udivs-backend

REST service for UDIVS hybrid app Host ec2-184-73-153-64.compute-1.amazonaws.com Database dft15go1rt1ppb User wjuuhyleifjbzp

Clone a repository

Use these steps to clone from SourceTree, our client for using the repository command-line free. Cloning allows you to work on your files locally. If you don't yet have SourceTree, download and install first. If you prefer to clone from the command line, see Clone a repository.

1. You'll see the clone button under the **Source** heading. Click that button.
2. Now click **Check out in SourceTree**. You may need to create a SourceTree account or log in.
3. When you see the **Clone New** dialog in SourceTree, update the destination path and name if you'd like to and then click **Clone**.
4. Open the directory you just created to see your repository's files.

You can push your change back to Bitbucket with SourceTree, or you can add, commit, and push from the command line.

Getting the Branches

1. Cordova : *git fetch && git checkout Cordova*
2. DataAnalysis: *git fetch && git checkout DataAnalysis* slack channel is (data analysis)
3. Latex*Src: _git fetch && git checkout Latex_src* slack channel is (paper)
4. REST: *git fetch && git checkout REST*

Bitbucket Cloud Tutorial

Edit a file, create a new file, and clone from Bitbucket in under 2 minutes

When you're done, you can delete the content in this README and update the file with details for others getting started with your repository.

We recommend that you open this README in another tab as you perform the tasks below. You can watch our video for a full demo of all the steps in this tutorial. Open the video in a new tab to avoid leaving Bitbucket.

Git branch

Once you have the other branches you can run *git branch* and it should show you all the branches in a list and highlight the branch you are currently working on, it should look something like this:

Git Checkout

To switch to the branch you need to work on by simply running *git checkout* without the angle brackets. this will switch your branch to the one you specified in

Utilizing *requirements.txt*

Instead of pushing the virtual environment to the repository we will just save our environments settings inside a requirements.txt file. Once you have your virtual environment set up outside of your local git repository use : ***pip install -r requirements.txt*** inside your activated virtual environment. It will automatically update your dependencies. If you make any changes to your local environment use pip ***freez>requirements.txt*** it will overwrite the requirements.txt file with all the python libraries that you have installed in your virtual environment including the ones that you recently added. This will save space and time on your commits and pulls from the repository.

Activating a virtual env

Activating a virtual env will put the virtualenv-specificpython and pip executable into your shell's PATH. You can confirm you're in the virtual env by checking the location of your Python interpreter, it should point to the env directory

On macOS and Linux

1. ***source env/bin/activate*** to activate the environment
2. ***which python*** to locate python interpreter .../env/bin/python

On Windows

1. ***.\env\Scripts\activate***
2. ***where python*** to locate python interpreter .../env/bin/python.exe

As long as your virtual env is activated pip will install packages into that specific environment and you'll be able to import and use packages in your Python application.

Leaving the virtual env

If you want to switch projects or otherwise leave your virtual env, simply run: ***deactivate***

Technology Stack

Server Side

1. SQL DB (not sure which one yet)
2. Python Flask RESTful: https://flask-restful.readthedocs.io/en/latest/
3. SQL Alchemy (if applicable): https://www.sqlalchemy.org/

Client Side

1. Android Develoment Kit

REST API and Flask RESTful

What is REST API

An API is a program that takes in some data and gives back some other data, usually after processing it.

We will be building such programs, so that our users can send us some data, we can process it, and then we can send them something else.

REST is acronym for REpresentational State Transfer. It is architectural style for distributed hypermedia systems and was first presented by Roy Fielding in 2000 in his famous dissertation.

Like any other architectural style, REST also does have it's own 6 guiding constraints which must be satisfied if an interface needs to be referred as RESTful. These principles are listed below.

1. *Client–server*- By separating the user interface concerns from the data storage concerns, we improve the portability of the user interface across multiple platforms and improve scalability by simplifying the server components.
2. *Stateless* – Each request from client to server must contain all of the information necessary to understand the request, and cannot take advantage of any stored context on the server. Session state is therefore kept entirely on the client.
3. *Cacheable* – Cache constraints require that the data within a response to a request be implicitly or explicitly labeled as cacheable or non-cacheable. If a response is cacheable, then a client cache is given the right to reuse that response data for later, equivalent requests.
4. *Uniform interface* – By applying the software engineering principle of generality to the component interface, the overall system architecture is simplified and the visibility of interactions is improved. In order to obtain a uniform interface, multiple architectural constraints are needed to guide the behavior of components. REST is defined by four interface constraints: identification of resources; manipulation of resources through representations; self-descriptive messages; and, hypermedia as the engine of application state.
5. *Layered system* – The layered system style allows an architecture to be composed of hierarchical layers by constraining component behavior such that each component cannot "see" beyond the immediate layer with which they are interacting.
6. *Code on demand (optional)* – REST allows client functionality to be extended by downloading and executing code in the form of applets or scripts. This simplifies clients by reducing the number of features required to be pre-implemented.

https://restfulapi.net/

What is Flask RESTful

Flask-RESTful is an extension for Flask that adds support for quickly building REST APIs. It is a lightweight abstraction that works with your existing ORM/libraries. Flask-RESTful encourages best practices with minimal setup.

https://flask-restful.readthedocs.io/en/latest/quickstart.html#full-example

Summary of HTTP Methods for RESTful APIs https://restfulapi.net/http-methods/

```
|-------------------------------------------------------------------------
--------------------------|
|  HTTP METHOD  |    CRUD       | ENTIRE COLLECTION (E.G. /USERS)   |
SPECIFIC ITEM (E.G. /USERS/123) |
|---------------|---------------|-----------------------------------|-----
-------------------------- |
|               |               |                                   |
|
|               |               |201 (Created), 'Location' header   |
|
|     POST      |   CREATE      |with link to /users/{id} containing|
Avoid using POST on          |
|               |               |new ID.                            |
single resource              |
|               |               |                                   |
|
|---------------|---------------|-----------------------------------|-----
--------------------------|
|               |               |                                   |
|
|               |               |  200 (OK), list of users. Use     | 200
(OK), single user. 404      |
|      GET      |     READ      |  pagination,sorting and filtering | (Not
Found), if ID not found or |
|               |               |  to navigate big lists.           |
invalid.                     |
|               |               |                                   |
|
|---------------|---------------|-----------------------------------|-----
--------------------------|
|               |               |                                   |
|
|               |               |  404 (Not Found), unless you want |
|
|               |               |  to update every resource in the  | 200
(OK) or 204 (No Content).   |
|      PUT      |Update/Replace |  entire collection of resource.   | Use
404 (Not Found), if ID not  |
|               |               |                                   |
found or invalid.            |
|               |               |                                   |
|
|---------------|---------------|-----------------------------------|-----
--------------------------|
|               |               |                                   |
|
|               |               |                                   |
|
|      PATCH    | Update/Modify |  404 (Not Found), unless you want | 200
(OK) or 204 (No Content).   |
|               |               |  to modify the collection itself. | Use
404 (Not Found), if ID not  |
|               |               |                                   |
```

```
found or invalid.               |
|               |               |                                     |
|
|---------------|---------------|-------------------------------------|-----
--------------------------|
|               |               |                                     |
|
|               |               | 404 (Not Found), unless you want  |
200 (OK). 404 (Not Found)    |
|     DELETE    |    DELETE     | to delete the whole collection —  |
if ID not found or invalid.  |
|               |               | use with caution.                   |
|
|               |               |                                     |
|
|---------------|---------------|-------------------------------------|-----
--------------------------|
```

Sample Rest API

```
"""

 Rest API gives resources to the client using http requests which run
through a url on the browser
 Many professional companies uses REST inside MVC Frame Works such as
Django, Rails, Node js, PHP symphoney,...,ect.
 Flask is what is known as a microservice, it is not a true MVC frame work
and is not as bloated with libraries and does
 not scale as well as a true MVC. MVC is normally used for web sites, but
sense this application is on a phone where processing
 will be done on the device as apposed to the DOM on browsers makes
Flask/Flask Restful a sutiable tool for the job

we start with writing a class that inherits from the resouce class of the
Flask RESTful frame work

"""

class Users(Resource):
"""
Rest API's usually will implemnt a subset of the following functions, in
Flask RESTful, they need to be explicitly overloaded
as shown below, and self must be passed as a parameter.
"""
    def get(self):
        pass

    def post(self):
```

```
        pass

    def put(self):
        pass

    def delete(self):
        pass
```