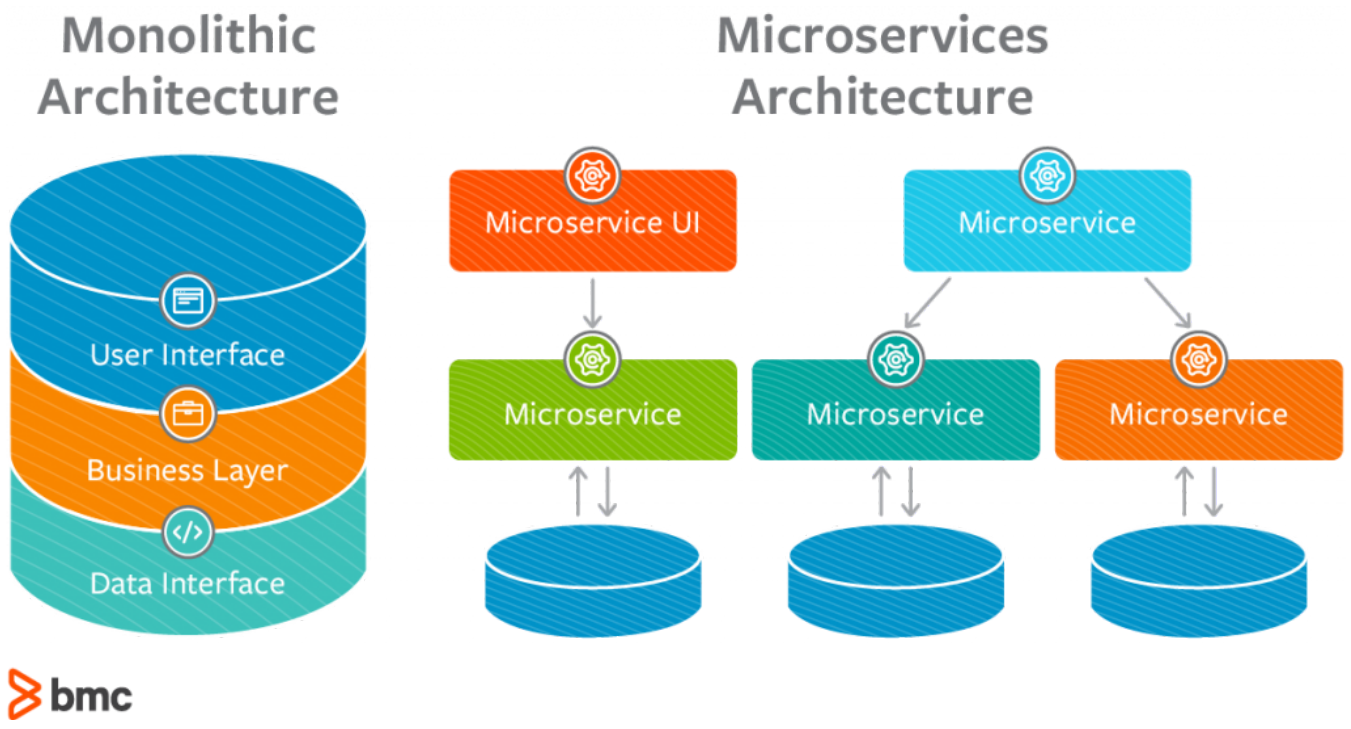


Microservice Architecture

08.20.2020

Kevin Jia
Intern - Summer 2020
VideoPoints LLC

Overview

VideoPoints is an educational media startup that aims to index university lecture videos, as well as caption the speech and make the lecture videos searchable. VideoPoints allows students to search lectures by topic, and also allows students to search specific terms within lecture videos. VideoPoints has expanded to many courses at the University of Houston, including many introductory science classes and upper division computer science courses. In the future, VideoPoints hopes to add more features, including a summarization feature for videos, a business model that emphasizes low costs for students, and expansion beyond the University of Houston.

The current architecture of the application is a monolithic kernel. It is written in php and not scalable, which means it will have difficulty adding more users.The proposal focuses on converting it to microservice architecture, which is an architectural style that structures an application as a collection of services.
These services should be:

- Highly maintainable and testable
- Loosely coupled
- Independently deployable
- Organized around business capabilities
- Owned by a small team

Goals

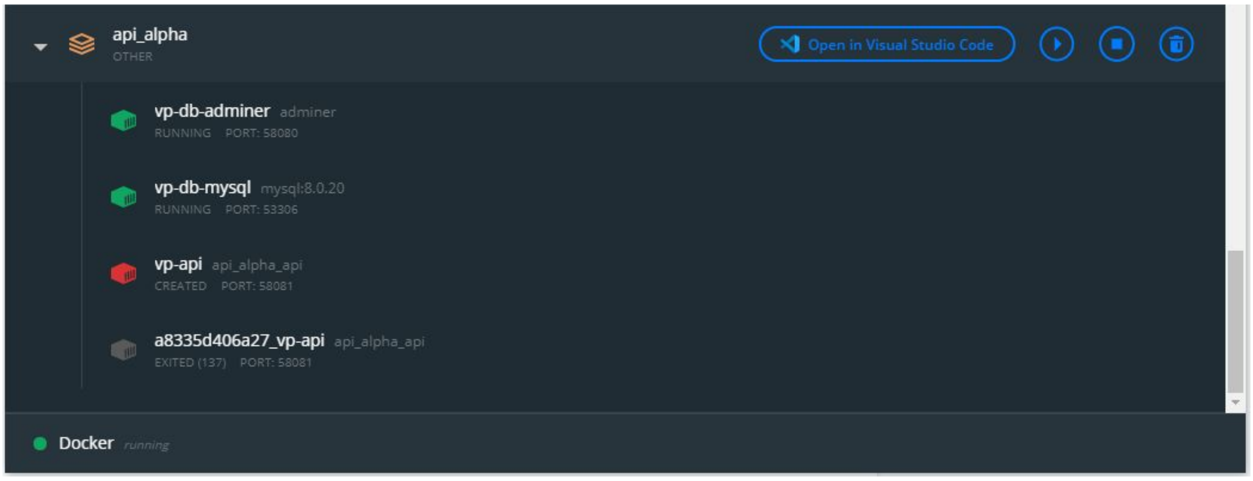
1. **CONTAINERIZE:** Use Docker containers
2. **UPGRADE:** Use latest version of MySQL database 8.0.20
3. **CONFIGURABLE:** Use Adminer, a web based management for MySQL server
4. **READABILITY:** Use Python as a programming language
5. **FLEXIBLE:** Use Flask as the web framework
6. **MICROSERVICES:** Use RESTful API
7. **TESTABILITY:** Use Postman to test the API
8. **VERSIONING:** Use Git for version control

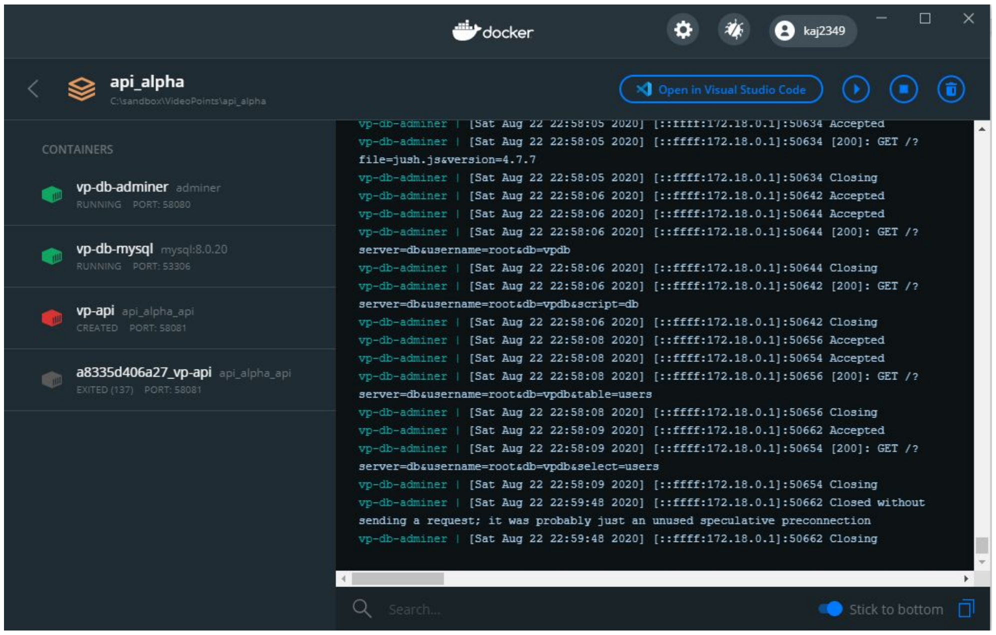
Milestones

I. Use Docker containers

Docker is a set of platform as a service products that use OS-level virtualization to deliver software in packages called containers. Containers are isolated from one another and bundle their own software, libraries and configuration files; they can communicate with each other through well-defined channels.

https://www.docker.com/





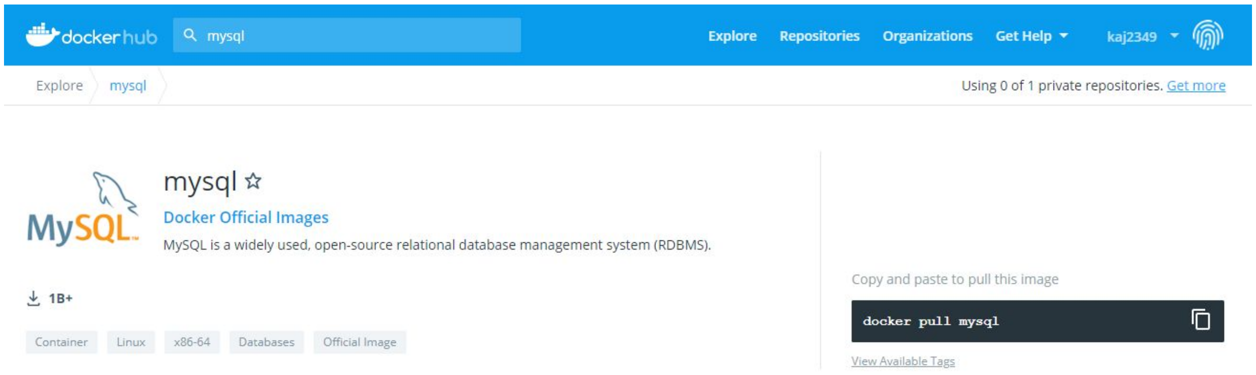
```
vp-db-adminer | [Sat Aug 22 22:58:05 2020] [::ffff:172.18.0.1]:50634 Accepted
vp-db-adminer | [Sat Aug 22 22:58:05 2020] [::ffff:172.18.0.1]:50634 [200]: GET /?
file=jush.js&version=4.7.7
vp-db-adminer | [Sat Aug 22 22:58:05 2020] [::ffff:172.18.0.1]:50634 Closing
vp-db-adminer | [Sat Aug 22 22:58:06 2020] [::ffff:172.18.0.1]:50642 Accepted
vp-db-adminer | [Sat Aug 22 22:58:06 2020] [::ffff:172.18.0.1]:50644 Accepted
vp-db-adminer | [Sat Aug 22 22:58:06 2020] [::ffff:172.18.0.1]:50644 [200]: GET /?
server=db&username=root&db=vpdb
vp-db-adminer | [Sat Aug 22 22:58:06 2020] [::ffff:172.18.0.1]:50644 Closing
vp-db-adminer | [Sat Aug 22 22:58:06 2020] [::ffff:172.18.0.1]:50642 [200]: GET /?
server=db&username=root&db=vpdb&script=db
vp-db-adminer | [Sat Aug 22 22:58:06 2020] [::ffff:172.18.0.1]:50642 Closing
vp-db-adminer | [Sat Aug 22 22:58:08 2020] [::ffff:172.18.0.1]:50656 Accepted
vp-db-adminer | [Sat Aug 22 22:58:08 2020] [::ffff:172.18.0.1]:50654 Accepted
vp-db-adminer | [Sat Aug 22 22:58:08 2020] [::ffff:172.18.0.1]:50656 [200]: GET /?
server=db&username=root&db=vpdb&table=users
vp-db-adminer | [Sat Aug 22 22:58:08 2020] [::ffff:172.18.0.1]:50656 Closing
vp-db-adminer | [Sat Aug 22 22:58:09 2020] [::ffff:172.18.0.1]:50662 Accepted
vp-db-adminer | [Sat Aug 22 22:58:09 2020] [::ffff:172.18.0.1]:50654 [200]: GET /?
server=db&username=root&db=vpdb&select=users
vp-db-adminer | [Sat Aug 22 22:58:09 2020] [::ffff:172.18.0.1]:50654 Closing
vp-db-adminer | [Sat Aug 22 22:59:48 2020] [::ffff:172.18.0.1]:50662 Closed without
sending a request; it was probably just an unused speculative preconnection
vp-db-adminer | [Sat Aug 22 22:59:48 2020] [::ffff:172.18.0.1]:50662 Closing
```

II. Use latest version of MySQL database 8.0.20

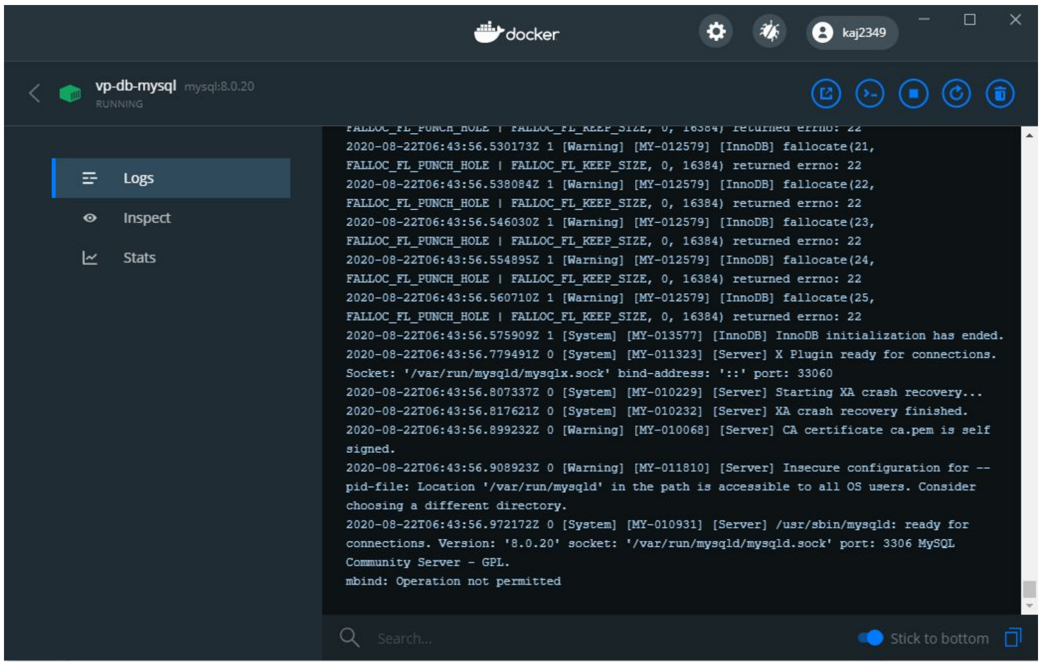
MySQL is an open-source relational database management system. Its name is a combination of "My", the name of co-founder Michael Widenius's daughter, and "SQL", the abbreviation for Structured Query Language.

https://www.mysql.com/

Latest image of mysql from docker hub was pulled.

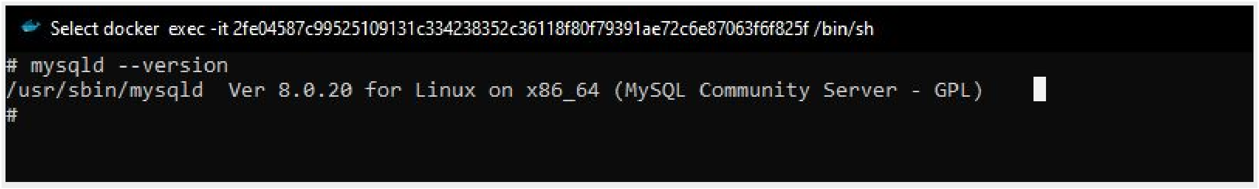


```
docker pull mysql
```



```
FALLOC_FL_PUNCH_HOLE | FALLOC_FL_KEEP_SIZE, 0, 16384) returned errno: 22
2020-08-22T06:43:56.530173Z 1 [Warning] [MY-012579] [InnoDB] fallocate(21,
FALLOC_FL_PUNCH_HOLE | FALLOC_FL_KEEP_SIZE, 0, 16384) returned errno: 22
2020-08-22T06:43:56.538084Z 1 [Warning] [MY-012579] [InnoDB] fallocate(22,
FALLOC_FL_PUNCH_HOLE | FALLOC_FL_KEEP_SIZE, 0, 16384) returned errno: 22
2020-08-22T06:43:56.546030Z 1 [Warning] [MY-012579] [InnoDB] fallocate(23,
FALLOC_FL_PUNCH_HOLE | FALLOC_FL_KEEP_SIZE, 0, 16384) returned errno: 22
2020-08-22T06:43:56.554895Z 1 [Warning] [MY-012579] [InnoDB] fallocate(24,
FALLOC_FL_PUNCH_HOLE | FALLOC_FL_KEEP_SIZE, 0, 16384) returned errno: 22
2020-08-22T06:43:56.560710Z 1 [Warning] [MY-012579] [InnoDB] fallocate(25,
FALLOC_FL_PUNCH_HOLE | FALLOC_FL_KEEP_SIZE, 0, 16384) returned errno: 22
2020-08-22T06:43:56.575909Z 1 [System] [MY-013577] [InnoDB] InnoDB initialization has ended.
2020-08-22T06:43:56.779491Z 0 [System] [MY-011323] [Server] X Plugin ready for connections.
Socket: '/var/run/mysqld/mysqlx.sock' bind-address: '::' port: 33060
2020-08-22T06:43:56.807337Z 0 [System] [MY-010229] [Server] Starting XA crash recovery...
2020-08-22T06:43:56.817621Z 0 [System] [MY-010232] [Server] XA crash recovery finished.
2020-08-22T06:43:56.899232Z 0 [Warning] [MY-010068] [Server] CA certificate ca.pem is self
signed.
2020-08-22T06:43:56.908923Z 0 [Warning] [MY-011810] [Server] Insecure configuration for --
pid-file: Location '/var/run/mysqld' in the path is accessible to all OS users. Consider
choosing a different directory.
2020-08-22T06:43:56.972172Z 0 [System] [MY-010931] [Server] /usr/sbin/mysqld: ready for
connections. Version: '8.0.20' socket: '/var/run/mysqld/mysqld.sock' port: 3306 MySQL
Community Server - GPL.
mbind: Operation not permitted
```

Version:



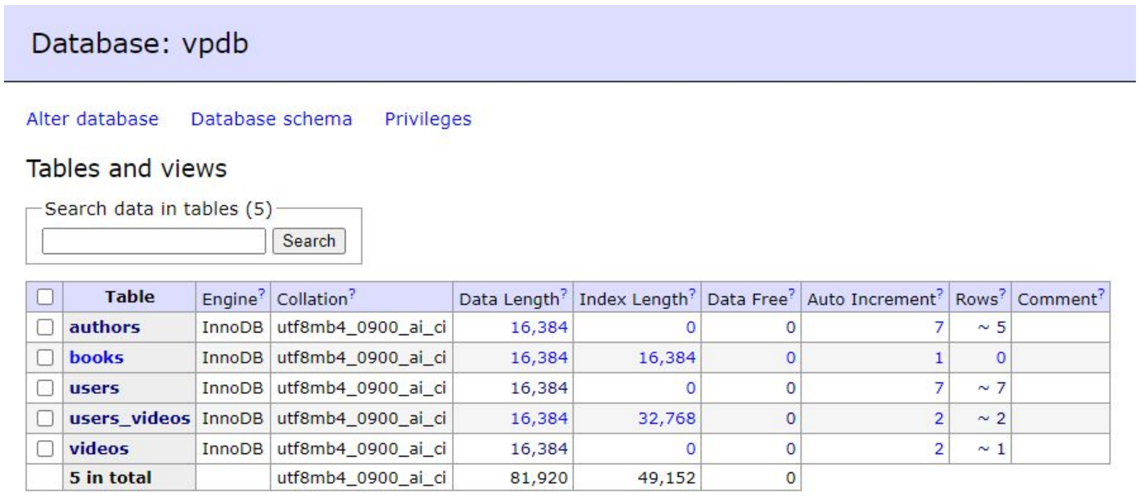
```
Select docker exec -it 2fe04587c99525109131c334238352c36118f80f79391ae72c6e87063f6f825f /bin/sh
# mysqld --version
/usr/sbin/mysqld  Ver 8.0.20 for Linux on x86_64 (MySQL Community Server - GPL)
#
```

III. Use Adminer, a web based management for MySQL server

Adminer is a tool for managing content in MySQL databases. Adminer is distributed under Apache license in a form of a single PHP file. Its author is Jakub Vrána who started to develop this tool as a light-weight alternative to phpMyAdmin, in July 2007.

https://www.adminer.org/

[Continued on the next page]



Database: vpdb

Alter database   Database schema   Privileges

Tables and views

Search data in tables (5)

Search

Table	Engine?	Collation?	Data Length?	Index Length?	Data Free?	Auto Increment?	Rows?	Comment?
authors	InnoDB	utf8mb4_0900_ai_ci	16,384	0	0	7	~ 5	
books	InnoDB	utf8mb4_0900_ai_ci	16,384	16,384	0	1	0	
users	InnoDB	utf8mb4_0900_ai_ci	16,384	0	0	7	~ 7	
users_videos	InnoDB	utf8mb4_0900_ai_ci	16,384	32,768	0	2	~ 2	
videos	InnoDB	utf8mb4_0900_ai_ci	16,384	0	0	2	~ 1	
5 in total		utf8mb4_0900_ai_ci	81,920	49,152	0			

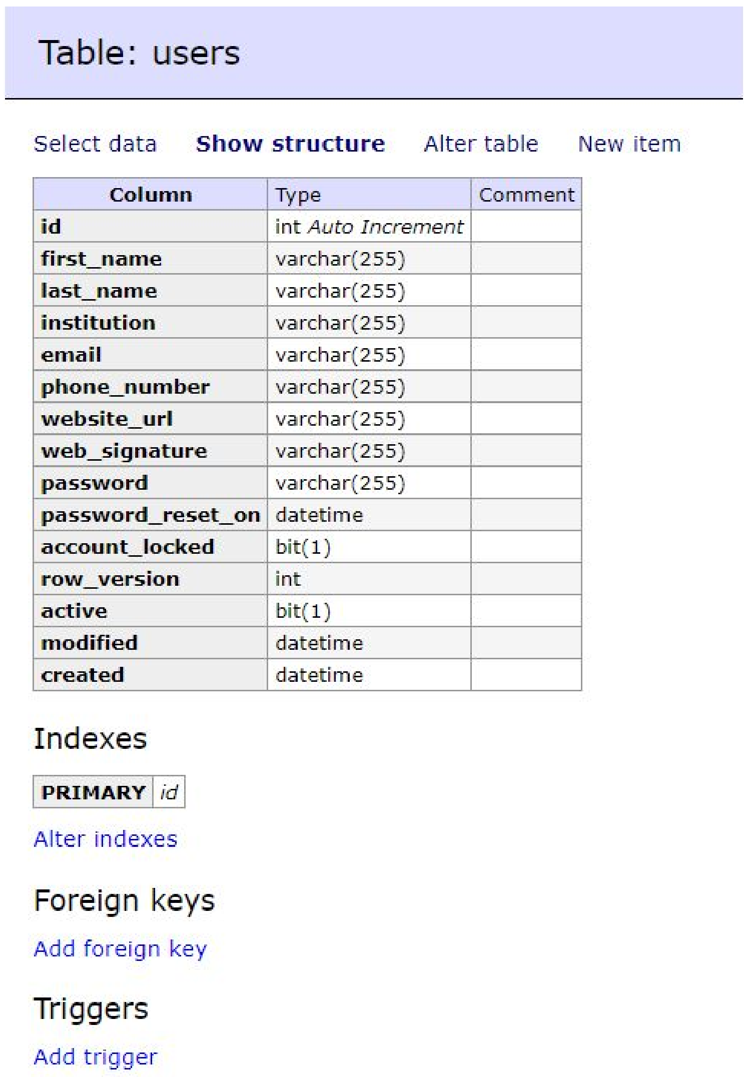


Table: users

Select data   **Show structure**   Alter table   New item

Column	Type	Comment
id	int *Auto Increment*	
first_name	varchar(255)	
last_name	varchar(255)	
institution	varchar(255)	
email	varchar(255)	
phone_number	varchar(255)	
website_url	varchar(255)	
web_signature	varchar(255)	
password	varchar(255)	
password_reset_on	datetime	
account_locked	bit(1)	
row_version	int	
active	bit(1)	
modified	datetime	
created	datetime	

Indexes

PRIMARY	*id*

Alter indexes

Foreign keys

Add foreign key

Triggers

Add trigger

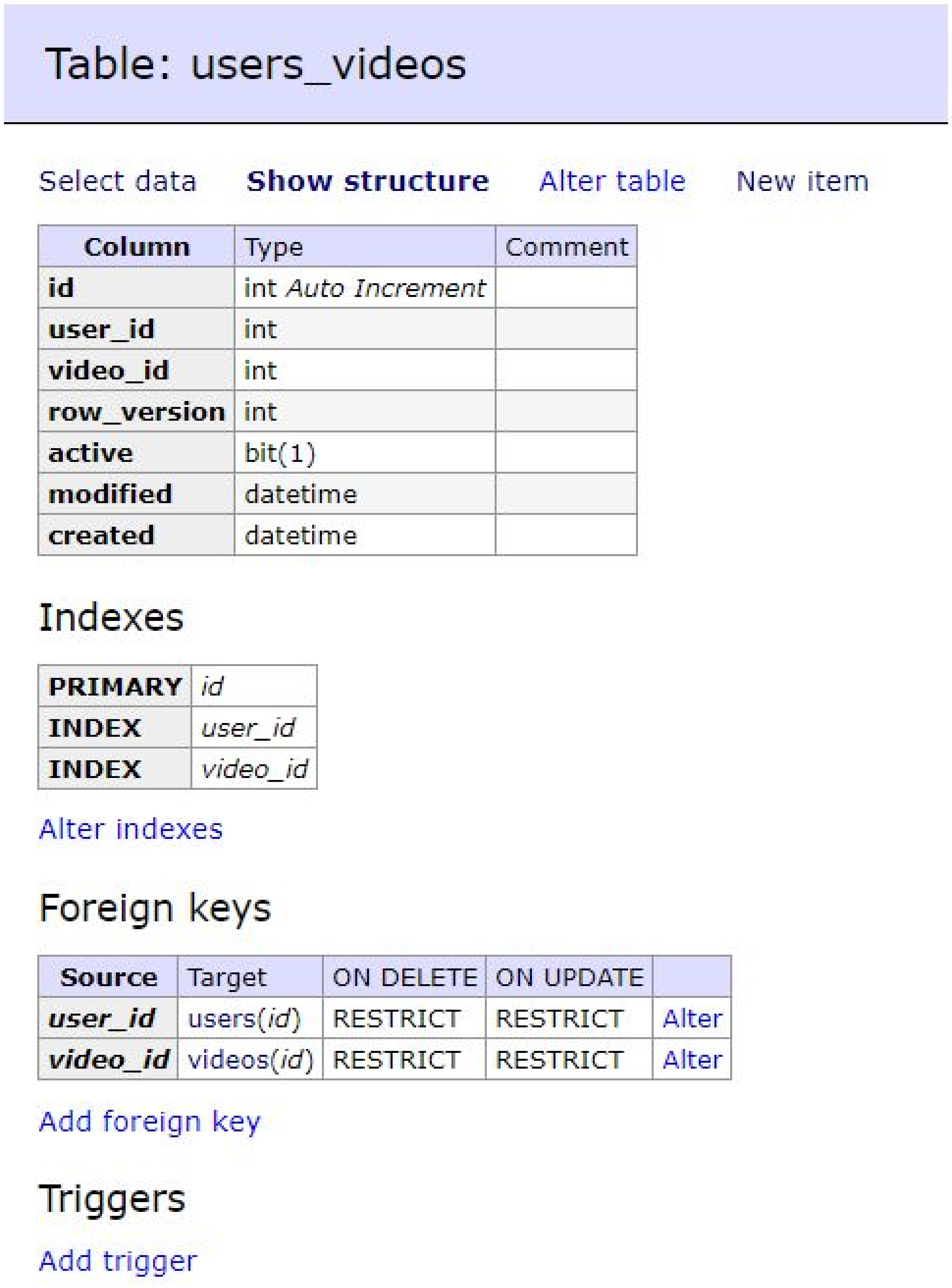


Table: users_videos

Select data **Show structure** Alter table New item

Column	Type	Comment
id	int *Auto Increment*	
user_id	int	
video_id	int	
row_version	int	
active	bit(1)	
modified	datetime	
created	datetime	

Indexes

PRIMARY	*id*
INDEX	*user_id*
INDEX	*video_id*

Alter indexes

Foreign keys

Source	Target	ON DELETE	ON UPDATE	
user_id	users(*id*)	RESTRICT	RESTRICT	Alter
video_id	videos(*id*)	RESTRICT	RESTRICT	Alter

Add foreign key

Triggers

Add trigger

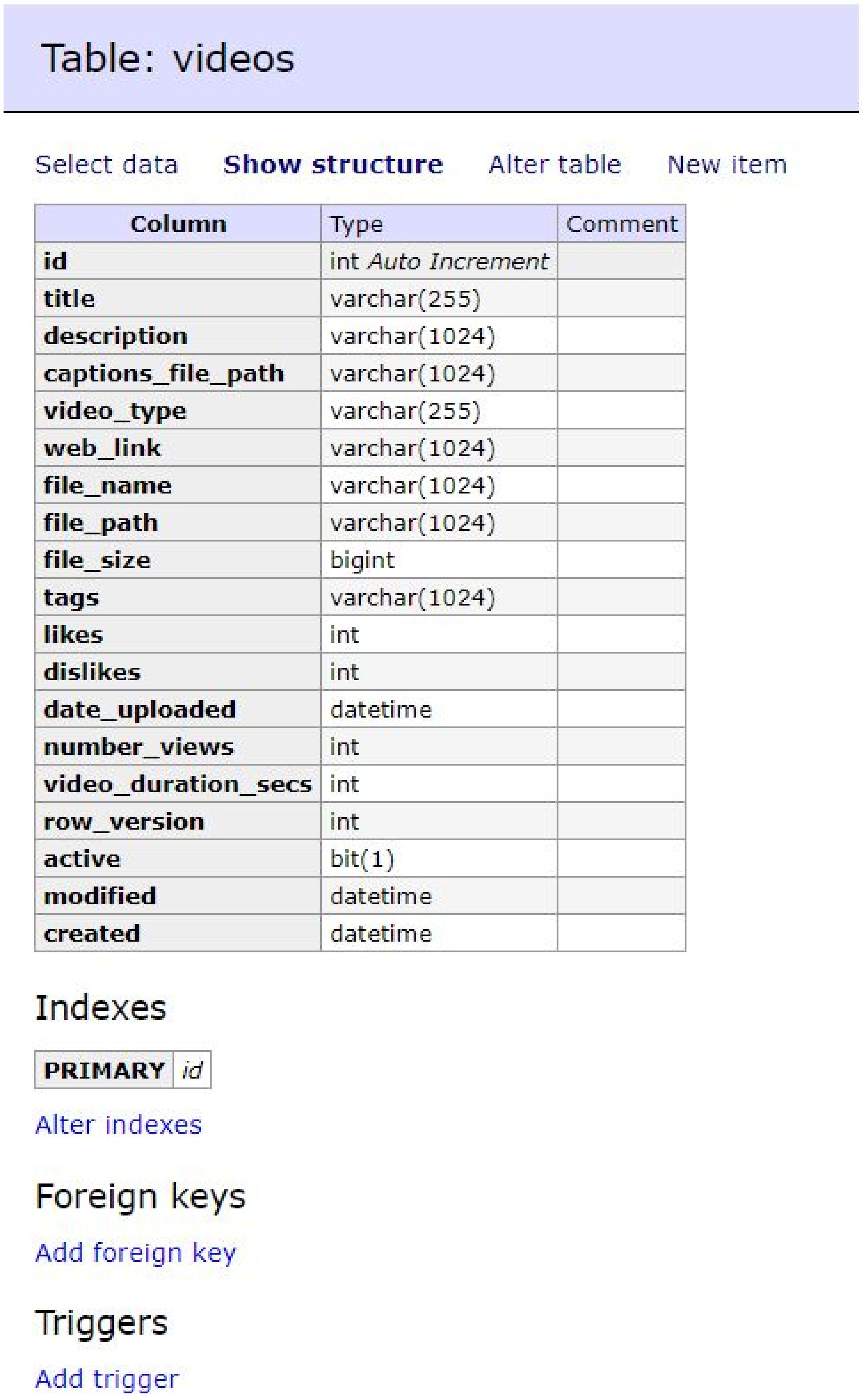


Table: videos

Select data **Show structure** Alter table New item

Column	Type	Comment
id	int *Auto Increment*	
title	varchar(255)	
description	varchar(1024)	
captions_file_path	varchar(1024)	
video_type	varchar(255)	
web_link	varchar(1024)	
file_name	varchar(1024)	
file_path	varchar(1024)	
file_size	bigint	
tags	varchar(1024)	
likes	int	
dislikes	int	
date_uploaded	datetime	
number_views	int	
video_duration_secs	int	
row_version	int	
active	bit(1)	
modified	datetime	
created	datetime	

Indexes

PRIMARY	*id*

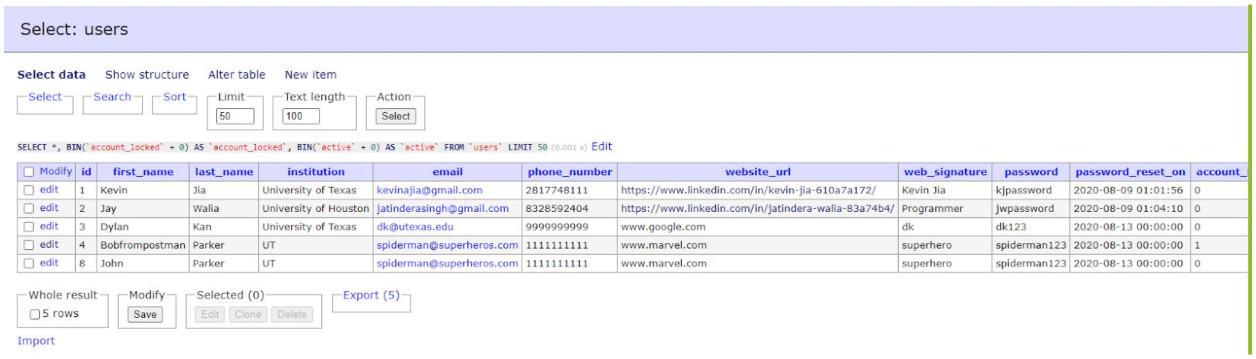
Alter indexes

Foreign keys

Add foreign key

Triggers

Add trigger



Select: users

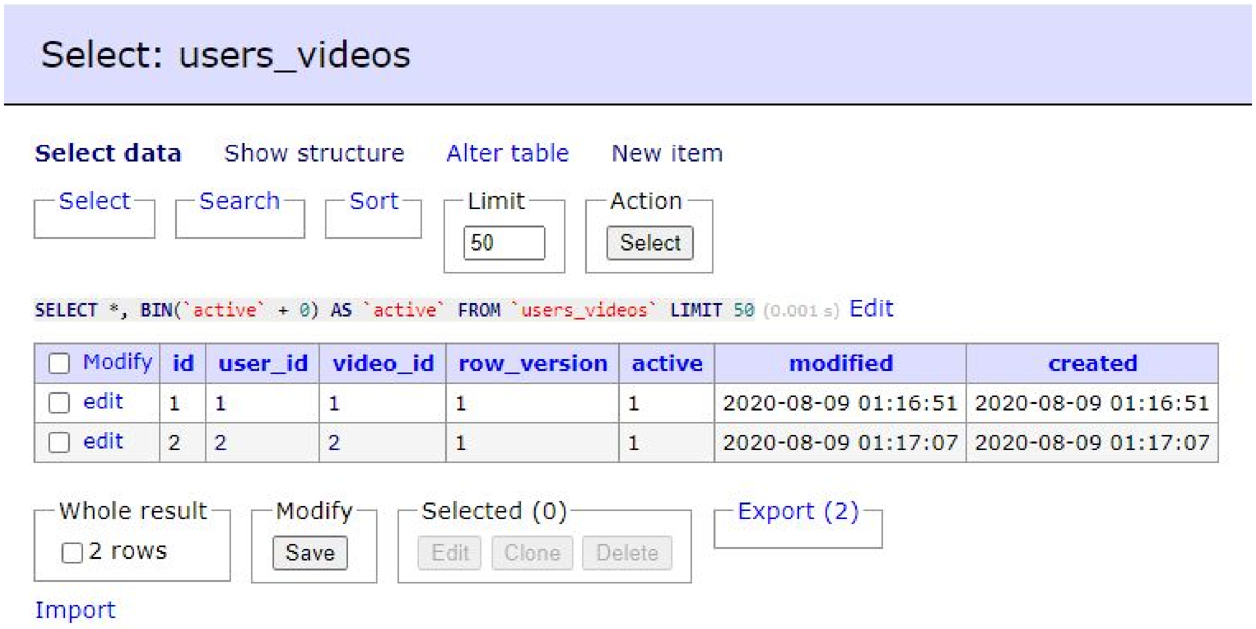
Select data Show structure Alter table New item

```
SELECT *, BIN(`account_locked` + 0) AS `account_locked`, BIN(`active` + 0) AS `active` FROM `users` LIMIT 50 (0.001 s) Edit
```

Modify	id	first_name	last_name	institution	email	phone_number	website_url	web_signature	password	password_reset_on	account_
edit	1	Kevin	Jia	University of Texas	kevinajia@gmail.com	2817748111	https://www.linkedin.com/in/kevin-jia-610a7a172/	Kevin Jia	kjpassword	2020-08-09 01:01:56	0
edit	2	Jay	Walia	University of Houston	jatinderasingh@gmail.com	8328592404	https://www.linkedin.com/in/jatindera-walia-83a74b4/	Programmer	jwpassword	2020-08-09 01:04:10	0
edit	3	Dylan	Kan	University of Texas	dk@utexas.edu	9999999999	www.google.com	dk	dk123	2020-08-13 00:00:00	0
edit	4	Bobfrompostman	Parker	UT	spiderman@superheros.com	1111111111	www.marvel.com	superhero	spiderman123	2020-08-13 00:00:00	1
edit	8	John	Parker	UT	spiderman@superheros.com	1111111111	www.marvel.com	superhero	spiderman123	2020-08-13 00:00:00	0

Whole result: 5 rows — Modify: Save — Selected (0): Edit Clone Delete — Export (5)

Import



Select: users_videos

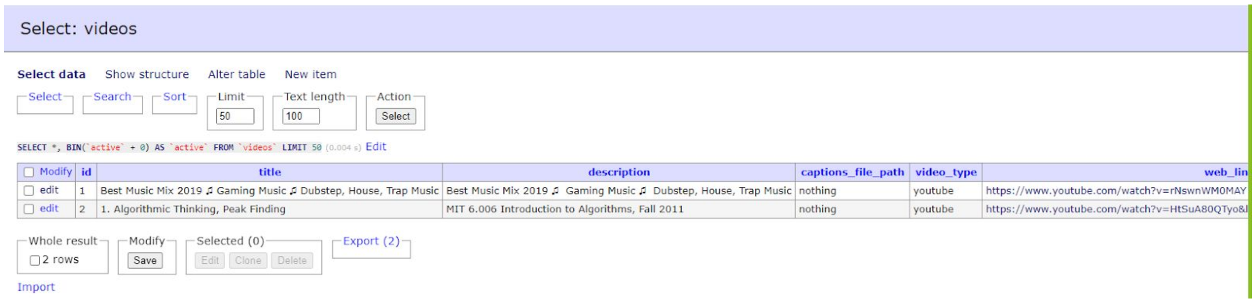
Select data Show structure Alter table New item

```
SELECT *, BIN(`active` + 0) AS `active` FROM `users_videos` LIMIT 50 (0.001 s) Edit
```

Modify	id	user_id	video_id	row_version	active	modified	created
edit	1	1	1	1	1	2020-08-09 01:16:51	2020-08-09 01:16:51
edit	2	2	2	1	1	2020-08-09 01:17:07	2020-08-09 01:17:07

Whole result: 2 rows — Modify: Save — Selected (0): Edit Clone Delete — Export (2)

Import



Select: videos

Select data Show structure Alter table New item

```
SELECT *, BIN(`active` + 0) AS `active` FROM `videos` LIMIT 50 (0.004 s) Edit
```

Modify	id	title	description	captions_file_path	video_type	web_lin
edit	1	Best Music Mix 2019 ♫ Gaming Music ♫ Dubstep, House, Trap Music	Best Music Mix 2019 ♫ Gaming Music ♫ Dubstep, House, Trap Music	nothing	youtube	https://www.youtube.com/watch?v=rNswnWM0MAY
edit	2	1. Algorithmic Thinking, Peak Finding	MIT 6.006 Introduction to Algorithms, Fall 2011	nothing	youtube	https://www.youtube.com/watch?v=HtSuA80QTyo&l

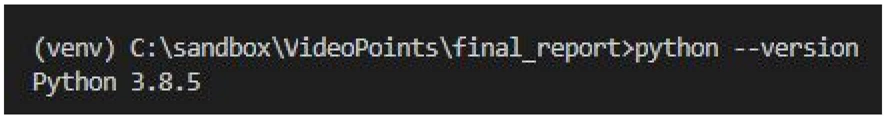
Whole result: 2 rows — Modify: Save — Selected (0): Edit Clone Delete — Export (2)

Import

IV. Use Python as a programming language

Python is an interpreted, high level, dynamically typed, object-oriented programming language that emphasizes code readability. Python was designed by Guido van Rossum and released in 1991. It is currently developed by the Python Software Foundation. Because python is a general-purpose coding language, it can be used for many types of programming besides web development, such as backend development and data science.

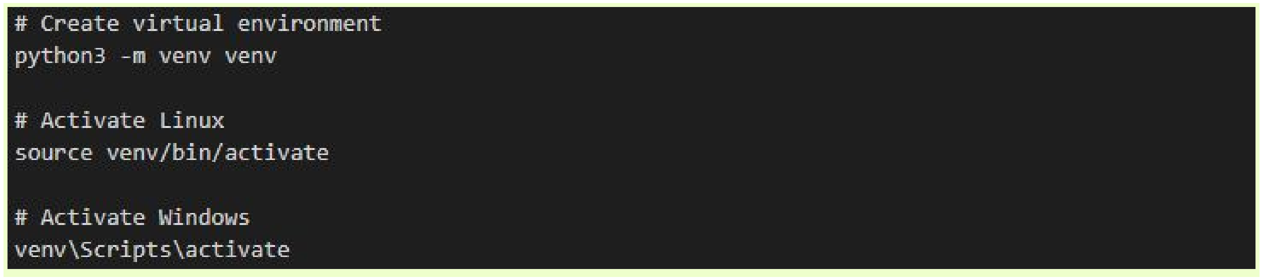
Python version 3.8.5 was used.



```
(venv) C:\sandbox\VideoPoints\final_report>python --version
Python 3.8.5
```

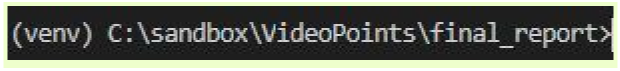
Virtual environment

Python applications will often use packages and modules that don't come as part of the standard library. Applications will sometimes need a specific version of a library, because the application may require that a particular bug has been fixed or the application may be written using an obsolete version of the library's interface. The solution for this problem is to create a virtual environment, a self-contained directory tree that contains a Python installation for a particular version of Python, plus a number of additional packages.



```
# Create virtual environment
python3 -m venv venv

# Activate Linux
source venv/bin/activate

# Activate Windows
venv\Scripts\activate
```

When the environment is activated, the command prompt displays as (venv).



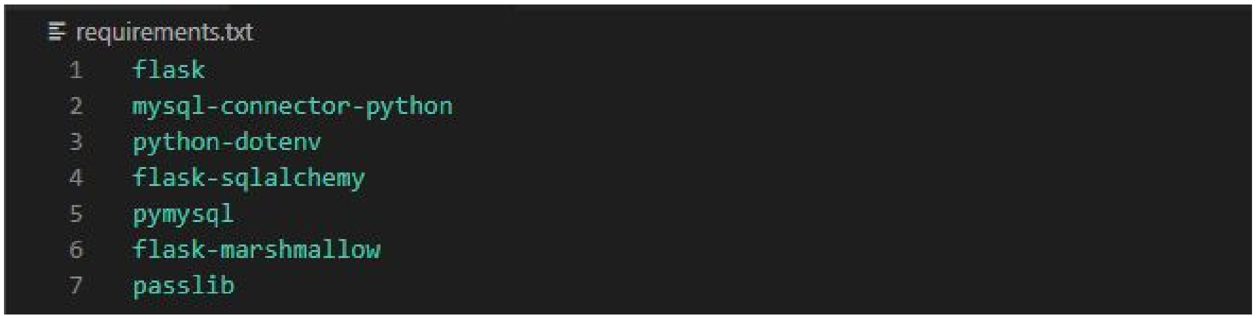
```
(venv) C:\sandbox\VideoPoints\final_report>
```

Pip

Pip is a de facto standard package-management system used to install and manage software packages written in Python. Many packages can be found in the default source for packages and their dependencies — Python Package Index.

```
# Upgrade pip installer
pip install --upgrade pip

# install Flask command
pip install flask

# install mySQL connector
pip install mysql-connector-python
```

Requirements.txt
All packages that need to be installed can be kept in requirements.txt



```
flask
mysql-connector-python
python-dotenv
flask-sqlalchemy
pymysql
flask-marshmallow
passlib
```

To install requirements, run the command

```
# Install requirements.txt
pip install -r requirements.txt
```

To get a snapshot of virtual environment, run the command

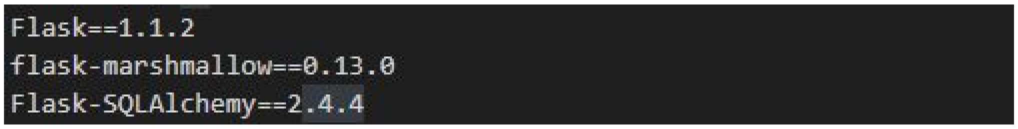
```
# Install requirements.txt
pip freeze > requirements.txt
pip install -r requirements.txt
```

V. Use Flask as the web framework

Flask is a micro web framework that is written in python. It is a microframework, and does not require particular tools or libraries. In addition Flask has no database abstraction layer, form validation, or any other components where pre-existing third party libraries provide common functions. Flask does however, support extensions that add application features as if they were implemented in Flask itself. Flask was created by Armin Ronacher of Pocoo, an international group of python enthusiasts that was formed in 2004. Flask provides the tools, libraries, and technologies needed to build a web application.

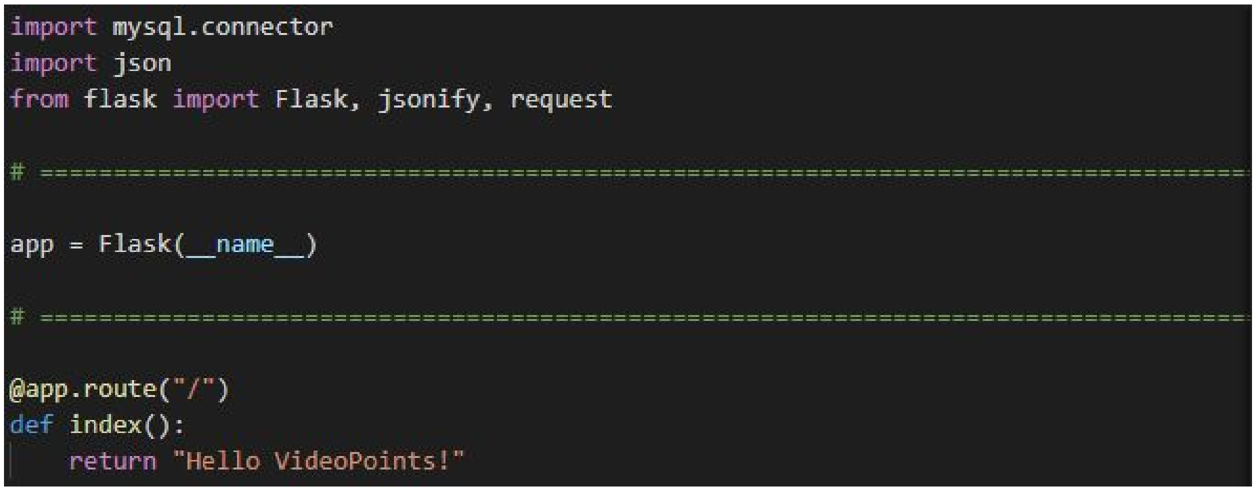
Flask version

Flask version 1.1.2



```
Flask==1.1.2
flask-marshmallow==0.13.0
Flask-SQLAlchemy==2.4.4
```

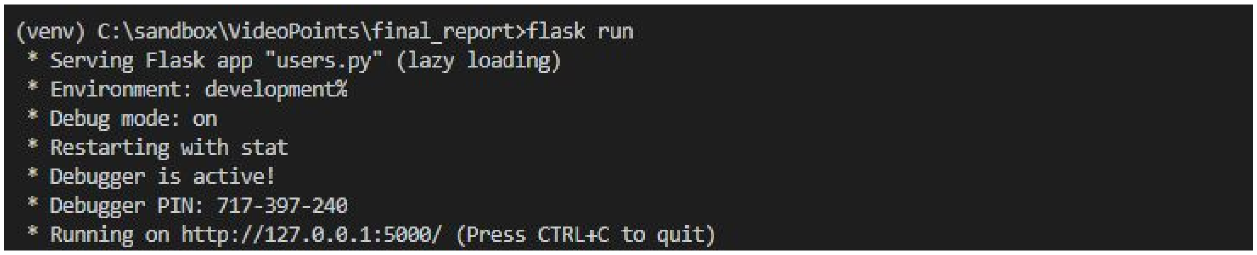
Hello VideoPoints!



```
import mysql.connector
import json
from flask import Flask, jsonify, request

# ================================================================================

app = Flask(__name__)

# ================================================================================

@app.route("/")
def index():
    return "Hello VideoPoints!"
```

Flask run

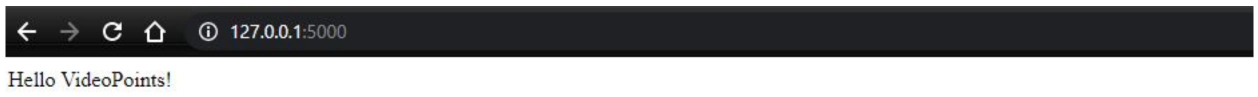
To execute, run



```
(venv) C:\sandbox\VideoPoints\final_report>flask run
 * Serving Flask app "users.py" (lazy loading)
 * Environment: development%
 * Debug mode: on
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 717-397-240
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

Port 5000
When Flask is run by default, it runs on port 5000.
http://127.0.0.1:5000/

Using browser to connect



```
Hello VideoPoints!
```

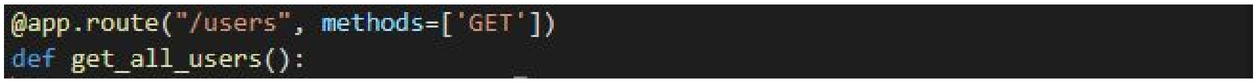
VI. Use RESTful API

The REST (Representational State Transfer) API design was followed. REST is a software architectural style that defines a set of constraints to be used for created web services. Such RESTful Web services provide interoperability between computer systems on the internet. These services allow requesting systems to access and manipulate textual representations of web resources by using a uniform and predefined set of stateless operations. A REST or RESTful API design is designed to take advantage of existing protocols. A REST API is an application program interface that uses HTTP requests to GET, HEAD, POST, PUT, PATCH, DELETE, CONNECT, OPTIONS and TRACE. Because RESTful systems use a stateless protocol and standard operations, they typically aim for fast performance, reliability and ability to grow by reusing components that can be managed and updated without affecting the whole system. A RESTful system is defined by six guiding constraints. These constraints restrict the ways in which the server can process and respond to client requests. The formal REST constraints are:

- Client-server
- Stateless
- Cacheable
- Uniform interface
- Layered system
- Code on demand (optional)

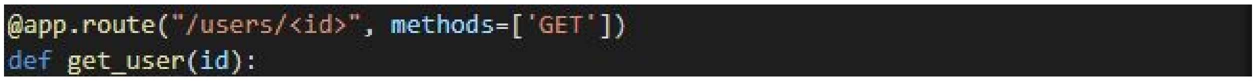
Through operating within these constraints, the system gains non-functional properties such as performance, scalability, simplicity, modifiability, visibility, portability, and reliability. If a system violates any of the required constraints, it cannot be considered RESTful.

API: Get all users
This API will return all registered users. More advanced API can provide paging functionality.



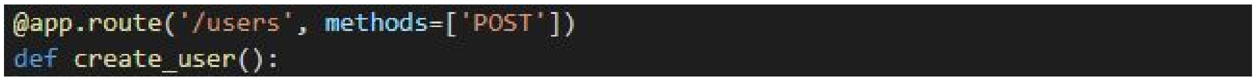
```
@app.route("/users", methods=['GET'])
def get_all_users():
```

API: Get user by ID
This API will return a user by ID.



```
@app.route("/users/<id>", methods=['GET'])
def get_user(id):
```

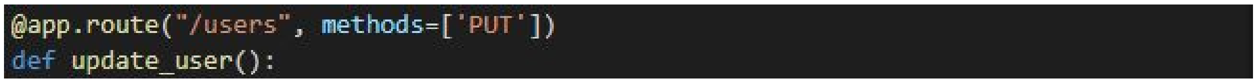
API: Create user
This API will create a new user from a json document.



```
@app.route('/users', methods=['POST'])
def create_user():
```

API: Update user

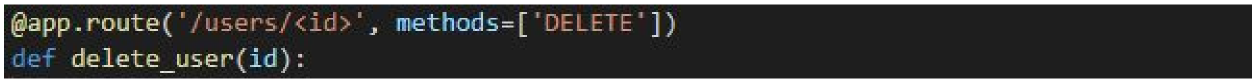
This API will update an existing user from a json document



```
@app.route("/users", methods=['PUT'])
def update_user():
```

API: Delete user

This API will delete an existing user by ID.



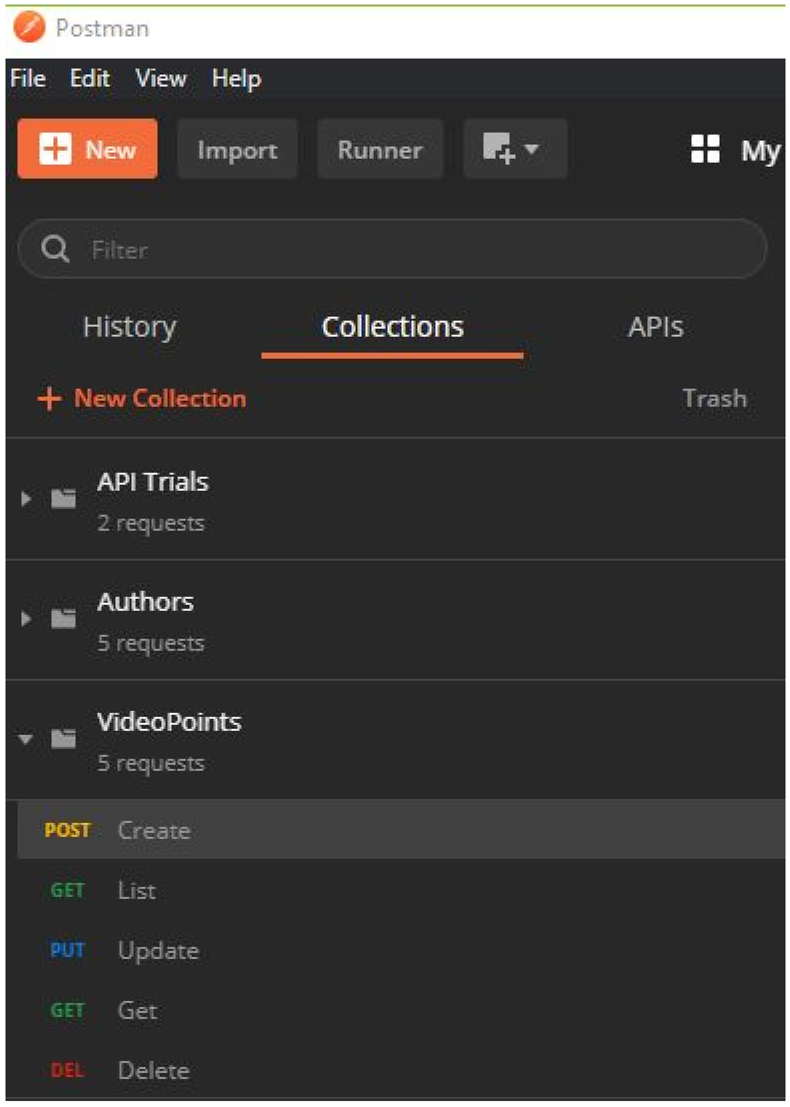
```
@app.route('/users/<id>', methods=['DELETE'])
def delete_user(id):
```

VII. Use Postman to test the API

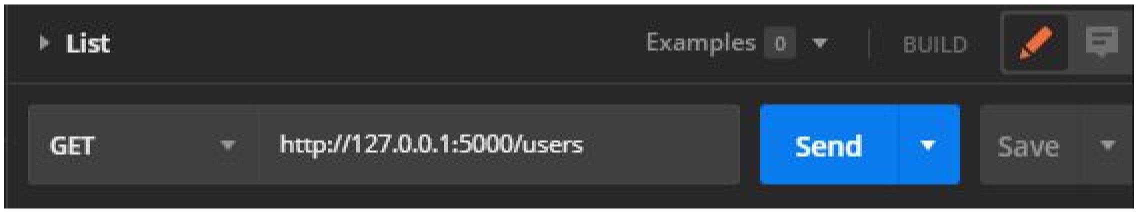
Postman is a collaboration platform for API development. Postman's features simplify each step of building an API and streamline collaboration.

https://www.postman.com/

After installing Postman, collections can be generated. A collection consists of API tests for a particular resource.

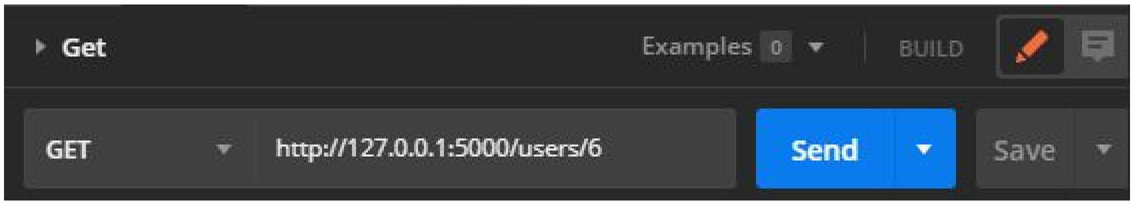


The List test will test the API for listing all users



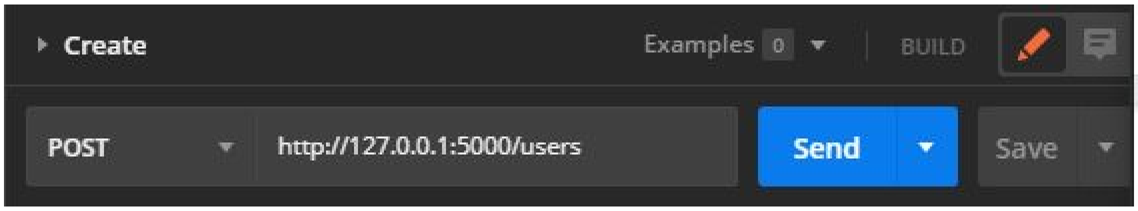
http://127.0.0.1:5000/users

The Get test will test the API for getting an user by ID.



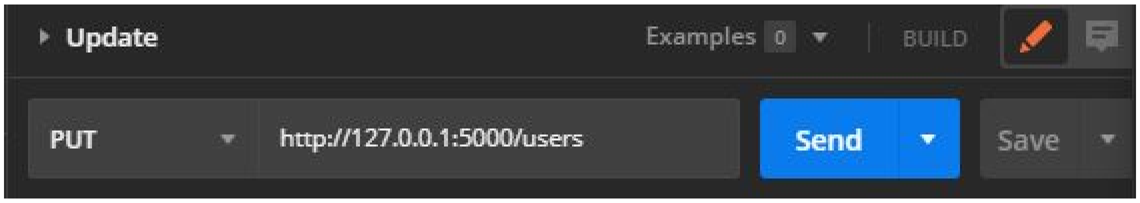
http://127.0.0.1:5000/users/6

The Create test will test the API for creating an user from a json document.



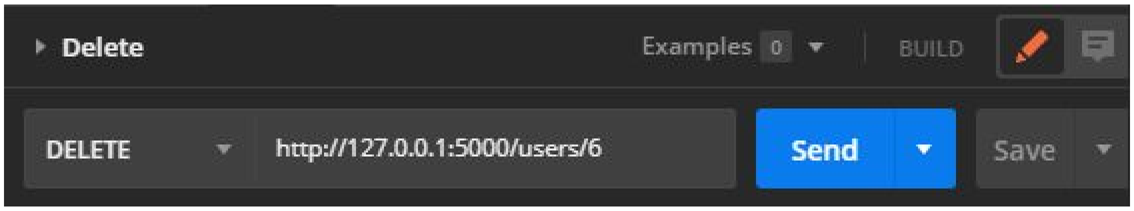
http://127.0.0.1:5000/users

The Update test will test the API for updating an existing user from a json document.



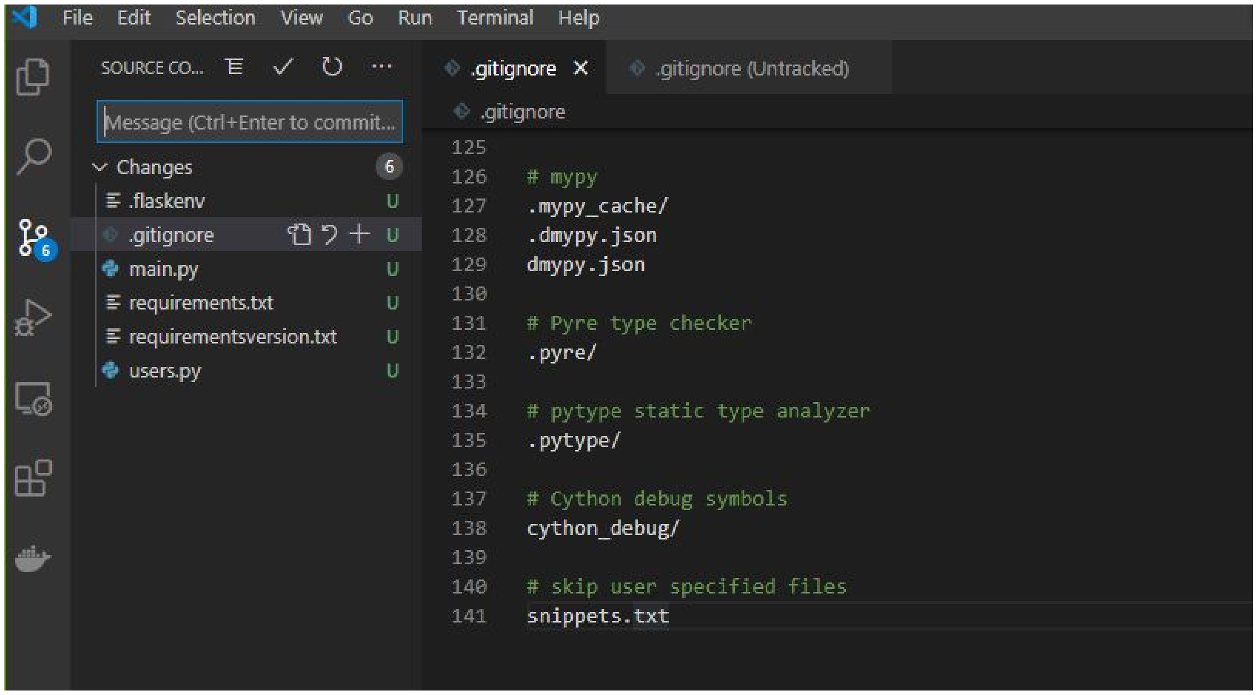
http://127.0.0.1:5000/users

The Delete test will test the API for deleting an existing user by ID.



http://127.0.0.1:5000/users/6

VIII. Use Git for version control



```
# mypy
.mypy_cache/
.dmypy.json
dmypy.json

# Pyre type checker
.pyre/

# pytype static type analyzer
.pytype/

# Cython debug symbols
cython_debug/

# skip user specified files
snippets.txt
```

Git is a distributed version-control system for tracking changes in source code during software development. It is designed for coordinating work among programmers, but it can be used to track changes in any set of files. Its goals include speed, data integrity, and support for distributed, non-linear workflows.

Git can be downloaded from https://git-scm.com/

Visual studio code has a plugin for Git
https://code.visualstudio.com/docs/editor/versioncontrol

.gitignore

A gitignore file specifies intentionally untracked files that Git should ignore.

Python gitignore

There's a well documented python gitignore that can be used.

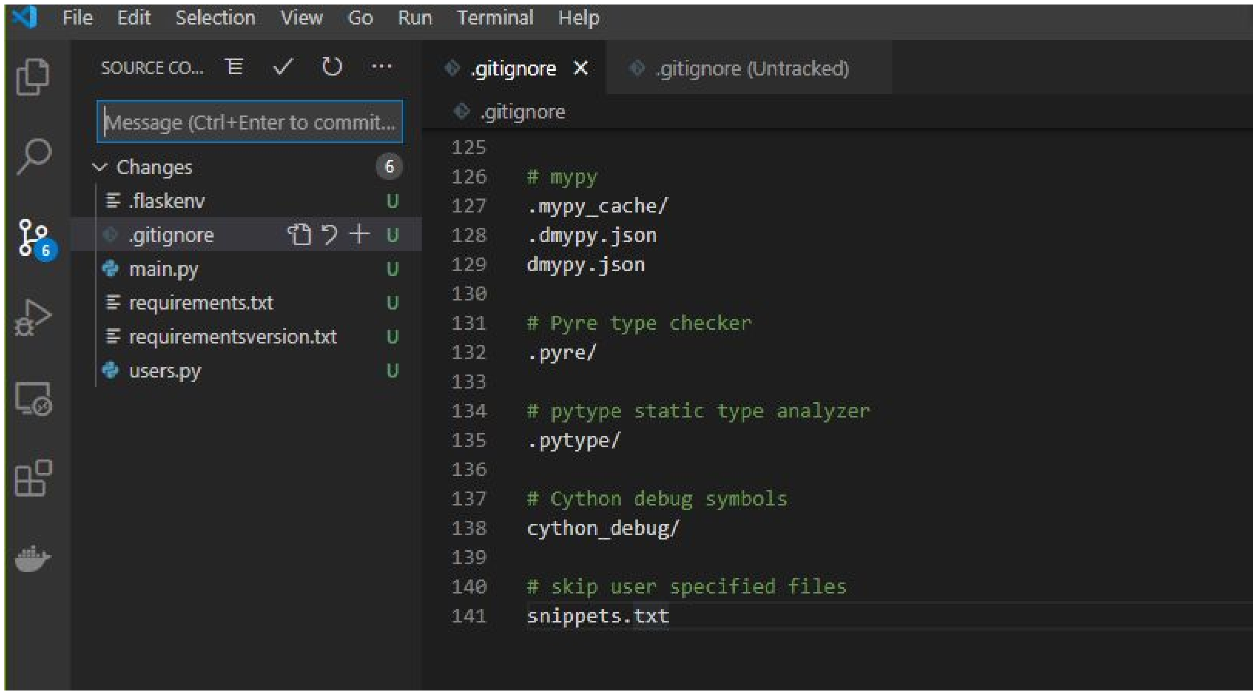
https://github.com/github/gitignore/blob/master/Python.gitignore

Github

GitHub, Inc. is an American multinational corporation that provides hosting for software development and version control using Git. It offers the distributed version control and source code management functionality of Git, plus its own features.

The project source along with this report is available at
https://github.com/kevinajia/VPMicroservices

A local repository was created. Source code was checked into and pushed into the repository on github.


File Edit Selection View Go Run Terminal Help
SOURCE CO...
Message (Ctrl+Enter to commit...
Changes 6
.gitignore
.gitignore (Untracked)

```
.flaskenv                 U
.gitignore                U
main.py                   U
requirements.txt          U
requirementsversion.txt   U
users.py                  U
```

```

# mypy
.mypy_cache/
.dmypy.json
dmypy.json

# Pyre type checker
.pyre/

# pytype static type analyzer
.pytype/

# Cython debug symbols
cython_debug/

# skip user specified files
snippets.txt
```

Summary

Microservice architecture will allow VideoPoints to scale and achieve high growth rates. Using containers will allow deployment across multiple cloud providers, while maintaining versioning and flexibility. It will also allow VideoPoints to utilize the power of devops. Flask has a small footprint while providing the ability to achieve high performance. Postman is a useful tool for testing the APIs and proceeding with test first development methodologies.

Work to be done

The focus of this internship was to develop a proof of concept for moving to microservice based architecture, but there is still work to be done. Following tasks are high priority to develop a comprehensive solution for the end user:

- APIs for other resources.
- Develop a front end using React.

Conclusion

Exposure to this project gave me valuable insight to the following areas of computer science, including:

Containers

APIs

Testing

Flask web framework

SQLAlchemy

Database schema design

I would like to thank Dr. Subhlok for giving me the opportunity to work on the project. I would also like to extend my gratitude to the VideoPoints Team for helping me overcome the challenges I faced while working on this project.

References

https://restfulapi.net/

https://www.mulesoft.com/resources/api/what-is-rest-api-design#:~:text=REST%20or%20RESTful%20API%20design,when%20used%20for%20Web%20APIs.

https://www.w3schools.com/

https://stackoverflow.com/

https://blog.miguelgrinberg.com/post/designing-a-restful-api-with-python-and-flask

https://pynative.com/

https://www.tutorialspoint.com/

https://www.docker.com/

https://www.mysql.com/

https://www.adminer.org/

https://flask-restful.readthedocs.io/en/latest/

https://docs.python.org/3/tutorial/venv.html

https://github.com/github/gitignore/blob/master/Python.gitignore

https://www.postman.com/

https://en.wikipedia.org/wiki/Git

https://git-scm.com/docs/gitignore

https://en.wikipedia.org/wiki/GitHub

https://realpython.com/python-comments-guide/

Book: Building REST APIs with Flask - Kunal Relan

https://www.amazon.com/Building-REST-APIs-Flask-Services-ebook/dp/B07XWB8VLL

Appendix - A (Code) (python users.py)

```
import mysql.connector

import json

from flask import Flask, jsonify, request


#
==================================================================================
=====


app = Flask(__name__)


#
==================================================================================
=====


@app.route("/")

def index():

    return "Hello VideoPoints!"


#
==================================================================================
=====


@app.route('/users', methods=['POST'])

def create_user():

    """

    Create an user from a json doc
    """

    posted_data = request.get_json()


    print (posted_data)


    tuple_params = tuple(posted_data.values())


    print (tuple_params)


    # connect to database


    mydb = mysql.connector.connect(

    host="127.0.0.1",

    #host = "db";

    port = "53306",

    user="vpUser",

    password="*************",

    database="vpdb"

    )


    # set insert cursor

    mycursor = mydb.cursor()


    query = "INSERT INTO users \

            (first_name, last_name, institution, email, phone_number,
website_url, web_signature, \

            password, password_reset_on, account_locked, row_version, active,
modified, created) \
            select \
            %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s"

    # parameters = ('Peter', 'Parker', 'UT', 'spiderman@superheros.com',
'1111111111', 'www.marvel.com',
    # 'superhero', 'spiderman123', '2020-08-13', 0, 1, 1, '2020-08-13',
'2020-08-13')

    mycursor.execute(query, tuple_params)

    mydb.commit()

    users_created = mycursor.rowcount

    mycursor.close()
    mydb.close()

    print(users_created)
    return "users_created: {}".format(users_created)

#
=================================================================================
=====

@app.route("/users/<id>", methods=['GET'])
def get_user(id):
    """
    Get a user by ID
```

```
"""

print (id)

mydb = mysql.connector.connect(
host="127.0.0.1",
#host = "db";
port = "53306",
user="vpUser",
password="*************",
database="vpdb"
)

mycursor = mydb.cursor()

mycursor.execute("select * from users where id = %s", (id, ))

row_headers=[x[0] for x in mycursor.description]

rv = mycursor.fetchall()
print (rv)

json_data=[]
for result in rv:
    json_data.append(dict(zip(row_headers,result)))
    return json.dumps(json_data, indent=4, sort_keys=True, default=str)


#
================================================================================
=====


@app.route("/users", methods=['PUT'])
def update_user():
    """
    Update a user from a json doc

    """

    posted_data = request.get_json()

    # print (posted_data)

    tuple_params = tuple(posted_data.values())

    print (tuple_params)

    # connect to database

    mydb = mysql.connector.connect(
    host="127.0.0.1",
    #host = "db";
    port = "53306",
    user="vpUser",
    password="*************",

    database="vpdb"

    )


    # set insert cursor

    mycursor = mydb.cursor()


    query = "UPDATE users SET \

    first_name = %s, \

    last_name = %s, \

    institution = %s, \

    email = %s, \

    phone_number = %s, \

    website_url = %s, \

    web_signature = %s, \

    password = %s, \

    password_reset_on = %s, \

    account_locked = %s, \

    row_version = %s, \

    active = %s, \

    modified = now() \

    WHERE id = %s"


    print (query)


    mycursor.execute(query, tuple_params)
```

```
    mydb.commit()


    user_updated = mycursor.rowcount


    mycursor.close()

    mydb.close()


    print(user_updated)

    return "user_updated: {}".format(user_updated)


#
==================================================================================
=====


@app.route("/users", methods=['GET'])

def get_all_users():

    """

    Get all users


    """


    mydb = mysql.connector.connect(

    host="127.0.0.1",

    #host = "db";

    port = "53306",

    user="vpUser",

    password="*************",

    database="vpdb"
    )

    mycursor = mydb.cursor()

    mycursor.execute("SELECT * FROM users")

    row_headers=[x[0] for x in mycursor.description]

    rv = mycursor.fetchall()

    json_data=[]
    for result in rv:
        json_data.append(dict(zip(row_headers,result)))

    mycursor.close()
    mydb.close()

    return json.dumps(json_data, indent=4, sort_keys=False, default=str)

#
=================================================================================
=====

@app.route('/users/<id>', methods=['DELETE'])
def delete_user(id):
    """

    Delete an existing user
```

```
"""

print (id)

# connect to database

mydb = mysql.connector.connect(
host="127.0.0.1",
#host = "db";
port = "53306",
user="vpUser",
password="*************",
database="vpdb"
)

# set insert cursor
mycursor = mydb.cursor()

query = "DELETE FROM users WHERE id = %s"

params = (id, )

print (query)

mycursor.execute(query, params)

mydb.commit()
```

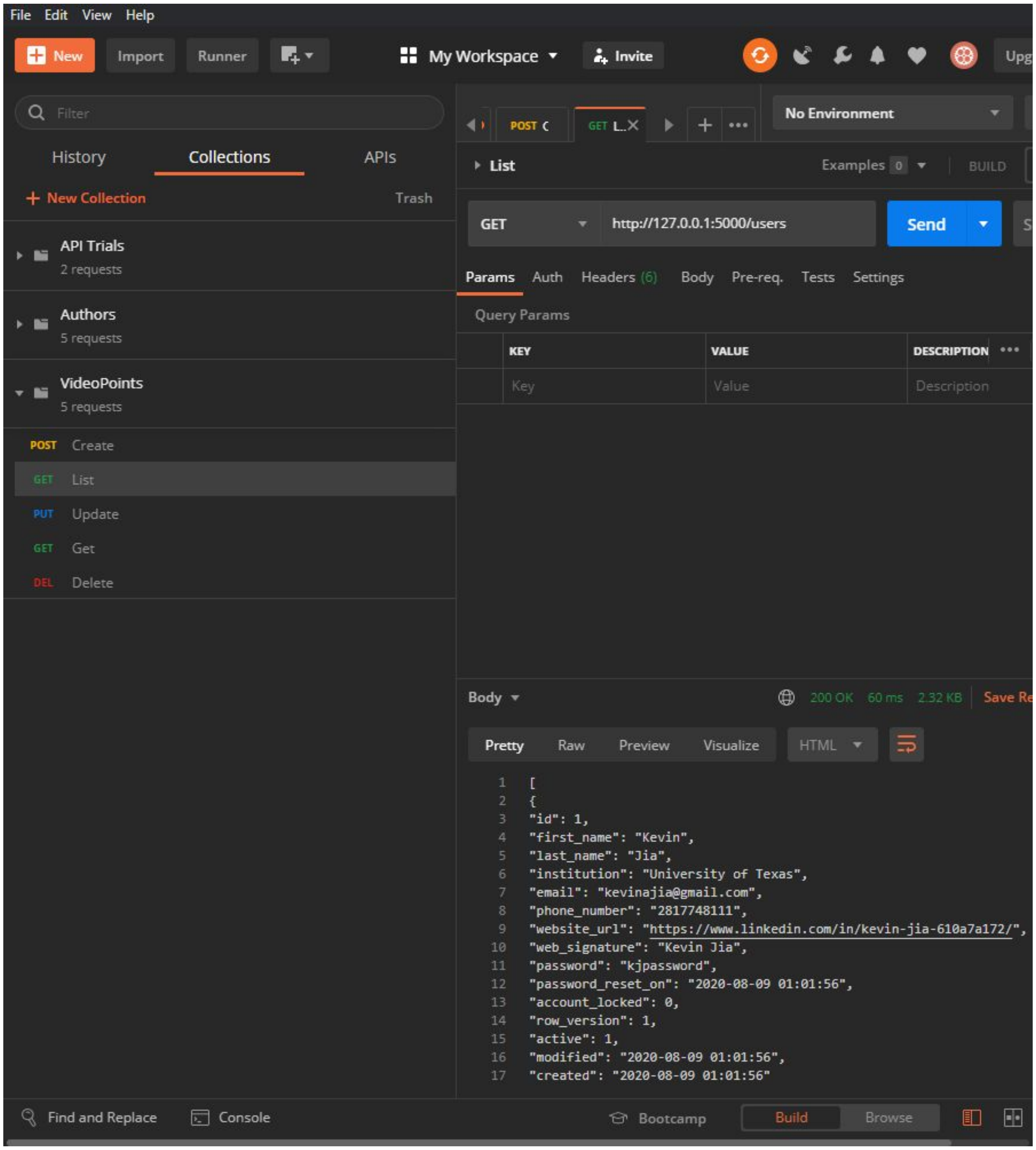
```
    message = ""
    user_deleted = mycursor.rowcount
    if user_deleted == 1:
        message = "user successfully deleted"
    else:
        message = "error in user deletion"

    mycursor.close()
    mydb.close()

    return message

# ======================================================================================

if __name__ == "__main__":
    app.run()
```

Appendix - B (Code) (database users, videos, user_videos)

```
CREATE TABLE `users` (
  `id` int NOT NULL AUTO_INCREMENT,
  `first_name` varchar(255) NOT NULL,
  `last_name` varchar(255) NOT NULL,
  `institution` varchar(255) CHARACTER SET utf8mb4 COLLATE utf8mb4_0900_ai_ci NOT NULL,
  `email` varchar(255) CHARACTER SET utf8mb4 COLLATE utf8mb4_0900_ai_ci NOT NULL,
  `phone_number` varchar(255) CHARACTER SET utf8mb4 COLLATE utf8mb4_0900_ai_ci NOT
NULL,
  `website_url` varchar(255) CHARACTER SET utf8mb4 COLLATE utf8mb4_0900_ai_ci NOT
NULL,
  `web_signature` varchar(255) CHARACTER SET utf8mb4 COLLATE utf8mb4_0900_ai_ci NOT
NULL,
  `password` varchar(255) CHARACTER SET utf8mb4 COLLATE utf8mb4_0900_ai_ci NOT NULL,
  `password_reset_on` datetime NOT NULL,
  `account_locked` bit(1) NOT NULL,
  `row_version` int NOT NULL,
  `active` bit(1) NOT NULL,
  `modified` datetime NOT NULL,
  `created` datetime NOT NULL ON UPDATE CURRENT_TIMESTAMP,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=8 DEFAULT CHARSET=utf8mb4
COLLATE=utf8mb4_0900_ai_ci


CREATE TABLE `users_videos` (
  `id` int NOT NULL AUTO_INCREMENT,
  `user_id` int NOT NULL,
  `video_id` int NOT NULL,
  `row_version` int NOT NULL,
  `active` bit(1) NOT NULL,
  `modified` datetime NOT NULL,
  `created` datetime NOT NULL,
  PRIMARY KEY (`id`),
  KEY `user_id` (`user_id`),
  KEY `video_id` (`video_id`),
  CONSTRAINT `users_videos_ibfk_1` FOREIGN KEY (`user_id`) REFERENCES `users` (`id`),
  CONSTRAINT `users_videos_ibfk_2` FOREIGN KEY (`video_id`) REFERENCES `videos` (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=3 DEFAULT CHARSET=utf8mb4
COLLATE=utf8mb4_0900_ai_ci
```

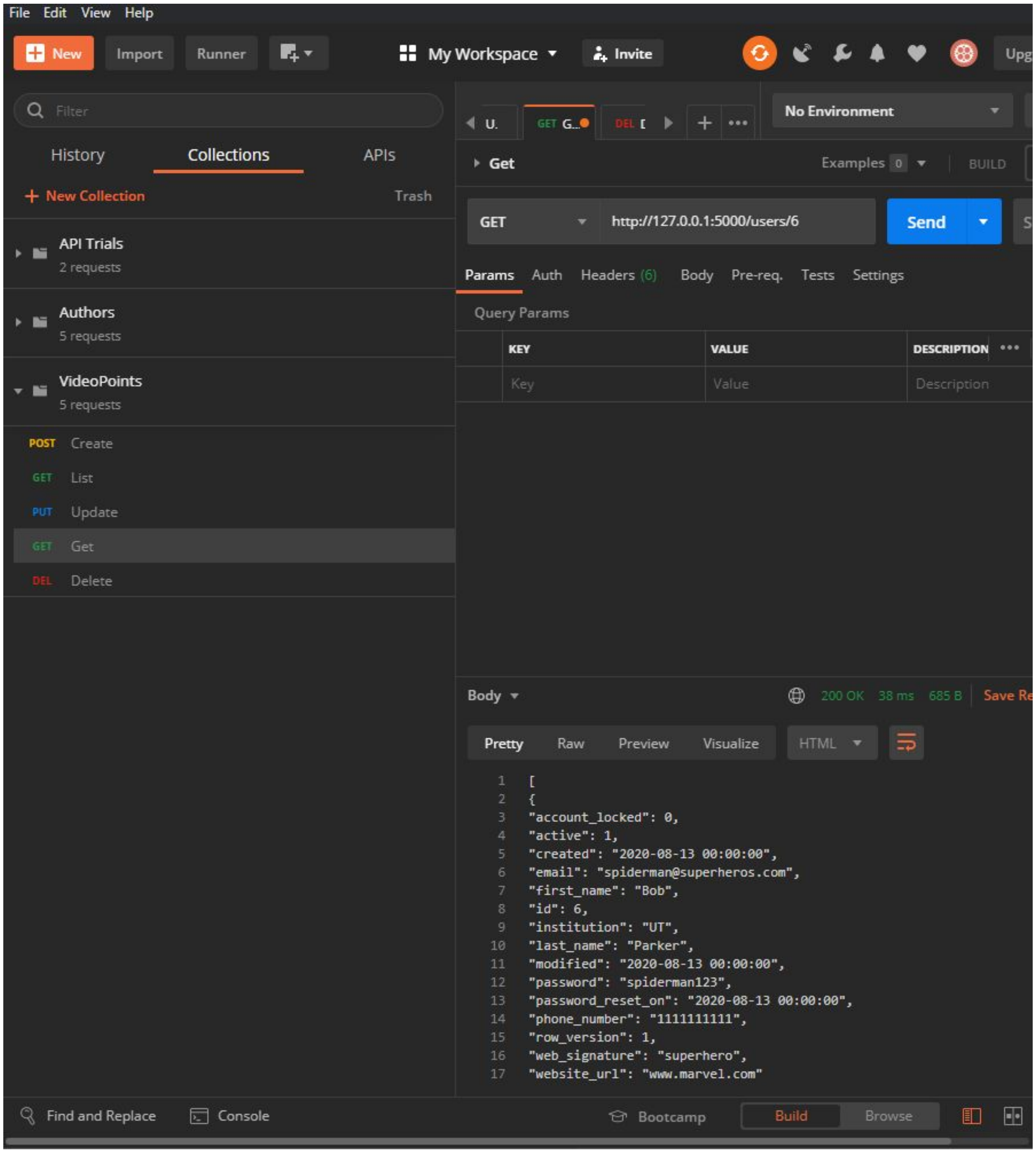
```
CREATE TABLE `videos` (
  `id` int NOT NULL AUTO_INCREMENT,
  `title` varchar(255) NOT NULL,
  `description` varchar(1024) NOT NULL,
  `captions_file_path` varchar(1024) NOT NULL,
  `video_type` varchar(255) NOT NULL,
  `web_link` varchar(1024) CHARACTER SET utf8mb4 COLLATE utf8mb4_0900_ai_ci NOT NULL,
  `file_name` varchar(1024) NOT NULL,
  `file_path` varchar(1024) NOT NULL,
  `file_size` bigint NOT NULL,
  `tags` varchar(1024) NOT NULL,
  `likes` int NOT NULL,
  `dislikes` int NOT NULL,
  `date_uploaded` datetime NOT NULL,
  `number_views` int NOT NULL,
  `video_duration_secs` int NOT NULL,
  `row_version` int NOT NULL,
  `active` bit(1) NOT NULL,
  `modified` datetime NOT NULL,
  `created` datetime NOT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=3 DEFAULT CHARSET=utf8mb4
COLLATE=utf8mb4_0900_ai_ci
```

Appendix - C (Testing) (Postman)



GET http://127.0.0.1:5000/users

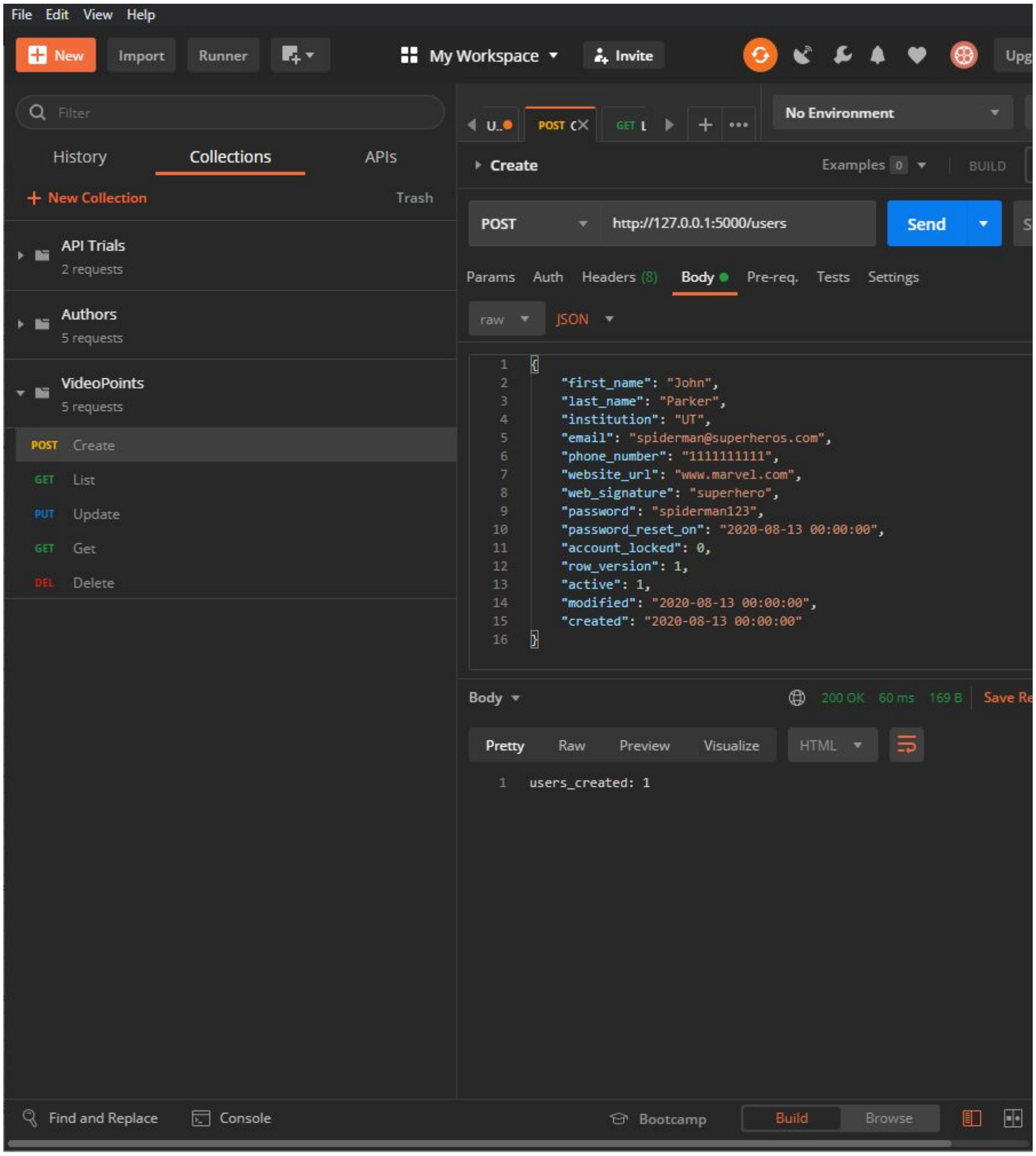
```
[
{
"id": 1,
"first_name": "Kevin",
"last_name": "Jia",
"institution": "University of Texas",
"email": "kevinajia@gmail.com",
"phone_number": "2817748111",
"website_url": "https://www.linkedin.com/in/kevin-jia-610a7a172/",
"web_signature": "Kevin Jia",
"password": "kjpassword",
"password_reset_on": "2020-08-09 01:01:56",
"account_locked": 0,
"row_version": 1,
"active": 1,
"modified": "2020-08-09 01:01:56",
"created": "2020-08-09 01:01:56"
```



GET http://127.0.0.1:5000/users/6

200 OK 38 ms 685 B

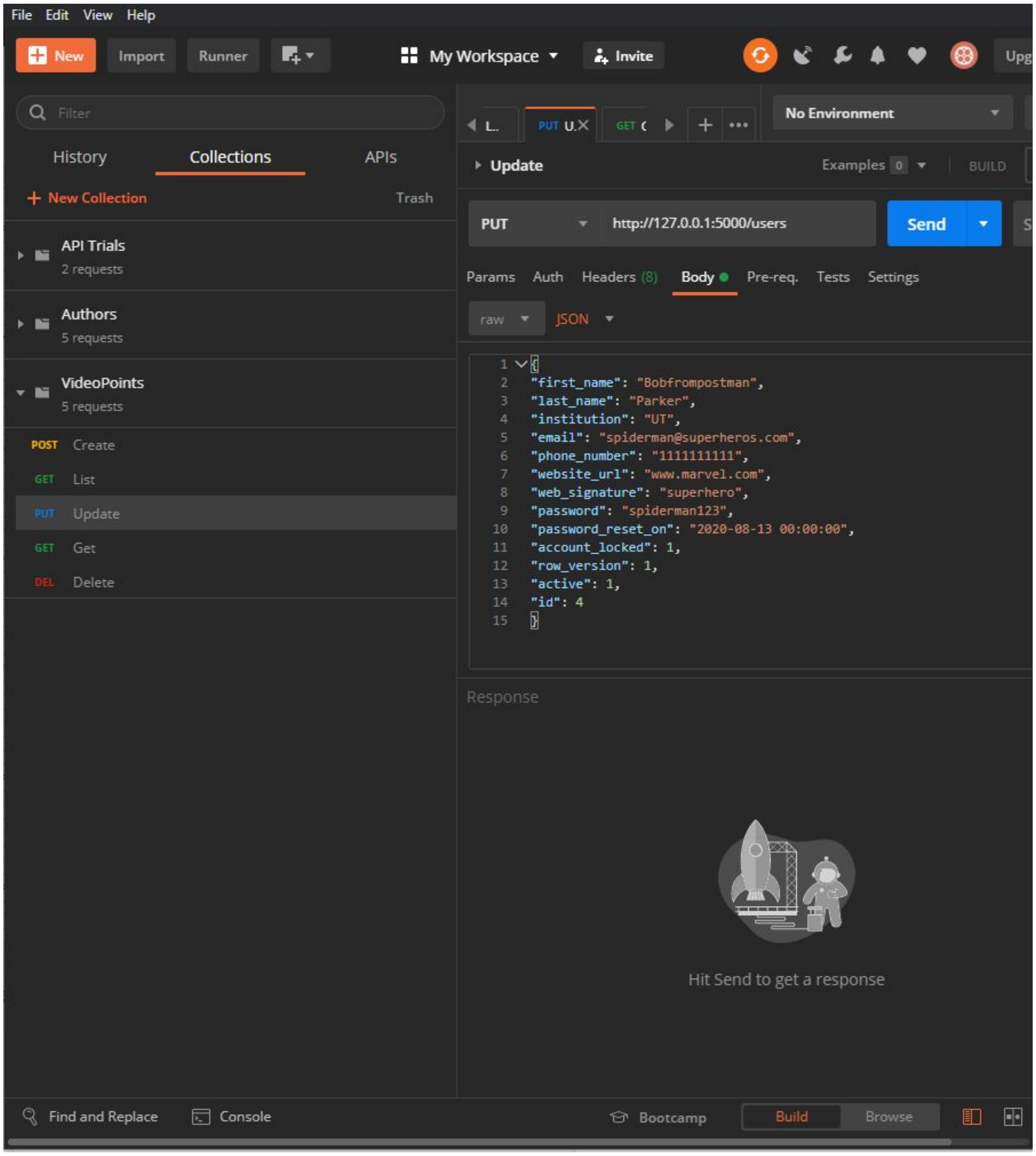
```
[
{
"account_locked": 0,
"active": 1,
"created": "2020-08-13 00:00:00",
"email": "spiderman@superheros.com",
"first_name": "Bob",
"id": 6,
"institution": "UT",
"last_name": "Parker",
"modified": "2020-08-13 00:00:00",
"password": "spiderman123",
"password_reset_on": "2020-08-13 00:00:00",
"phone_number": "1111111111",
"row_version": 1,
"web_signature": "superhero",
"website_url": "www.marvel.com"
```



POST http://127.0.0.1:5000/users

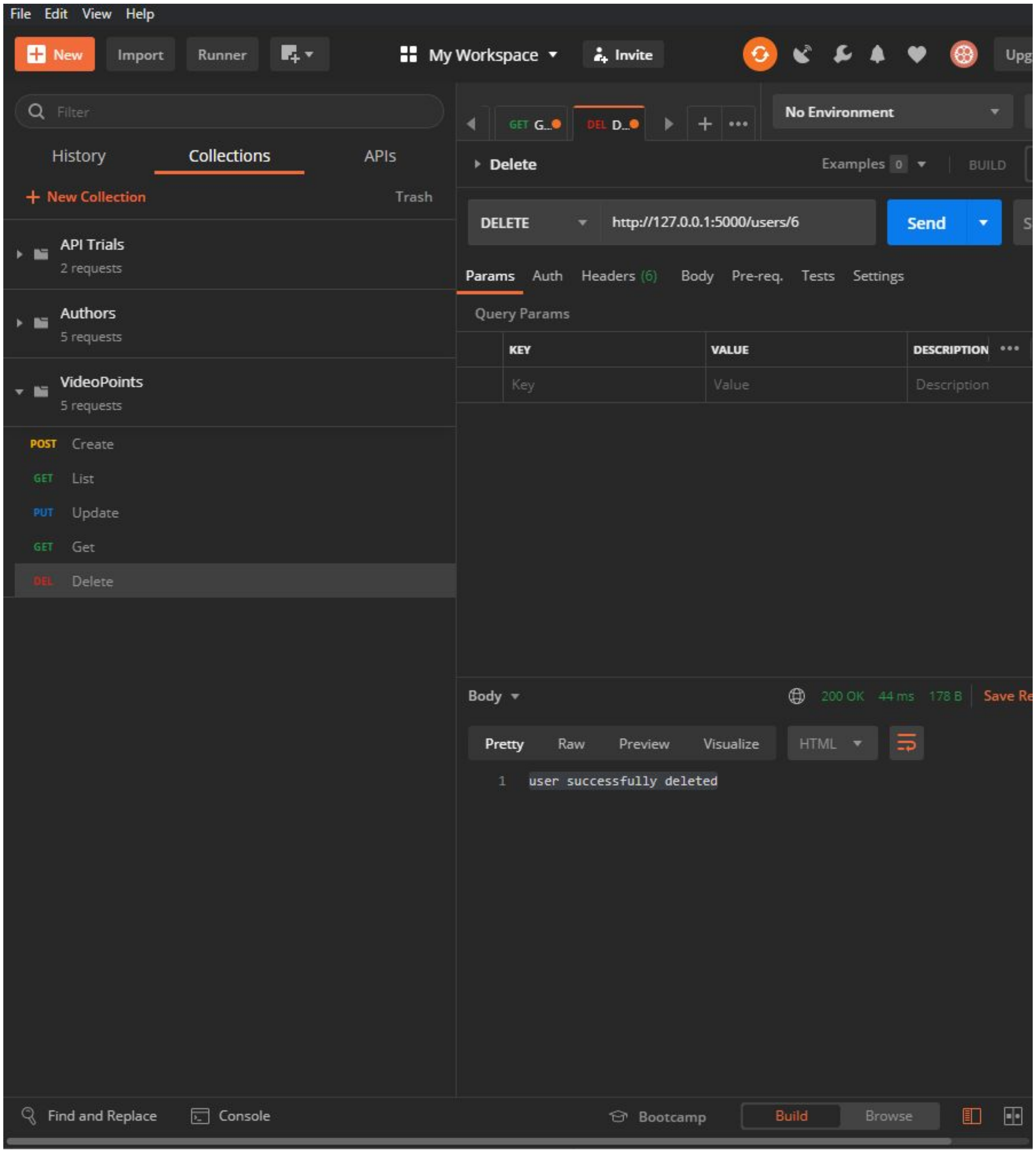
```
{
    "first_name": "John",
    "last_name": "Parker",
    "institution": "UT",
    "email": "spiderman@superheros.com",
    "phone_number": "1111111111",
    "website_url": "www.marvel.com",
    "web_signature": "superhero",
    "password": "spiderman123",
    "password_reset_on": "2020-08-13 00:00:00",
    "account_locked": 0,
    "row_version": 1,
    "active": 1,
    "modified": "2020-08-13 00:00:00",
    "created": "2020-08-13 00:00:00"
}
```

```
users_created: 1
```



PUT http://127.0.0.1:5000/users Send

```
{
"first_name": "Bobfrompostman",
"last_name": "Parker",
"institution": "UT",
"email": "spiderman@superheros.com",
"phone_number": "1111111111",
"website_url": "www.marvel.com",
"web_signature": "superhero",
"password": "spiderman123",
"password_reset_on": "2020-08-13 00:00:00",
"account_locked": 1,
"row_version": 1,
"active": 1,
"id": 4
}
```



DELETE http://127.0.0.1:5000/users/6

200 OK 44 ms 178 B

```
user successfully deleted
```