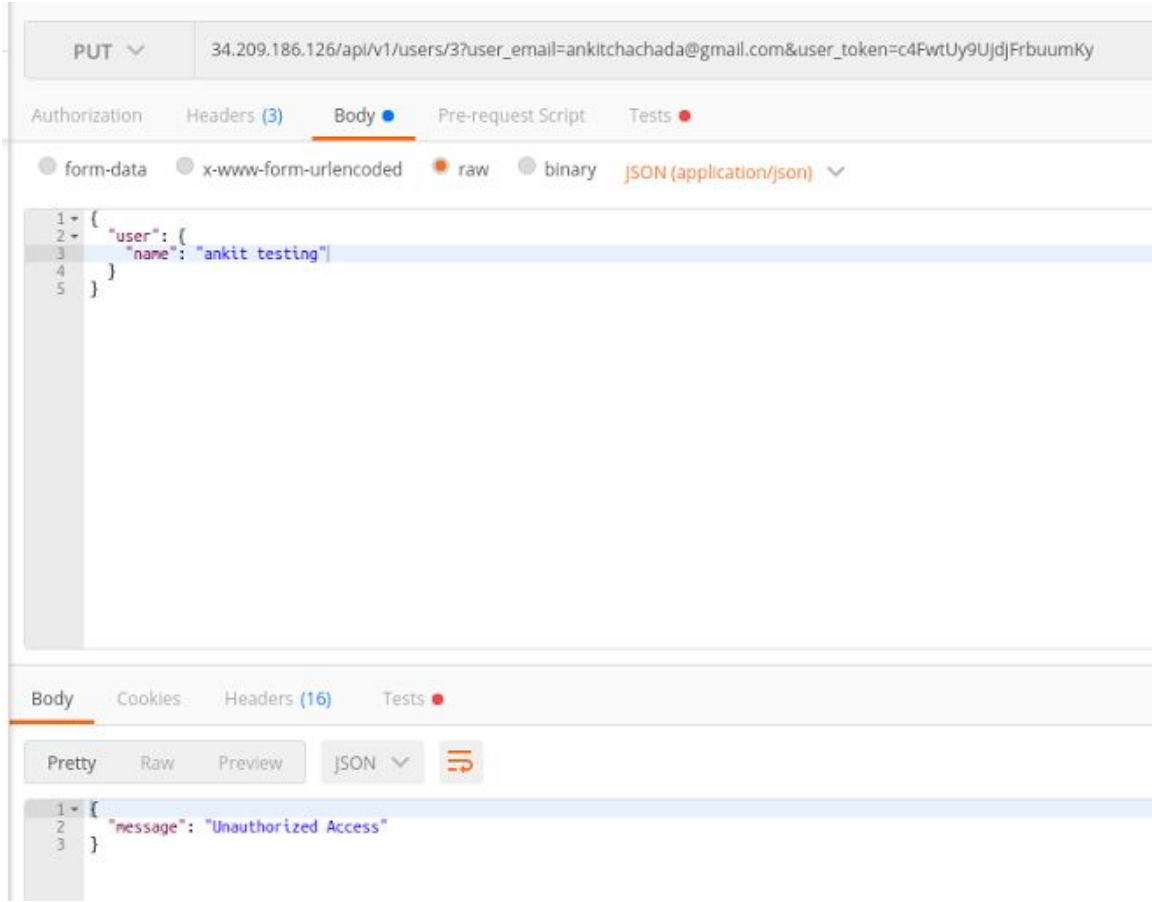
11. Delete User



API Endpoint:
api/v1/users/<ID>?user_email=<EMAIL>&user_token=<AUTH TOKEN>
HTTP Method:
Delete
JSON Response:
Success and username

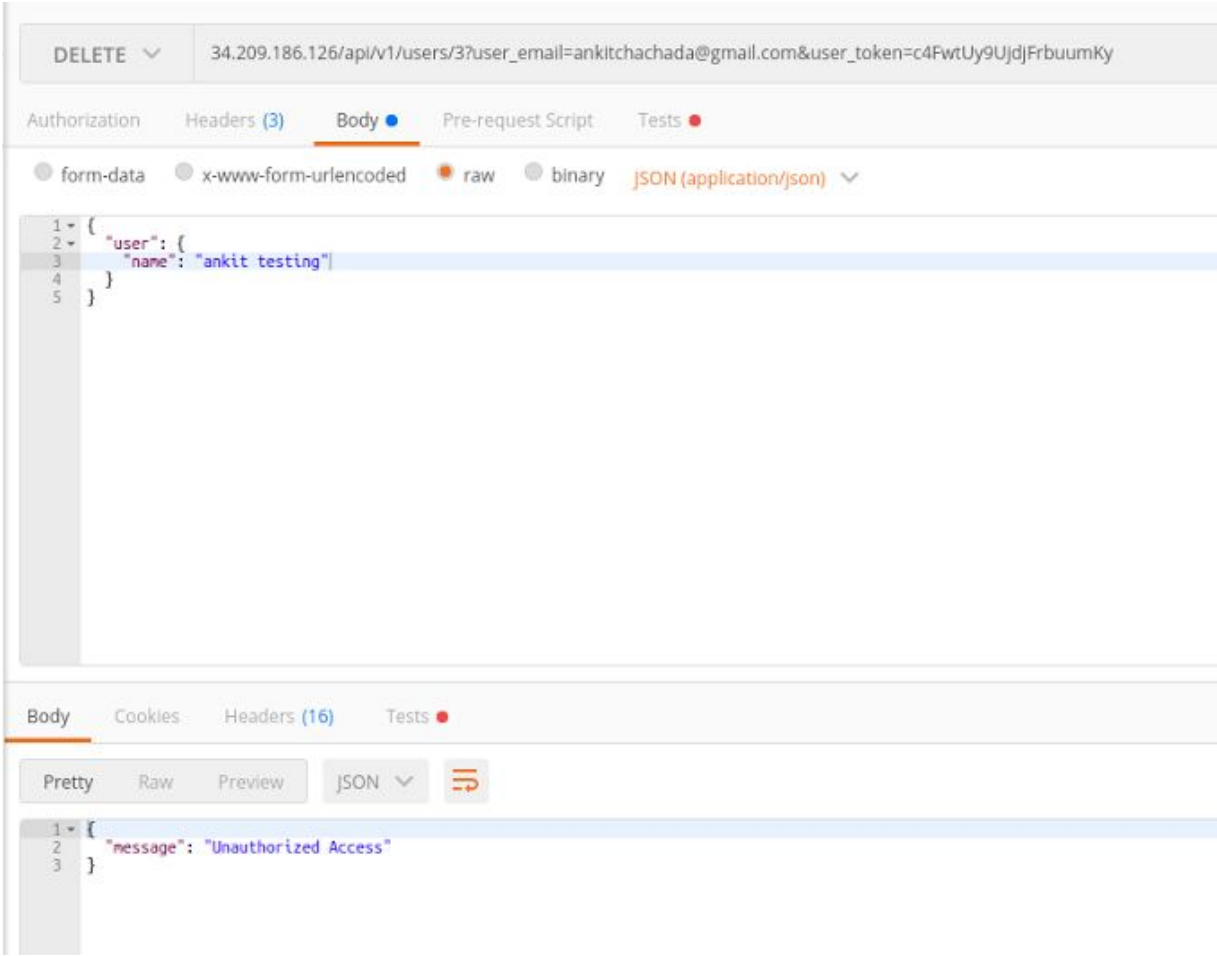
The request for deleting a post is authenticated using email and auth token which is received during the sign in process. This mechanism is implemented to ensure user's authenticity using 'simple_token_authentication' gem.

12. Unauthorized Edit user



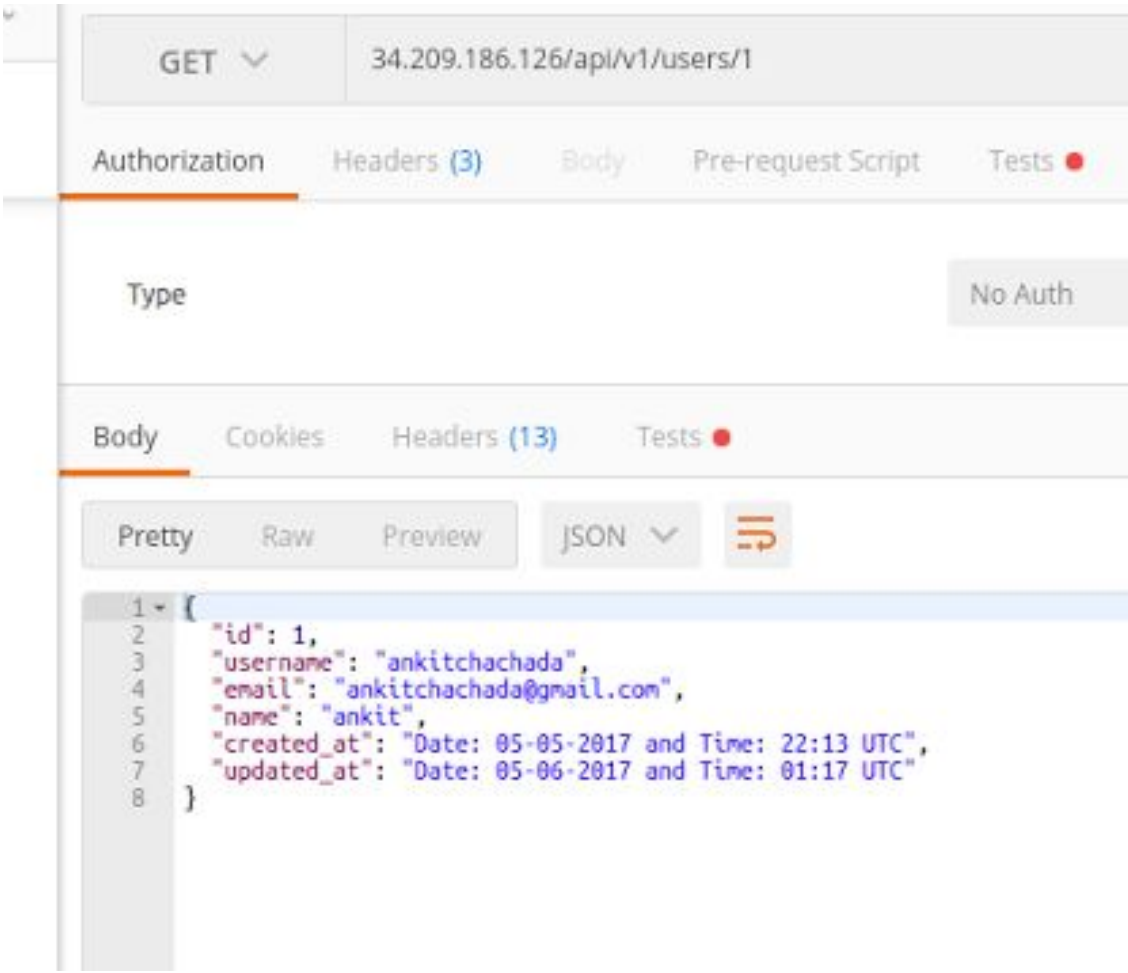
If a user tries to edit other user's detail he will get unauthorized error.

13. Unauthorized Delete user



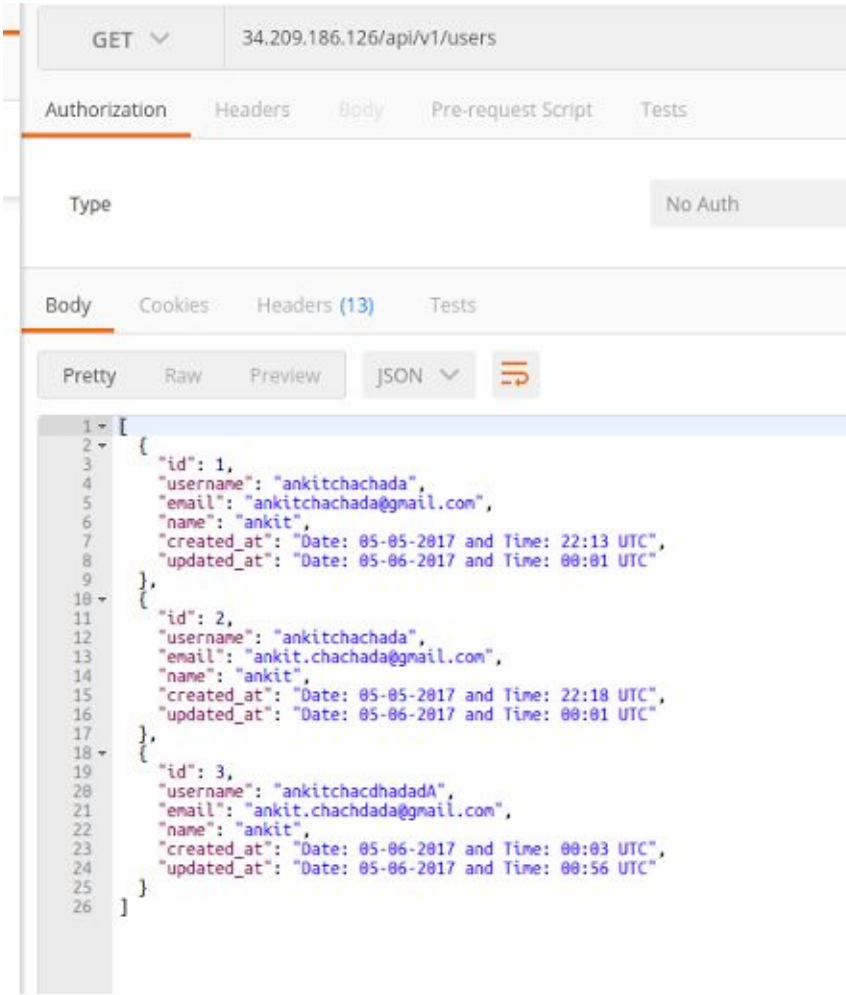
If a user tries to delete other user he will get unauthorized error.

14. View User



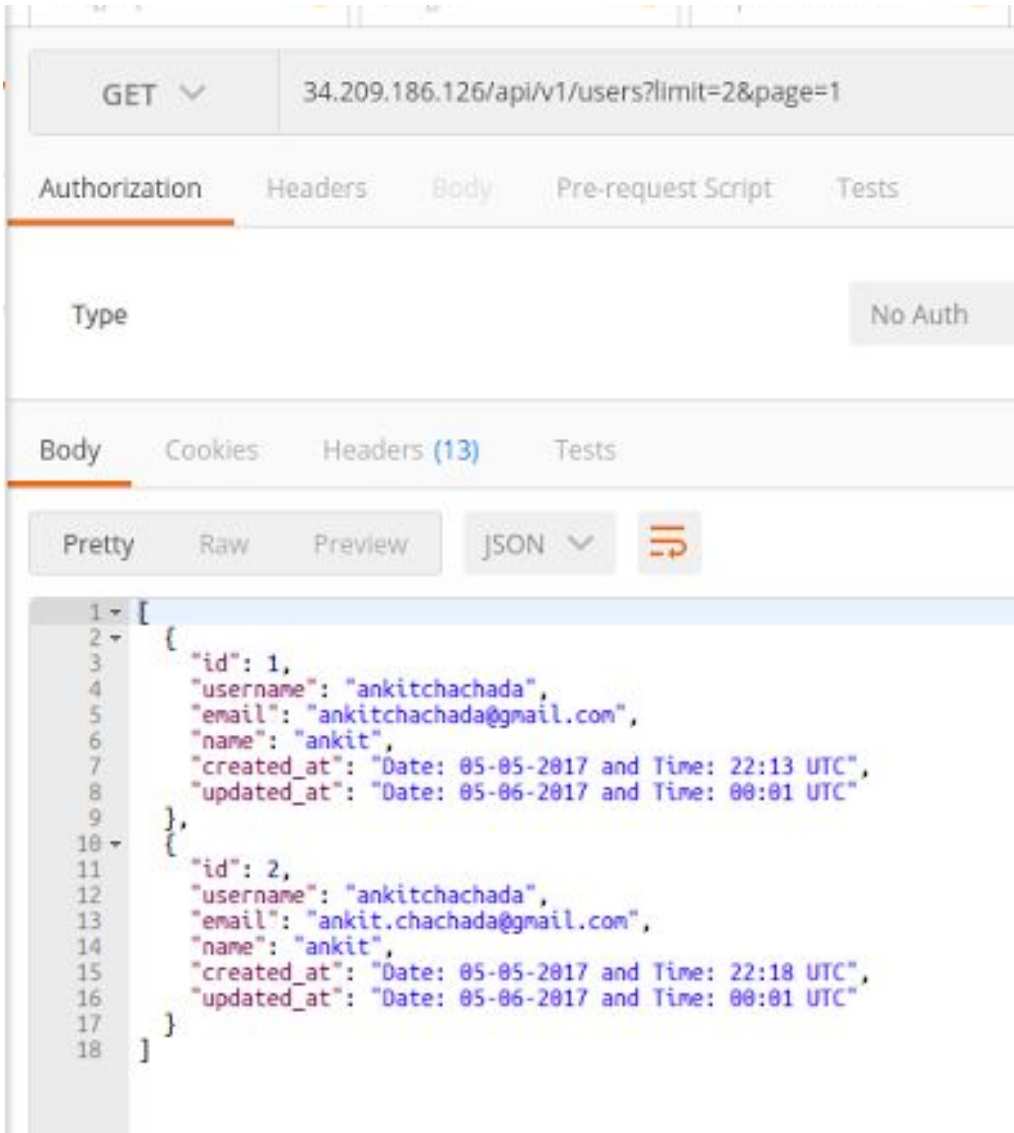
API Endpoint:
api/v1/users/<ID>
HTTP Method:
Get
JSON Response:
Id, username, name, email, created_at and updated_at
The request for viewing a user is not authenticated.

15. View All Users
    13.1 Without Pagination



API Endpoint:
api/v1/users
HTTP Method:
Get
JSON Response:
Id, username, email, name,created_at and updated_at
This returns all the users(no authentication)

13.2 With Pagination



API Endpoint:

api/v1/users/?limit=<NUMBER OF USERS>&page=<PAGE NUMNER>

HTTP Method:

Get

JSON Response:

Id, username, email, name,created_at and updated_at

The returns 2 posts from page 1(no authentication), if limit = 2 and page = 1