
!BETSOL®

BETSOL CAMPUS WORKSHOP HANDS-ON
!BETSOL GUIDE

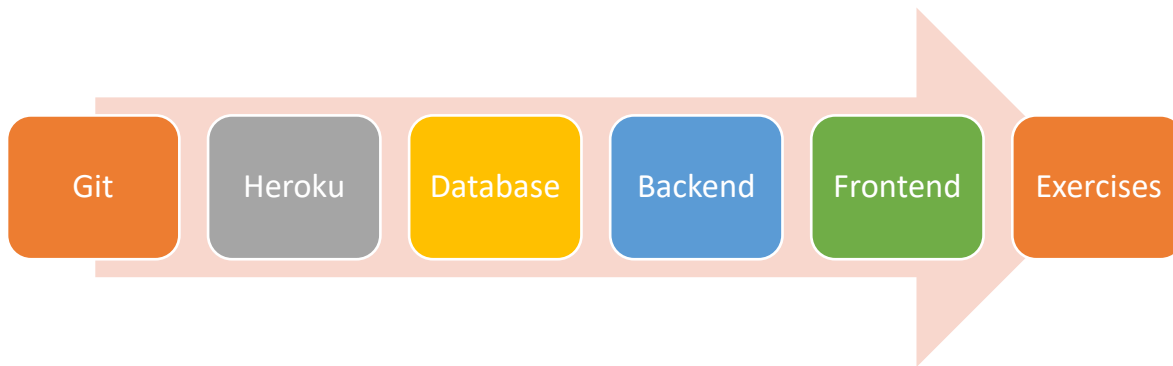
WORKSHOP HANDS-ON

Version 1.0

A Data Management and Intelligent Automation
Company

CONTENTS

Campus Workshop.....	3
What will we be Learning in the Workshop?	3
Prerequisites for the Workshop	3
Git	4
Prerequisites.....	4
Tasks to be performed.....	4
Fork & CLONE, the repository.....	4
Clone Repo locally.....	5
Add your first commit.....	5
Heroku and Database Setup	7
Prerequisites.....	7
Tasks to be performed.....	7
Create a get-my-db app in Heroku and get the credentials	7
Configure The Heroku in PGAdmin	11
Create a table	13
Backend Development.....	15
Prerequisites.....	15
The task to be Performed	15
Installing Postman (For Reference):	15
Fork and clone the repository.....	15
deployment	17
Remote Deployment.....	18
LOCAL DEPLOYMENT *Optional	25
Frontend Development	31
Prerequisites.....	31
Tasks to be Performed	31
FORK & CLONE	31
Update the backend endpoint URL	32
Deploy frontend on Heroku	33
Exercises:	37
EXERCISE 1: Marking a to do item as complete	37
EXERCISE 2: EDITING A TO DO ITEM	37



WHAT WILL WE BE LEARNING IN THE WORKSHOP?

We will build a simple **To-do app** and deploy the database and backend on the cloud. We will be using Python & Flask framework to build the backend, React JS to build the frontend, PostgreSQL for the database, and finally Heroku to deploy our finished database and backend on the cloud.

Full Stack Technologies that you will learn

- Flask a light-weight Python framework will be used to build a RESTful API which will serve as the backend
- ReactJS as JavaScript library will be used to build the frontend UI application
- PostgreSQL as a database engine managed by pgAdmin a Database Management System (DBMS).
- Heroku a Platform-as-a-Service cloud platform to realize our full stack application.

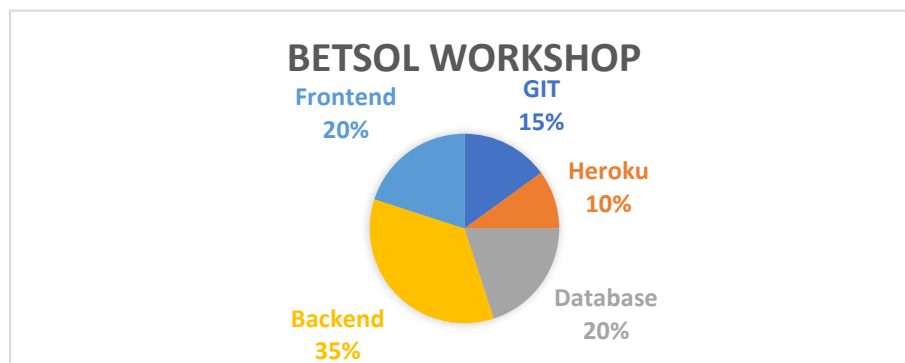
PREREQUISITES FOR THE WORKSHOP

Cloud Accounts

- GitHub Account SignUp
- Heroku Account SignUp

Installable on your PC

- Heroku CLI Installation
- pgAdmin Download Page
- Python
- NodeJS and NPM



GIT

Sam has started working on a To-do application. He decides to work with front-end and back-end projects separately as they are based on different technology stacks.

To keep track of all the changes made to the code and have the flexibility to revert to old parts of his code, Sam decides to use Git version control for the front-end and back-end projects in separate repositories.

Help Sam to initialize a local repository and fetch the project repositories to his computer to start working on the Full stack app.

PREREQUISITES

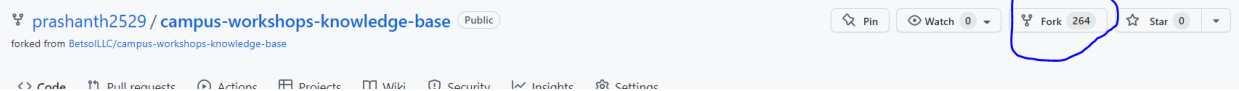
- Git installed on the computer.

TASKS TO BE PERFORMED.

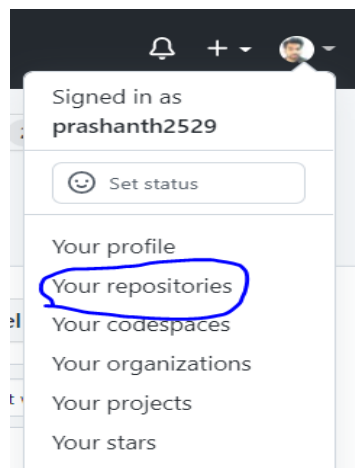
1. Fork & clone the repository.
2. Clone Repo locally
3. Add your first commit

FORK & CLONE, THE REPOSITORY.

1. Login to GitHub
2. Navigate to the repository with the reference material for this workshop -> BetsolLLC/campus-workshops-knowledge-base
3. Fork the repository using <https://github.com/BetsolLLC/campus-workshops-knowledge-base> ,Click on the Fork button fork to copy the repository to your account.



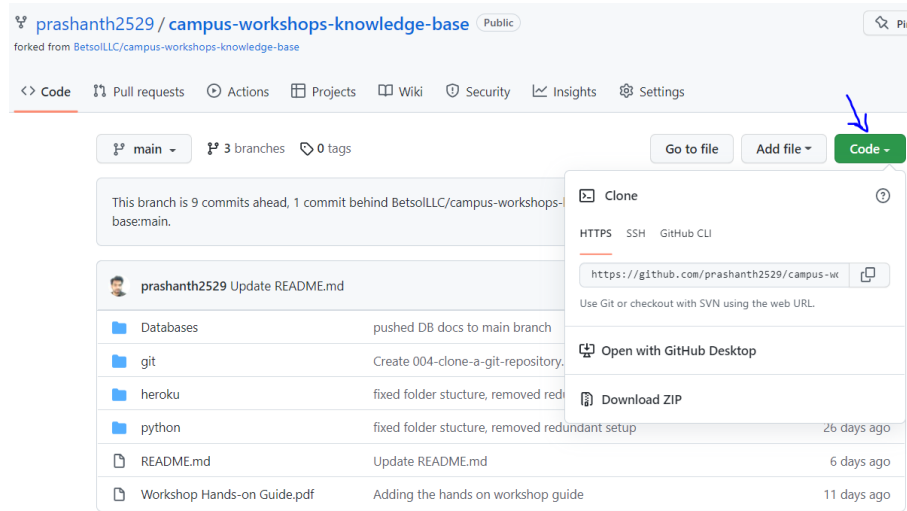
4. Goto your account and you can find the copy of your repository and click on the campus-workshops-knowledge-base



campus-workshops-knowledge-base Public
Forked from BetsolLLC/campus-workshops-knowledge-base
Knowledge Base for Campus Workshops
264 Updated 5 days ago

Star

- Click on the code and copy the link present in it.



About this repository [BetsolLLC/campus-workshops-knowledge-base](#)

This repository contains all the relevant guides and reference material that you need to complete in this Workshop. It is important that you have forked the **BetsolLLC/campus-workshops-knowledge-base** repository into your GitHub Account before cloning it into your PC's local repository.

Note: If you are using git with your GitHub account for the first time then you can [click here](#), which details the steps to configure your git settings so that you can start pulling and pushing data to your remote repository

CLONE REPO LOCALLY

- Open the Git Bash in your preferred working directory.
- Setup your local git config to contribute with your name and email (it is important that the email matches with the email used to signup with GitHub)


```
git config --global user.name "Firstname Lastname"
git config --global user.email "your_email@youremail.com"
```
- Create a directory called to-do-app and change the working directory to it


```
mkdir to-do-app
cd to-do-app
```
- Using the git clone command clone the repository


```
git clone <copied-repo-link>
```

 - If you face any SSL issues in this step 4. run the following command and run step 4. again before continuing to step 5.


```
git config http.sslVerify "false"
```
- Verify that you have a folder called campus-workshops-knowledge-base/ (You may use ls command on Linux, macOS and dir command on Windows)
- Change directory into the folder


```
cd campus-workshops-knowledge-base/
```

ADD YOUR FIRST COMMIT

- Edit the README.md file using any text editor.
- To check the status of your changes to the repository:


```
git status
```
- stage your changes to the staging area using the git add command



```
git add .    or  
git add README.md
```

4. Commit files in Git: The next state for a file after the staged state is the committed state. The message in the " " is given so that the other users can read the message and see what changes you made

```
git commit -m "First commit"
```

5. The git push command pushes the changes in your local repository up to the remote repository you specified as the origin.

```
git push -u origin main #pushes changes to origin
```

NOTE: Each time you make changes that you want to be reflected on GitHub, the following are the most common flow of commands:

```
git add .  
git status #Lists all new or modified files to be committed  
git commit -m "Second commit"  
git push -u origin master
```

HEROKU AND DATABASE SETUP

Sam is well-versed with Git. He wants to use Heroku a Platform-as-a-Service to host his TO-DO application. Initially, he will be starting with creating an account and then an app in Heroku on which he will create a PostgreSQL database. Using the credentials obtained for the database from Heroku Sam will be configuring pgAdmin a Database Management System (DBMS) to interact with the database from his PC.

PREREQUISITES

- Heroku account
- Installed pgAdmin Client should be installed.

TASKS TO BE PERFORMED

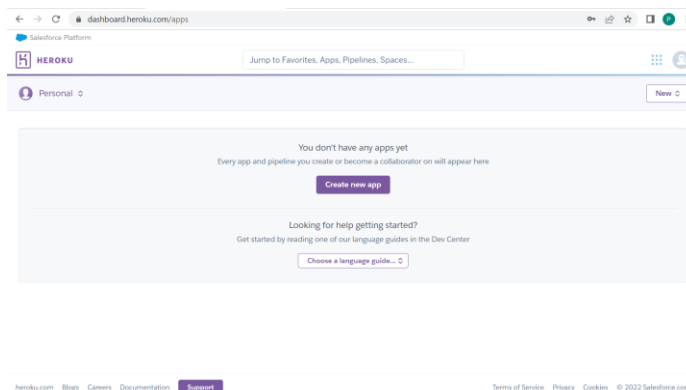
1. Create an account in Heroku
2. Create a get-my-db app in Heroku and get the credentials
3. Configure the Heroku in PGAdmin
4. Create a table

CREATE A GET-MY-DB APP IN HEROKU AND GET THE CREDENTIALS

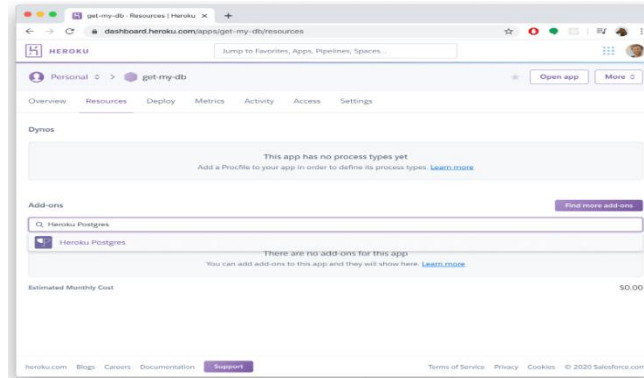
1. Create an Account: <https://signup.heroku.com/>

Note: You must leave the Company Name blank.

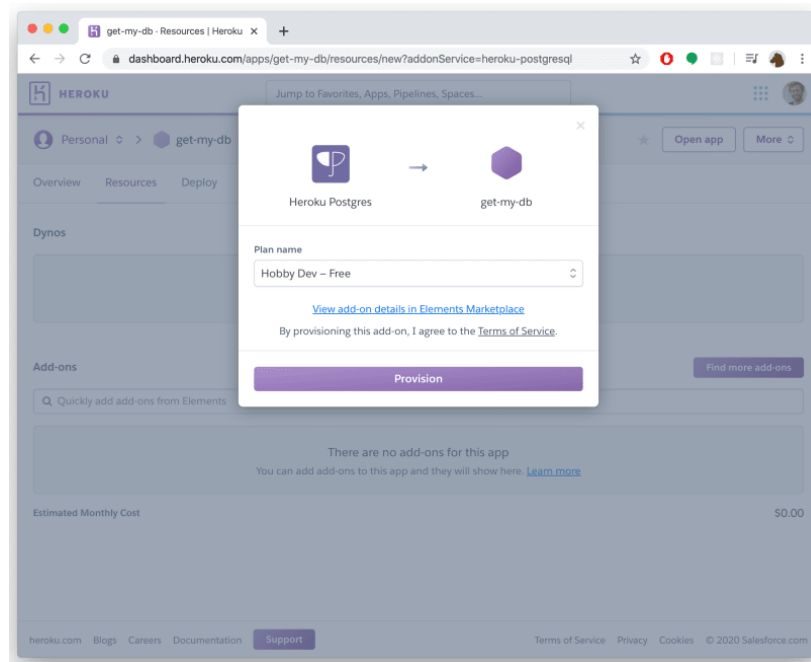
2. Log in to Heroku and go to the Application dashboard.
3. Create a new app with the name get-my-db (choose the available name)



4. Navigate to the **Resources** tab of the Application dashboard & Add Heroku Postgres Add-on

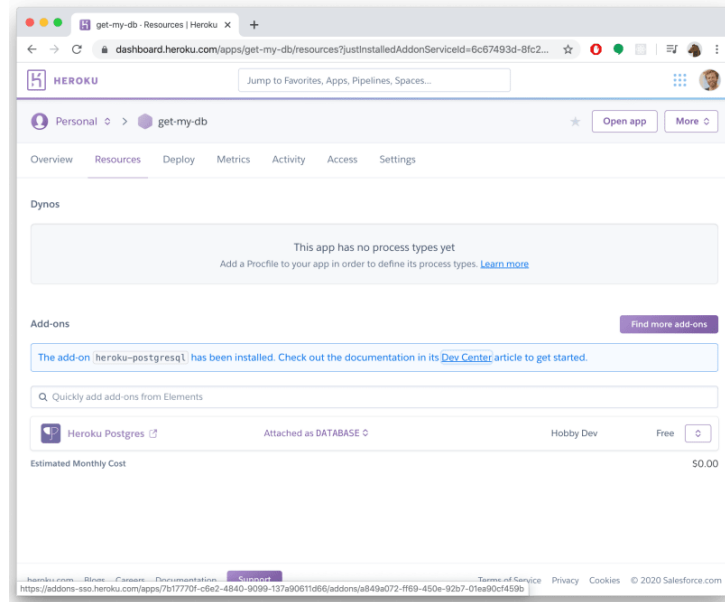


5. Select Hobby Dev - Free in the Plan Name Dropdown

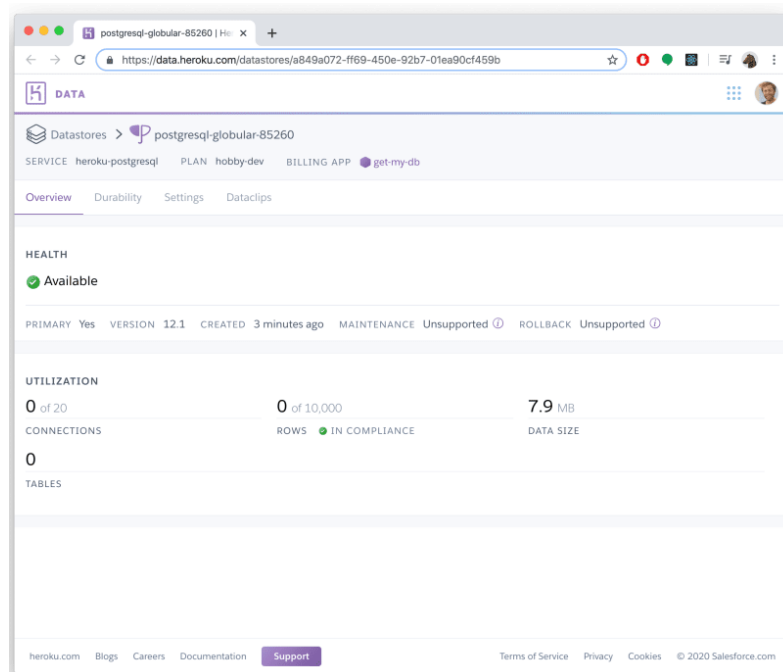


6. Get the database credentials and connection URL

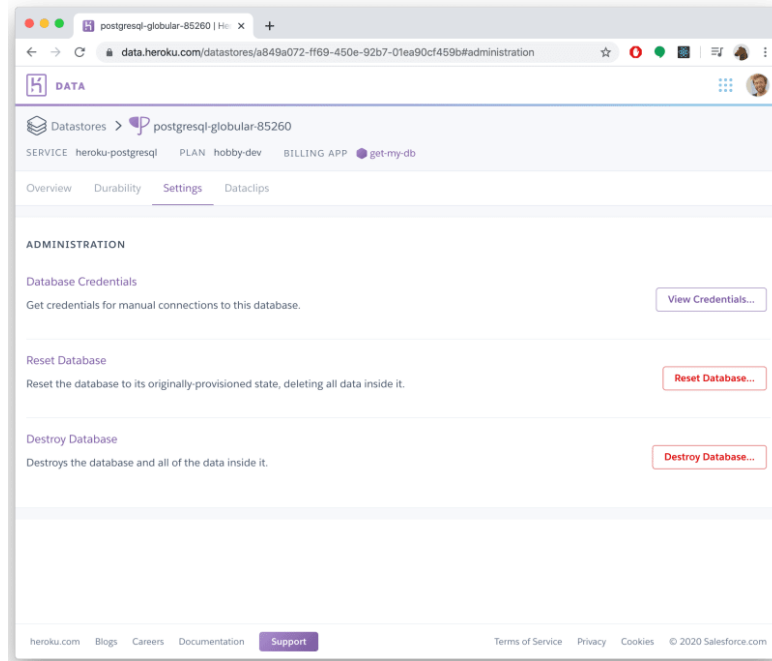
- Navigate to the Resources tab again
- Select Heroku Postgres



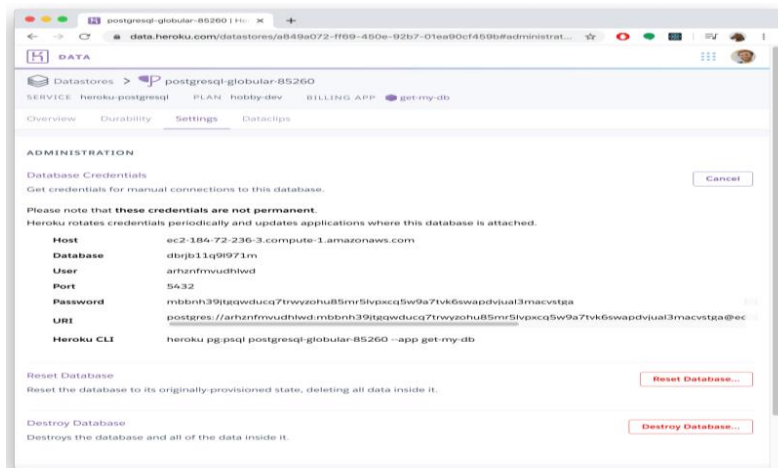
- Should look something like this



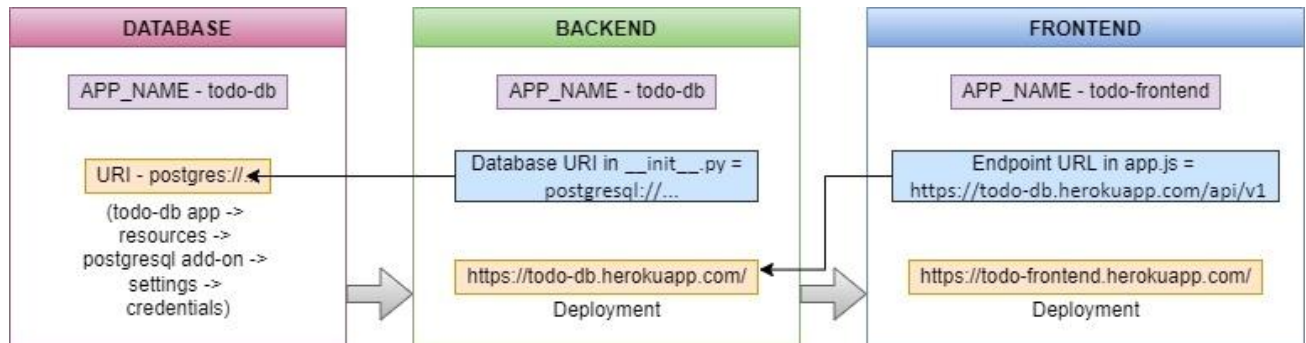
- Select Settings Tab



- Click on View Credentials

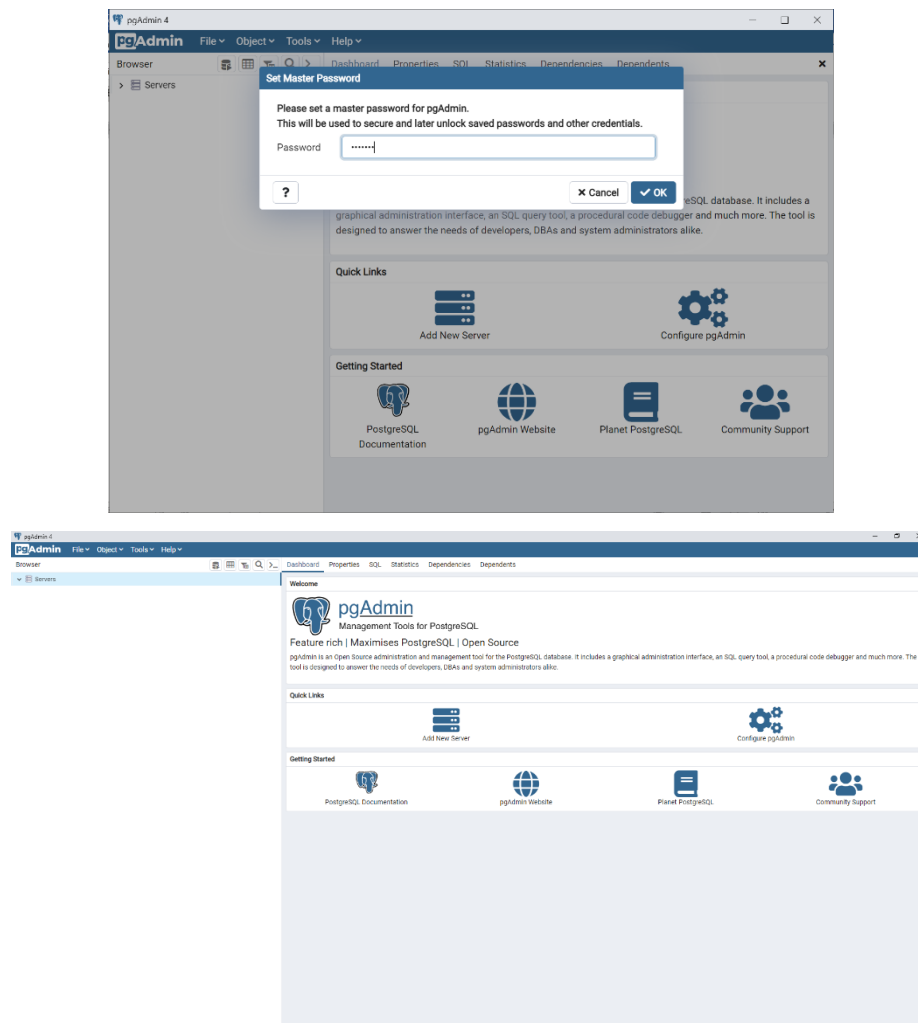


- Note down the credentials in notepad or keep the tab open for the next steps.
Host, Database, User, Port, Password, URI, Heroku CLI

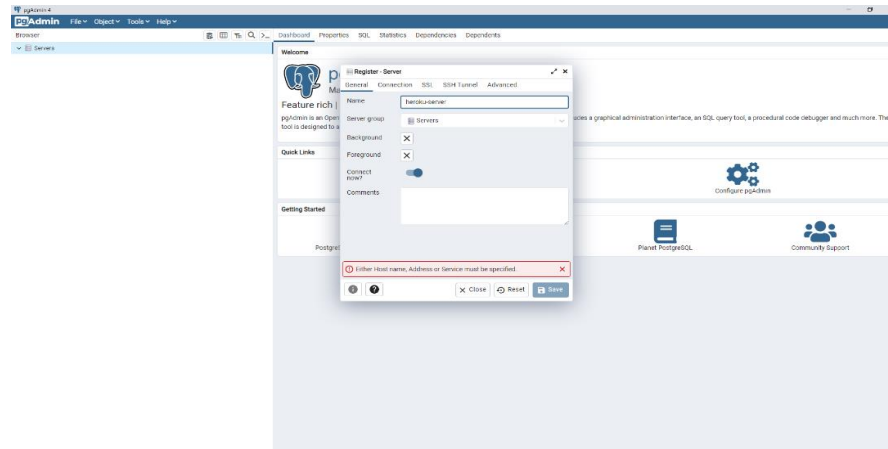


CONFIGURE THE HEROKU IN PGADMIN

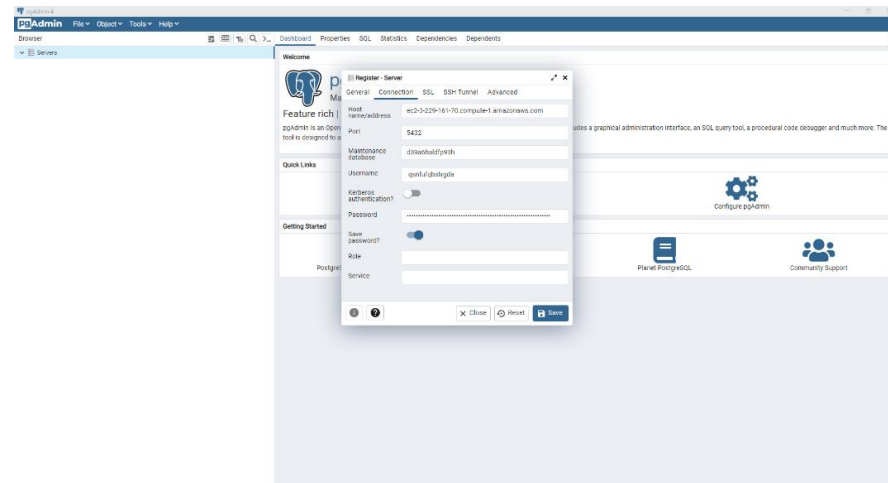
1. Open pgAdmin and choose any master password of your wish.



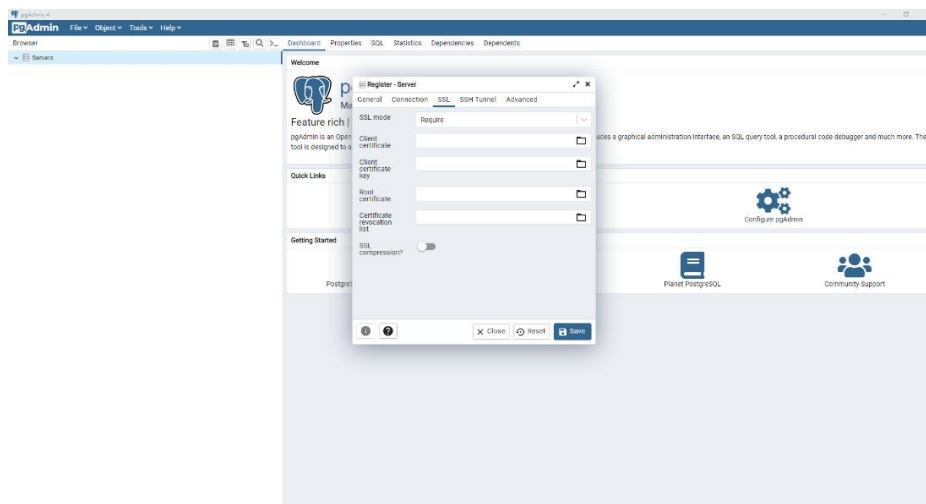
2. Right-Click on Servers > Register > Server and enter a name of your choice



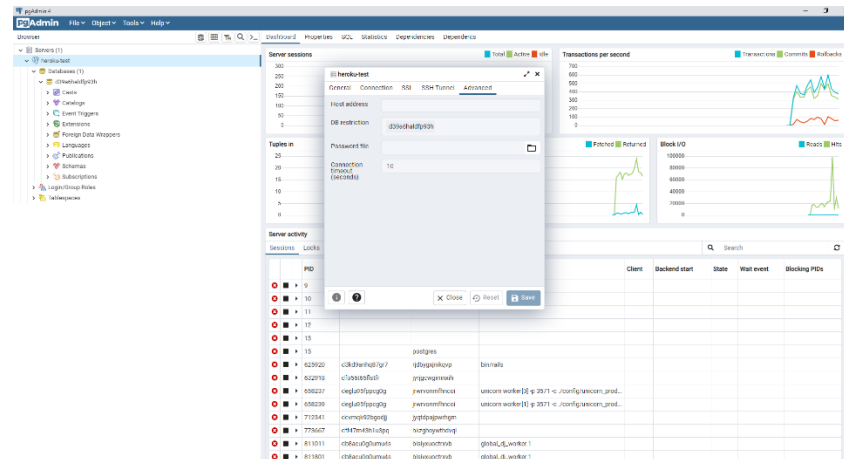
3. Navigate to the Connection Tab and enter the connection information collected at the end of Step 5



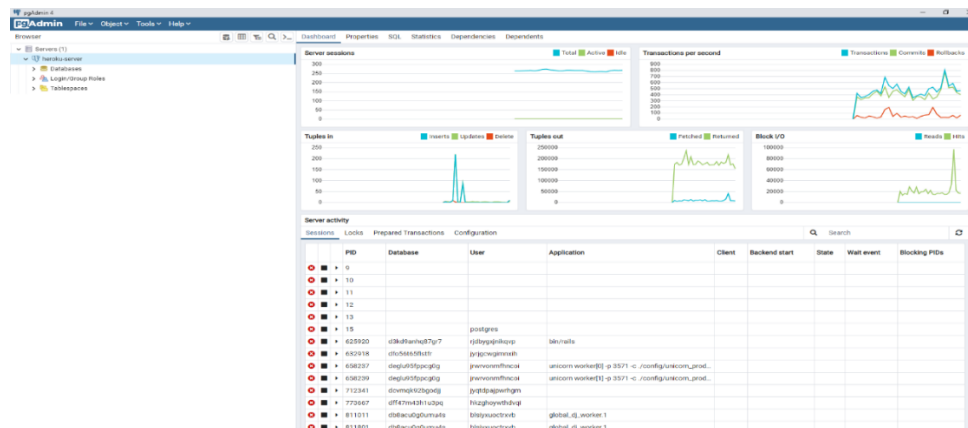
4. Navigate to the SSL Tab and set SSL Mode as required



- Navigate to the Advanced tab and set DB Restriction to the name of the database you previously copied at the end of step 5

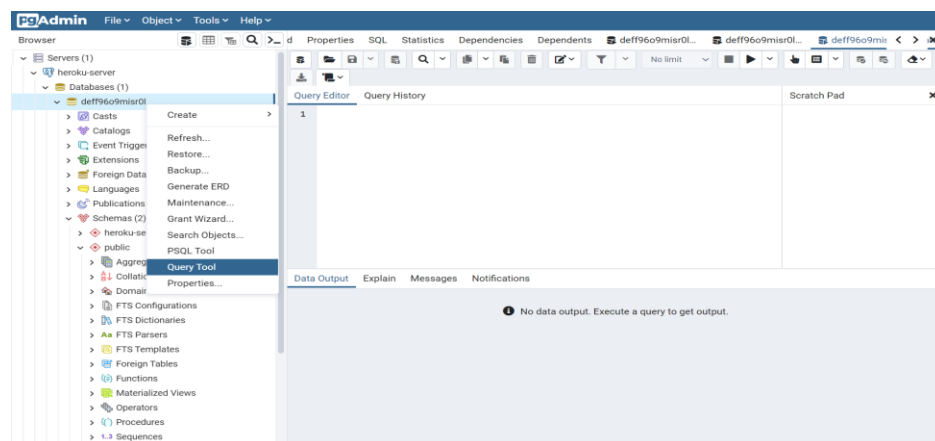


- Click on Save to see a similar dashboard



CREATE A TABLE

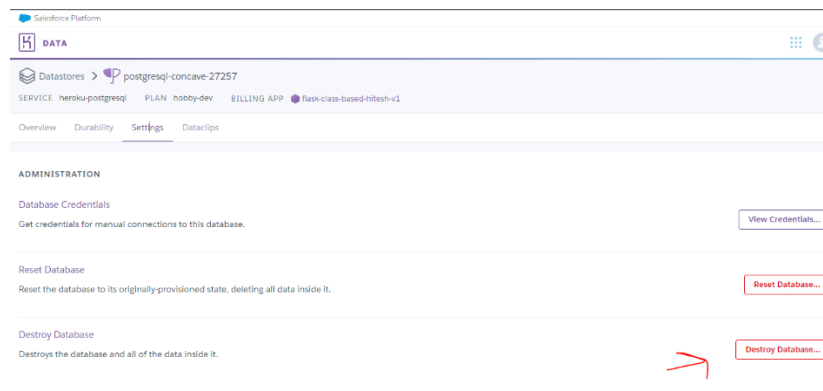
- Right-click on the database click on Query Tool and type the queries.



2. For demonstrating our Database, let us create a TODO table. below is the command by which we can create the two tables:

```
CREATE TABLE TODO
(
  id SERIAL PRIMARY KEY,
  title VARCHAR(50) NOT NULL,
  complete BOOLEAN,
  date_modified TIMESTAMP
);
```

Note: If you face issues with respect to receiving error messages, which contain the words like, *“too many clients or if it is internal server error”* then try to destroy the database(delete). Create a new database in Heroku.



BACKEND DEVELOPMENT

Sam has just finished creating the Database, now that he wants to get on to Backend Development using Python. For this, he will be start working with Python and uses the database that he has created in his TO-DO Application.

PREREQUISITES

- Installed Python
- Installed PyCharm
- Installed Postman
- Install Heroku CLI

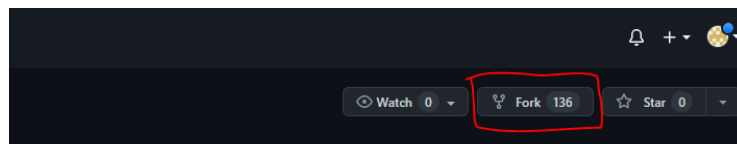
THE TASK TO BE PERFORMED

1. Fork and clone the repository
2. Deployment
 - 2.1. Remote Deployment
 - 2.1.1. Modify the Database URI
 - 2.1.2. Deploying the App
 - 2.1.3. Check the response using Postman
 - 2.2. Local Deployment (Optional)

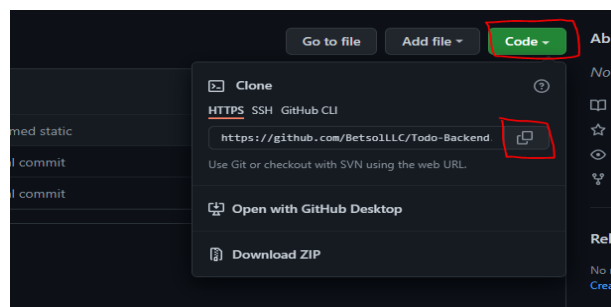
INSTALLING POSTMAN (FOR REFERENCE):

1. Go to Postman (Download)
2. Select your OS and click on the Download option
3. Select and run the .exe file to install Postman.
4. Refer to [Postman Documentation](#) for any queries
5. **You can also use the web version of [Postman](#) (localhost requests will not work here)**

FORK AND CLONE THE REPOSITORY

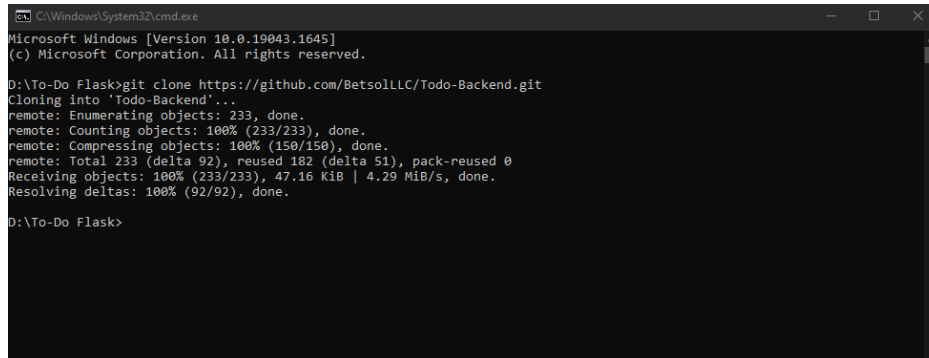


1. Click on the Fork Button on the Top-Right
2. Then go to yourUserName/ToDo-Backend to clone the repository (Clone the code from the forked repository under your name. **NOT the original repository that you forked previously**).
3. Click code and copy



4. Go to your desired folder preferably D drive and open the git-bash
5. Type the command

```
git clone <linkyoucopiedfromyourrepository>
```

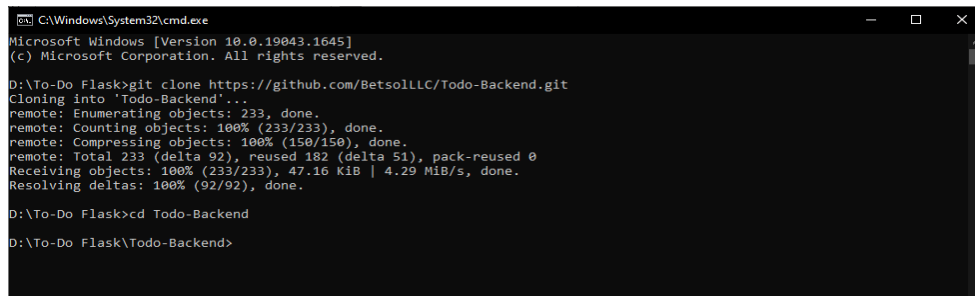


```
C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.19043.1645]
(c) Microsoft Corporation. All rights reserved.

D:\To-Do Flask>git clone https://github.com/BetsolLLC/ToDo-Backend.git
Cloning into 'ToDo-Backend'...
remote: Enumerating objects: 233, done.
remote: Counting objects: 100% (233/233), done.
remote: Compressing objects: 100% (150/150), done.
remote: Total 233 (delta 92), reused 182 (delta 51), pack-reused 0
Receiving objects: 100% (233/233), 47.16 KiB | 4.29 MiB/s, done.
Resolving deltas: 100% (92/92), done.

D:\To-Do Flask>
```

6. Move into the folder using the cd command

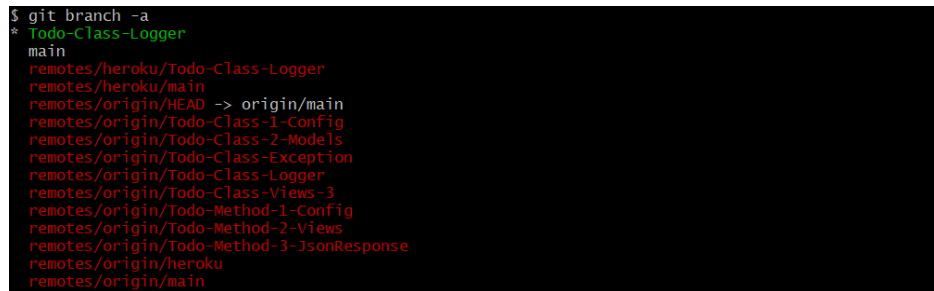


```
C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.19043.1645]
(c) Microsoft Corporation. All rights reserved.

D:\To-Do Flask>git clone https://github.com/BetsolLLC/ToDo-Backend.git
Cloning into 'ToDo-Backend'...
remote: Enumerating objects: 233, done.
remote: Counting objects: 100% (233/233), done.
remote: Compressing objects: 100% (150/150), done.
remote: Total 233 (delta 92), reused 182 (delta 51), pack-reused 0
Receiving objects: 100% (233/233), 47.16 KiB | 4.29 MiB/s, done.
Resolving deltas: 100% (92/92), done.

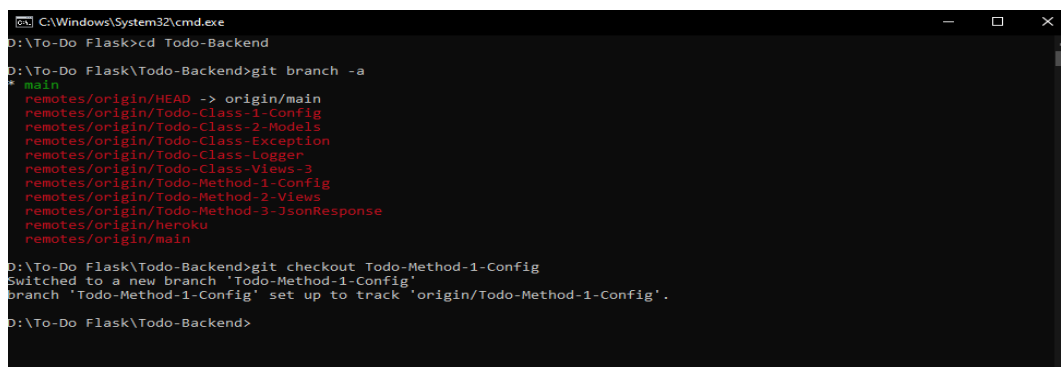
D:\To-Do Flask>cd ToDo-Backend
D:\To-Do Flask\ToDo-Backend>
```

7. Use `git branch -a` to see all branches



```
$ git branch -a
* Todo-Class-Logger
main
remotes/heroku/ToDo-Class-Logger
remotes/heroku/main
remotes/origin/HEAD -> origin/main
remotes/origin/ToDo-Class-1-Config
remotes/origin/ToDo-Class-2-Models
remotes/origin/ToDo-Class-Exception
remotes/origin/ToDo-Class-Logger
remotes/origin/ToDo-Class-Views-3
remotes/origin/ToDo-Method-1-Config
remotes/origin/ToDo-Method-2-Views
remotes/origin/ToDo-Method-3-JsonResponse
remotes/origin/heroku
remotes/origin/main
```

8. Use `git checkout Todo-Class-Logger` to change to the **Todo-Class-Logger** branch



```
C:\Windows\System32\cmd.exe
D:\To-Do Flask>cd ToDo-Backend
D:\To-Do Flask\ToDo-Backend>git branch -a
* main
remotes/origin/HEAD -> origin/main
remotes/origin/ToDo-Class-1-Config
remotes/origin/ToDo-Class-2-Models
remotes/origin/ToDo-Class-Exception
remotes/origin/ToDo-Class-Logger
remotes/origin/ToDo-Class-Views-3
remotes/origin/ToDo-Method-1-Config
remotes/origin/ToDo-Method-2-Views
remotes/origin/ToDo-Method-3-JsonResponse
remotes/origin/heroku
remotes/origin/main

D:\To-Do Flask\ToDo-Backend>git checkout Todo-Method-1-Config
Switched to a new branch 'ToDo-Method-1-Config'
branch 'ToDo-Method-1-Config' set up to track 'origin/ToDo-Method-1-Config'.

D:\To-Do Flask\ToDo-Backend>
```


- Verify the files present in the directory to be the same as the content given below.

PS: The below command was executed using “*git bash*”.

If you are using cmd or Powershell use *dir* instead of *ll*

```
$ ll
total 52
-rw-r--r-- 1 Hitesh.Balegar 1049089 22 Apr 20 01:03 Procfile
-rw-r--r-- 1 Hitesh.Balegar 1049089 23267 Apr 20 02:09 'Todo APP.postman_collection.json'
drwxr-xr-x 1 Hitesh.Balegar 1049089 0 Apr 20 01:25 __pycache__/
-rw-r--r-- 1 Hitesh.Balegar 1049089 4922 Apr 19 23:03 readme.md
-rw-r--r-- 1 Hitesh.Balegar 1049089 434 Apr 20 01:00 requirements.txt
drwxr-xr-x 1 Hitesh.Balegar 1049089 0 Apr 20 01:46 static/
-rw-r--r-- 1 Hitesh.Balegar 1049089 16154 Apr 19 23:42 static.log
-rw-r--r-- 1 Hitesh.Balegar 1049089 335 Apr 19 23:03 steps.txt
-rw-r--r-- 1 Hitesh.Balegar 1049089 53 Apr 19 23:55 wsgi.py
```

- Verify the content present within the file `cat requirements.txt` to be the same as given below.

PS: The below command was executed using “*git bash*”.

If you are using cmd or Powershell open and check the file contents directly using any file explorer .

```
$ cat requirements.txt
alembic==1.7.7
aniso8601==9.0.1
click==8.1.0
colorama==0.4.4
Flask==2.1.0
Flask-Cors==3.0.10
Flask-Migrate==3.1.0
Flask-RESTful==0.3.9
Flask-SQLAlchemy==2.5.1
Flask-WTF==0.9.3
greenlet==1.1.2
gunicorn==20.1.0
importlib-metadata==4.11.3
itsdangerous==2.1.2
Jinja2==3.1.1
Mako==1.2.0
MarkupSafe==2.1.1
psycopg2-binary==2.9.3
pytz==2022.1
six==1.16.0
SQLAlchemy==1.4.32
werkzeug==2.1.0
WTForms==3.0.1
zipp==3.7.0
```

Once verified, we are good to go. Henceforth the branch “**Todo-Class-Logger**” would be used to demonstrate the **Class-based views approach** to a Flask application and if interested you can switch the branch “**Todo-Method-3-JsonResponse**” to view the **Method-based approach**.

DEPLOYMENT



The backend application we would be developing can be deployed in two environments local and remote deployment.

1. Remote Deployment:
 - a. A This type of deployment mainly covers the starting up of the backend application process in the remote environment which can be a cloud server, offshored system/server, etc.
 - b. The app deployment, in this case, covers the starting up of flask in the remote environment of Heroku and trying out all the APIs listed in the resource URLs.
2. Local Deployment:
 - a. This type of deployment mainly covers the starting up of the backend application process in the local environment which can be one's server, laptop, on-premises server, etc.
 - b. The app deployment in this case covers the starting up of flask in one's system and trying out all the APIs listed in the resource URLs.


For all purposes for the Workshop, **please get started with the Remote Deployment first and then try the local Deployment** if you find time.


REMOTE DEPLOYMENT

INSTALL HEROKU CLI (IF NOT DONE ALREADY)


In this section you can start the deployment on Heroku without setting up your local environment, we can proceed with deploying backend on Heroku. For this, follow the below steps:


1. Install Heroku CLI from <https://devcenter.heroku.com/articles/heroku-cli> using the appropriate method based on your OS.
2. Go for the **64-bit installer for Windows Operating System**.

 The Windows installers display a warning titled "Windows protected your PC" to some users. To run the installation when this warning shows, click "More info", verify the publisher as "salesforce.com, inc", then click the "Run anyway" button.

 **macOS**

```
$ brew tap heroku/brew && brew install heroku
```

 **Windows**
Download the appropriate installer for your Windows installation:
[64-bit installer](#)
[32-bit installer](#)

 **Ubuntu 16+**
Run the following from your terminal:

```
$ sudo snap install --classic heroku
```

3. Verify the installation using the command

heroku version

```
$ heroku version
» Warning: heroku update available from 7.53.0 to 7.60.1.
heroku/7.53.0 win32-x64 node-v12.21.0
```

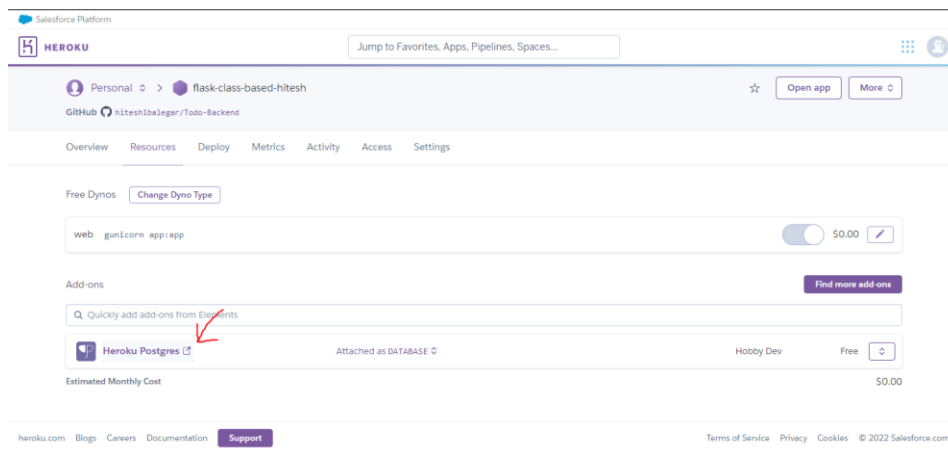
Note: you can ignore the warning if your version is ≥ 7.53

If you do not get the above output after you install the cli tool, try to close and reopen your terminal and try again, if that doesn't work try to restart your system.

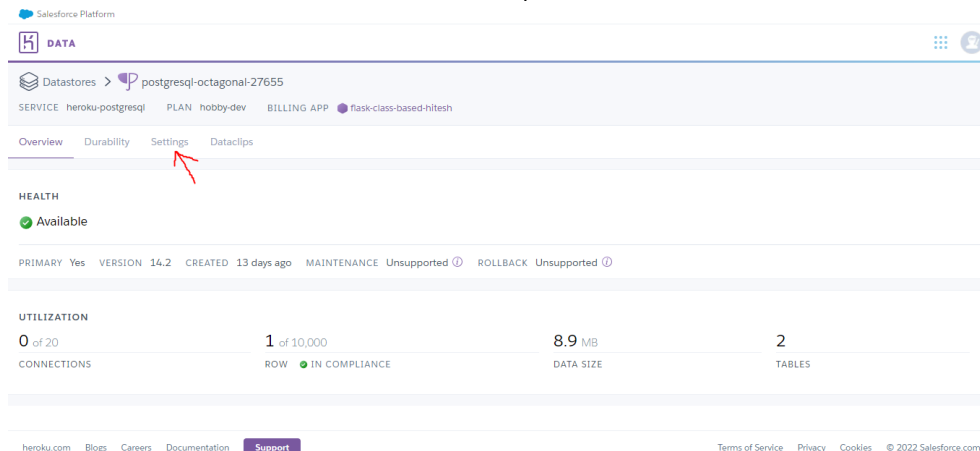
MODIFY THE DATABASE URI

The URL for the projects under app.py in a method based or __init__.py in class methods are the same URI that you receive from the "Heroku: Resource:YOUR_CUSTOM_postgres_app" with a small exception of adding

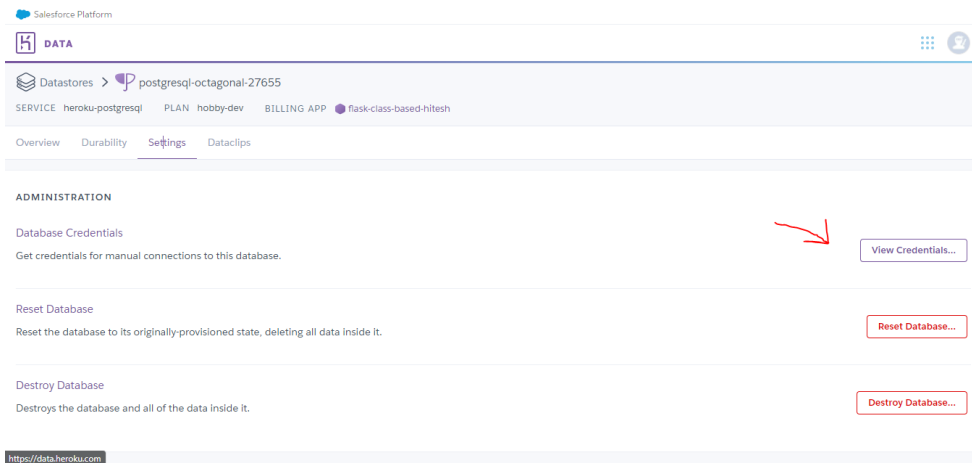
1. Get to the "Resources" section from your created app on Heroku.



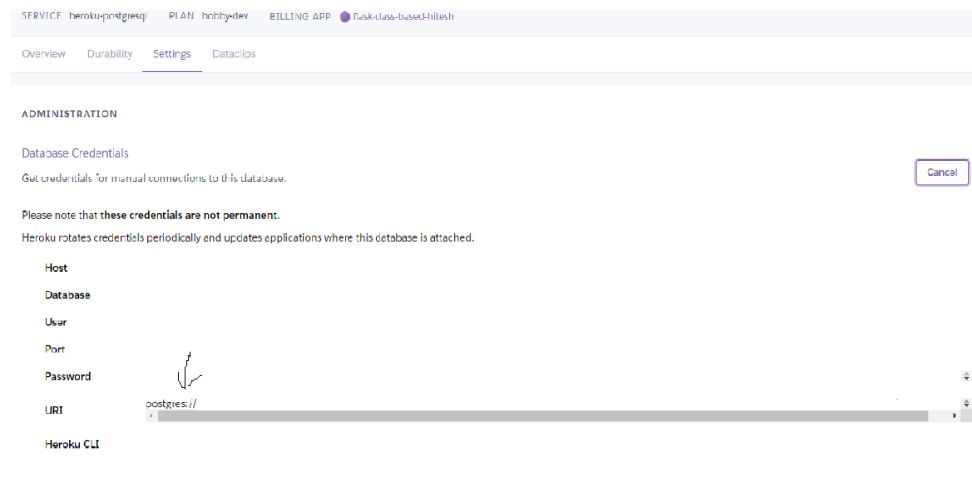
2. Once there click on the resource which would open a new tab.



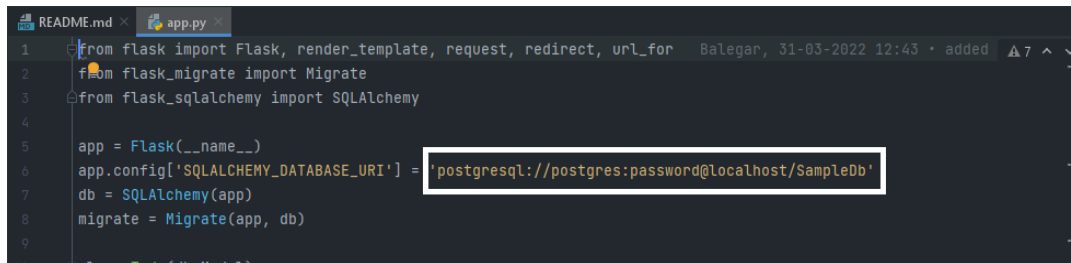
3. Click on settings



- Once you click on the “View credentials” button, you would be directed to a new tab that would have all the Database URI required by the app to connect to the database present on Heroku



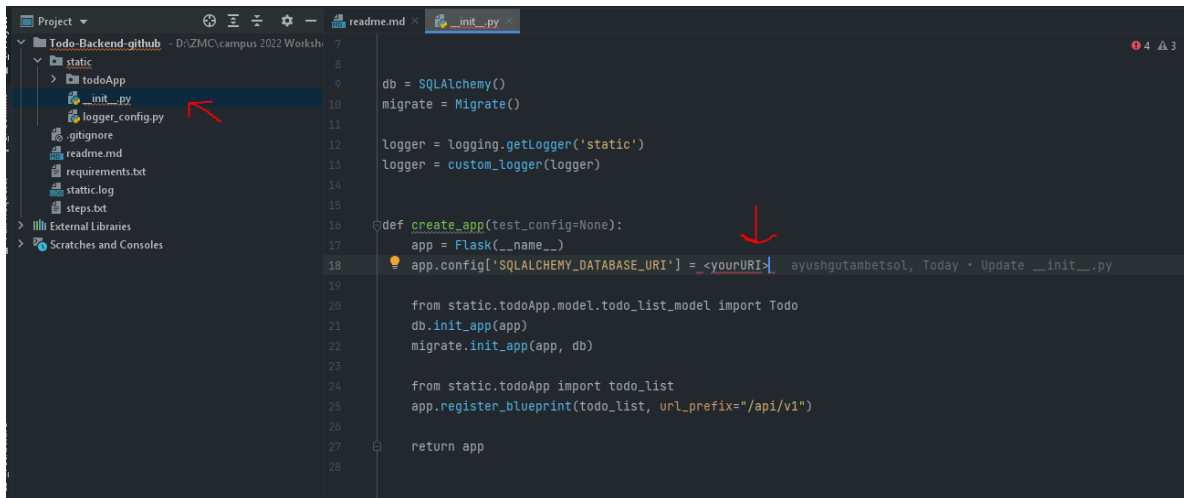
- Once you have the URI which would be in the format like “postgresql://__CREDENTIALS__”, you need to replace the once present your app.py in method-based views and “__init__.py” in class-based views.



```

1 from flask import Flask, render_template, request, redirect, url_for
2 from flask_migrate import Migrate
3 from flask_sqlalchemy import SQLAlchemy
4
5 app = Flask(__name__)
6 app.config['SQLALCHEMY_DATABASE_URI'] = 'postgresql://postgres:password@localhost/SampleDb'
7 db = SQLAlchemy(app)
8 migrate = Migrate(app, db)
9

```



```

7 db = SQLAlchemy()
8 migrate = Migrate()
9
10 logger = logging.getLogger('static')
11 logger = custom_logger(logger)
12
13 def create_app(test_config=None):
14     app = Flask(__name__)
15     app.config['SQLALCHEMY_DATABASE_URI'] = '<yourURI>'
16
17     from static.todoApp.model.todo_list_model import Todo
18     db.init_app(app)
19     migrate.init_app(app, db)
20
21     from static.todoApp import todo_list
22     app.register_blueprint(todo_list, url_prefix="/api/v1")
23
24     return app
25

```

6. Change the URI from

postgres://user:password@localhost/DbName

to

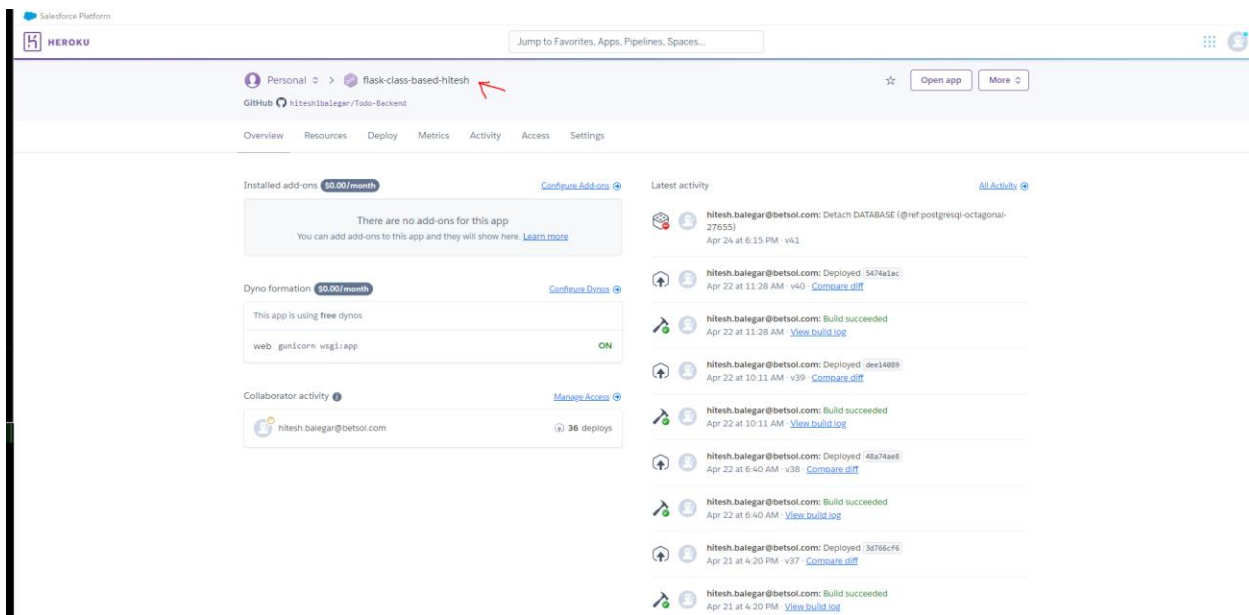
postgresql://user:password@localhost/DbName

DEPLOYING THE APP

Once you have the changes saved into the corresponding file. It's time for deployment, the procedure is given here would be the same for the class-based views or method-based-views.

1. Get your application name which was given when you created your app on Heroku

For Ex:



In the above image, we can see that the name of our app is *"flask-class-based-hitesh"*

We use that in the below step.

- To deploy our app on Heroku now, attach it to the Heroku remote repository that was created for our Heroku app created recently. Use the command –

```
heroku git:remote -a flask-class-based-hitesh
```

```
hitesh.balegar@BOS1-8NG-1377 MINGW64 /d/ZNC/campus 2022 Workshop/Github Repo Original LLC/ToDo-Backend (ToDo-Class-Logger)
$ heroku git:remote -a flask-class-based-hitesh
» Warning: heroku update available from 7.53.0 to 7.60.1.
set git remote heroku to https://git.heroku.com/flask-class-based-hitesh.git
(venvPycharm)
```

- Commit the changes made to the respective file for the class-based views or method-based-views using

```
git add .
git commit -m "YOUR_MESSAGE"
```
- Push your changes to the remote branch and trigger a build

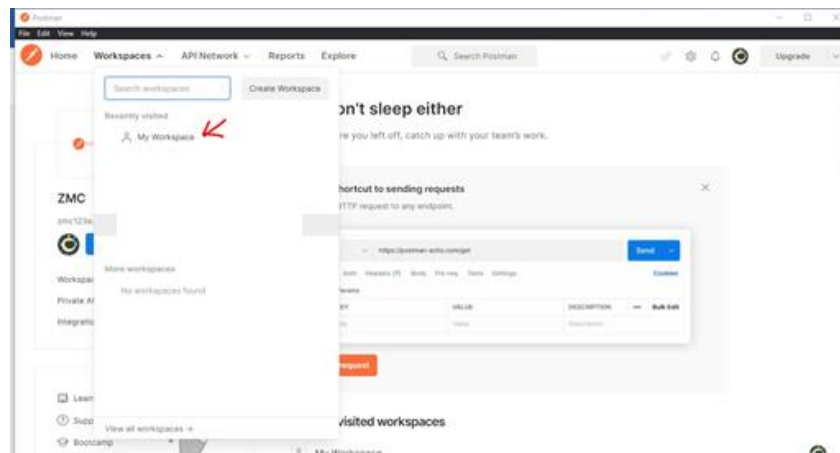
```
git push origin ToDo-Class-Logger
git push heroku ToDo-Class-Logger:main
```

```
Enumerating objects: 127, done.
Counting objects: 100% (127/127), done.
Delta compression using up to 8 threads
Compressing objects: 100% (84/84), done.
Writing objects: 100% (127/127), 31.78 KiB | 5.30 MiB/s, done.
Total 127 (delta 36), reused 85 (delta 20), pack-reused 0
remote: Compressing source files... done.
remote: Building source:
remote:
remote: ----> Building on the Heroku-20 stack
remote: ----> Using buildpack: heroku/python
remote: ----> Python app detected
remote: ----> No Python version was specified. Using the same version as the last build: python-3.10.4
remote: ----> To use a different version, see: https://devcenter.heroku.com/articles/python-runtimes
remote: ----> No change in requirements detected, installing from cache
remote: ----> Using cached install of python-3.10.4
remote: ----> Installing pip 22.0.4, setuptools 60.10.0 and wheel 0.37.1
remote: ----> Installing SQLite3
remote: ----> Installing requirements with pip
remote: ----> Discovering process types
remote: ----> Procfile declares types -> web
remote:
remote: ----> Compressing...
remote: ----> Done: 69.3M
remote: ----> Launching...
remote: ----> Released v32
remote: ----> https://flask-class-based-hitesh.herokuapp.com/ deployed to Heroku
remote:
remote: Verifying deploy... done.
To https://git.heroku.com/flask-class-based-hitesh.git
+ c2783c8...1c1dc4f Todo-Class-Logger -> main (forced update)
```

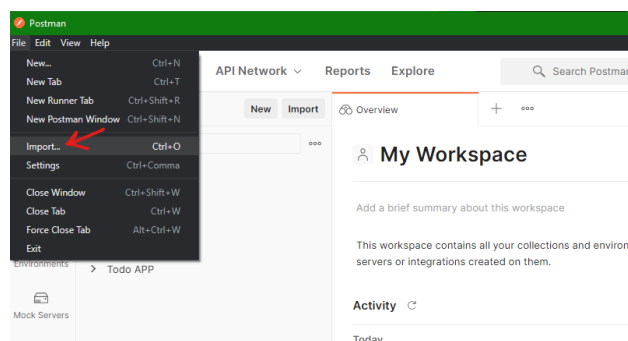
- Once this is done, you can see that a URL is given in the command output, in the above picture the URL generated in our case is <https://flask-class-based-hitesh.herokuapp.com/>, we need to copy and paste this URL in postman to hit and try out the various API endpoints created in the app.

CHECKING THE RESPONSE ON POSTMAN

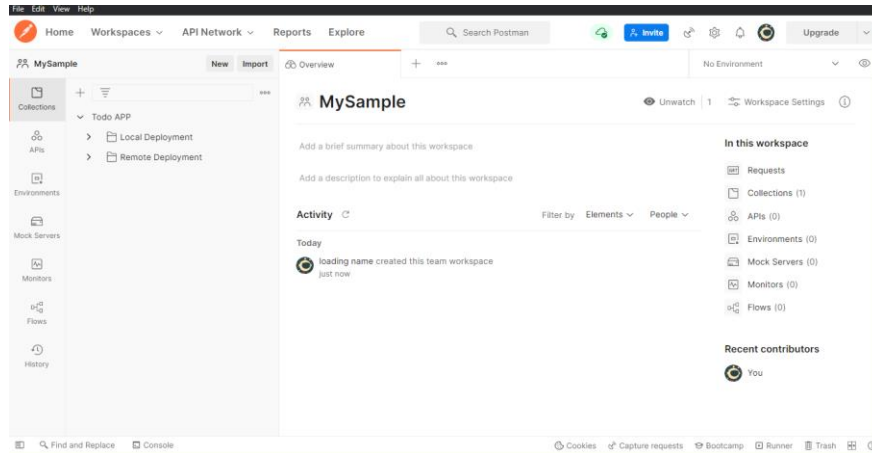
- Open your Postman and either go to workspace -> My Workspace or create a new workspace from the same option.



- Once selected click on file -> Import



3. You will get a window to upload files, click on the upload button and upload the '**Todo APP.postman_collection.json**', which can be found from your present working directory.
4. Once uploaded, you can probably see something like the figure below.



5. From here, you can open the Remote Deployment and click on the "Get All Tasks" collection to check out the get method for our App.
 - a. Make sure to replace your application's URL in the request's URL section in POSTMAN and **add /api/v1** at the end as given in the below picture.
 - b. Once done hit Send and see if the response is same as below.

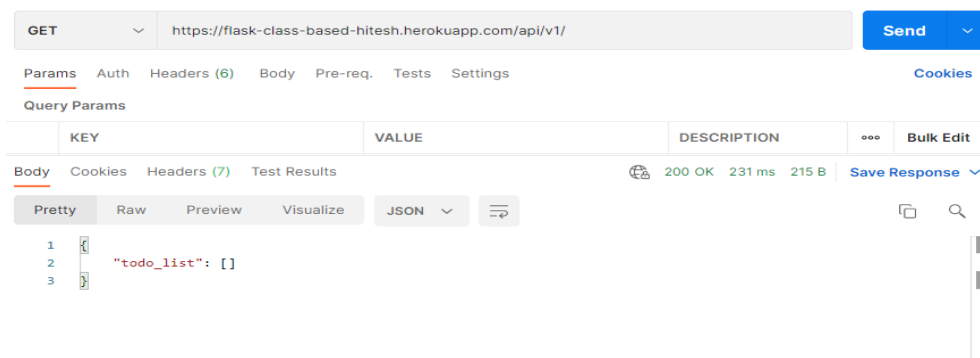
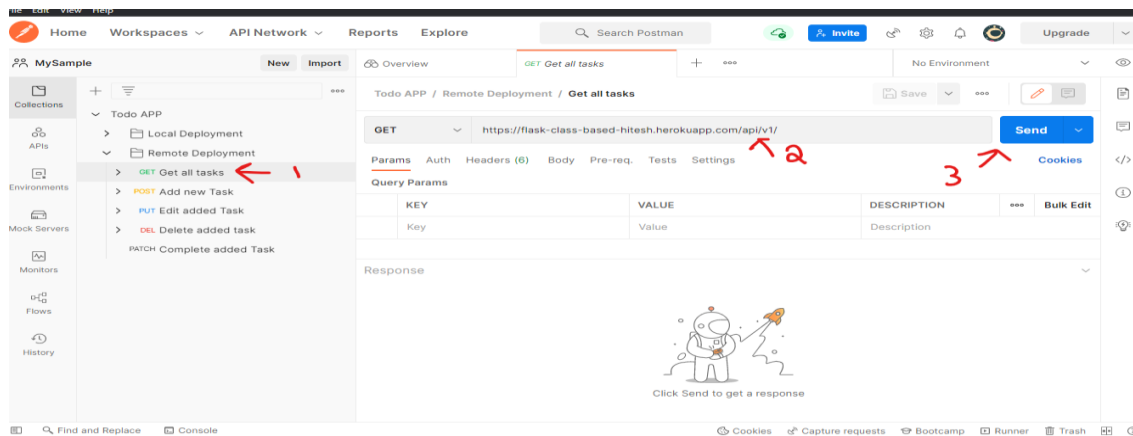


Figure 1 Get response with no content body

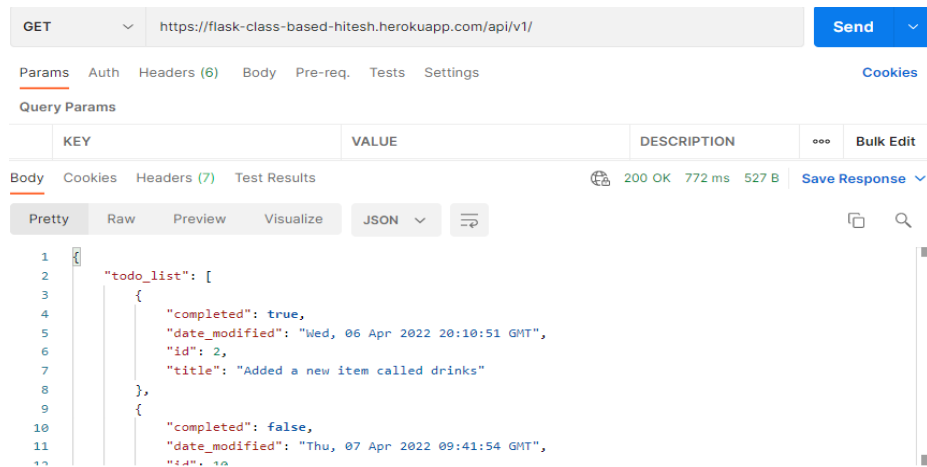


Figure 2 Get containing tasks already in the database

You can also explore the same examples under each request, which has already captured the sample output for the respective method. From here you can go out and explore the various methods implemented on our App which are GET, POST, PUT and DELETE

LOCAL DEPLOYMENT *OPTIONAL

This section covers the local deployment of the to-do app in one's system you can do it **if you have time and have completed the Frontend section including the exercises**. The deployment can be divided into the following steps.

- Installing the Virtual Environment package
- Creating and activating the virtual environment in case of Visual studio code
- Creating and activating the virtual environment in case of PyCharm
- Starting up the Flask process

INSTALLING THE VIRTUAL ENVIRONMENT PACKAGE

Use pip install virtualenv to install virtualenv module

```

PS D:\To-Do Flask\Todo-Backend> pip install virtualenv
Requirement already satisfied: virtualenv in c:\users\ayush.gautam\appdata\local\programs\python\python310\lib\site-packages (20.14.0)
Requirement already satisfied: six<2,>=1.9.0 in c:\users\ayush.gautam\appdata\local\programs\python\python310\lib\site-packages (from virtualenv) (1.16.0)
Requirement already satisfied: platformdirs<3,>=2 in c:\users\ayush.gautam\appdata\local\programs\python\python310\lib\site-packages (from virtualenv) (2.5.1)
Requirement already satisfied: filelock<4,>=3.2 in c:\users\ayush.gautam\appdata\local\programs\python\python310\lib\site-packages (from virtualenv) (3.6.0)
Requirement already satisfied: distlib<1,>=0.3.1 in c:\users\ayush.gautam\appdata\local\programs\python\python310\lib\site-packages (from virtualenv) (0.3.4)
PS D:\To-Do Flask\Todo-Backend>

```

Note: If the above command fails with an error output like the one below. Then you can continue to step 4 directly without activating the environment. Just make sure to use pip instead of pipenv

Ex: pipenv install -r requirements.txt ❌ pip install -r requirements.txt

```
$ pip install virtualenv
Collecting virtualenv
  Using cached virtualenv-20.14.1-py2.py3-none-any.whl (8.8 MB)
Requirement already satisfied: six<2,=>=1.9.0 in c:\python310\lib\site-packages (from virtualenv) (1.16.0)
Requirement already satisfied: platformdirs<3,=>=2 in c:\python310\lib\site-packages (from virtualenv) (2.5.2)
Requirement already satisfied: distlib<1,=>=0.3.1 in c:\python310\lib\site-packages (from virtualenv) (0.3.4)
Requirement already satisfied: filelock<4,=>=3.2 in c:\python310\lib\site-packages (from virtualenv) (3.6.0)
Installing collected packages: virtualenv
  WARNING: Failed to write executable - trying to use .delete logic
ERROR: Could not install packages due to an OSError: [WinError 2] The system can not find the file specified: 'C:\Python310\Scripts\virtualenv.exe' -> 'C:\Python310\Scripts\virtualenv.exe.delete'

```

Figure 1 shows one of the possible errors during the installation of virtualenv

CREATING AND ACTIVATING THE VIRTUAL ENVIRONMENT IN CASE OF VISUAL STUDIO CODE

In the case of visual studio code, we do not have a straightforward plugin to activate the virtual environment, hence we'll look at activating the environment using the terminal/command line.

Ps:

1. Make sure to navigate to the directory in which you have created your virtual environment.
2. The commands executed below are done using “*git bash*” application. Please make sure you are using the same else you might face certain errors specific to your command line tool.

1. Execute `virtualenv <yourVenvName>`

In our case the virtual environment name is “*venvPycharm*” in your case it is whatever you have defined while setting it up.

The screenshot shows the Visual Studio Code interface with the 'Get Started' sidebar on the left and a terminal window at the bottom. The terminal shows the execution of the `virtualenv venvPycharm` command, which successfully creates a virtual environment. The output includes details about the Python version (3.10.4) and the packages installed in the environment.

```

Hitesh.Balegar@BSSL-RNG-L377 MINGW64 /d:/ZMC/campus 2022 Workshop/campus Demo
$ virtualenv venvPycharm/
created virtual environment CPython3.10.4.final.0-64 in 4678ms
creator CPython3Windows(dest=D:\ZMC\campus 2022 Workshop\campus Demo\venvPycharm, clear=False, no_vcs_ignore=False, global=False)
seeders FromAppData(download=False, pip-bundle=True, setuptools-bundle=True, wheel-bundle=True, via=copy, app_data_dir=C:\Users\Hitesh.Balegar\AppData\Local\Virtualenvs)
added seed packages: Flask==2.1.0, Flask-Cors==3.0.10, Flask-Migrate==3.1.0, Flask-RESTful==0.3.9, Flask-SQLAlchemy==2.5.1, Flask-MIF==0.9.3, Jinja2==3.1.1, Mako==1.2.0, MarkupSafe==2.1.1, SQLAlchemy==1.4.32, WTForms==3.0.1, Werkzeug==2.1.0, alembic==1.7.7, aniso8601==9.0.1, click==8.1.0, colorama==0.4.4, greenlet==1.1.2, importlib_metadata==4.11.3, itsdangerous==2.1.2, pip==22.0.4, psycopp2_binary==2.9.3, pytz==2022.1, setuptools==62.1.0, six==1.16.0, wheel==0.37.1, zipp==3.7.0
activators BashActivator, BatchActivator, FishActivator, NushellActivator, PowerShellActivator, PythonActivator

```

2. Type `source venv/Scripts/activate` to activate a virtual environment

```

Hitesh.Balegar@BSL-BNG-L377 MINGW64 /d/ZMC/campus 2022 Workshop/campus Demo
$ source venvPycharm/Scripts/activate
(venvPycharm)
Hitesh.Balegar@BSL-BNG-L377 MINGW64 /d/ZMC/campus 2022 Workshop/campus Demo
$

```

Note: The args after source “venvPycharm/Scripts/activate”, has *venvPycharm* as the virtual env name, this can be different for your local setup depending on the name you give.

3. Then `<pip install -r requirements.txt>` to install all the dependencies

PS: Make sure you are under the same directory where the “requirements.txt”

```

(venvPycharm)
Hitesh.Balegar@BSL-BNG-L377 MINGW64 /d/ZMC/campus 2022 Workshop/campus Demo/ToDo-Backend (Todo-Class-Logger)
$ pip install -r requirements.txt
Requirement already satisfied: alembic==1.7.7 in d:\zmc\campus 2022 workshop\campus demo\venvpycharm\lib\site-packages (from -r requirements.txt (line 1)) (1.7.7)
Requirement already satisfied: aniso8601==9.0.1 in d:\zmc\campus 2022 workshop\campus demo\venvpycharm\lib\site-packages (from -r requirements.txt (line 2)) (9.0.1)
Requirement already satisfied: click==8.1.0 in d:\zmc\campus 2022 workshop\campus demo\venvpycharm\lib\site-packages (from -r requirements.txt (line 3)) (8.1.0)
Requirement already satisfied: colorama==0.4.4 in d:\zmc\campus 2022 workshop\campus demo\venvpycharm\lib\site-packages (from -r requirements.txt (line 4)) (0.4.4)
Requirement already satisfied: Flask==2.1.0 in d:\zmc\campus 2022 workshop\campus demo\venvpycharm\lib\site-packages (from -r requirements.txt (line 5)) (2.1.0)
Requirement already satisfied: Flask-Cors==3.0.10 in d:\zmc\campus 2022 workshop\campus demo\venvpycharm\lib\site-packages (from -r requirements.txt (line 6)) (3.0.10)
Requirement already satisfied: Flask-Migrate==3.1.0 in d:\zmc\campus 2022 workshop\campus demo\venvpycharm\lib\site-packages (from -r requirements.txt (line 7)) (3.1.0)

```

In the above image, the output of the command is something like “Requirements already satisfied”, this is because some of packages are already installed under the environment.

Note: For issues related to python setup not being in your **PATH** under your system environment variable you can click [this document](#).

STARTING THE FLASK PROCESS ON COMMAND LINE

The steps to start the flask process is as follows.

1. You have to set up the app level using the command “*export*” in “*git bash*” this is different for CMD and PS. You can check the corresponding command according for it.

```

(venvPycharm)
Hitesh.Balegar@BSL-BNG-L377 MINGW64 /d/ZMC/campus 2022 Workshop/campus Demo/ToDo-Backend (Todo-Class-Logger)
$ export FLASK_APP=wsgi.py:app

```

Ps: Make sure you are under the root directory of the app, in above image the root directory is “*ToDo-Backend*” and the virtual env is activated properly with all the libraries installed.

2. Next, start the flask process by executing “*flask run*”. Once this is done you can start trying out all the requests in the app.

```

(venvPycharm)
Hitesh.Balegar@BSL-BNG-L377 MINGW64 /d/ZMC/campus 2022 Workshop/campus Demo/ToDo-Backend (Todo-Class-Logger)
$ flask run
* Serving Flask app "wsgi.py:app" (lazy loading)
* Environment: production
WARNING: This is a development server. Do not use it in a production deployment.
Use a production WSGI server instead.
* Debug mode: off
* Running on http://127.0.0.1:5000 (Press CTRL+C to quit)

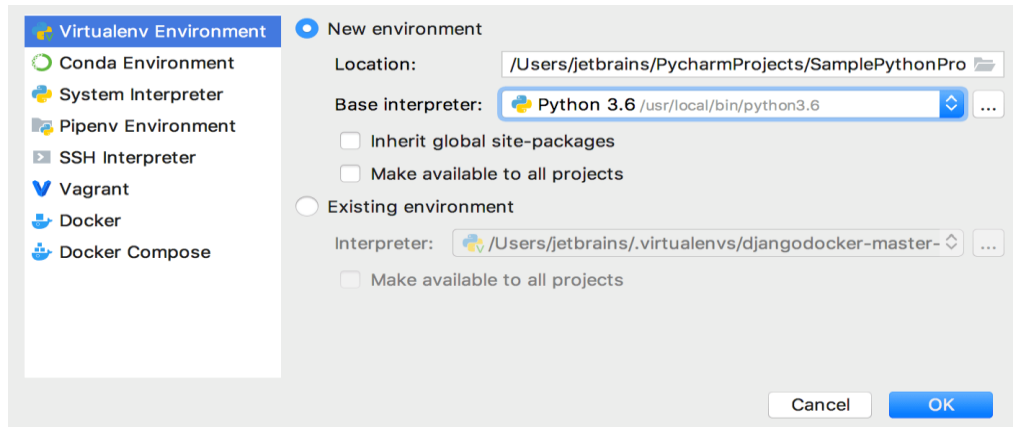
```

CREATING AND ACTIVATING THE VIRTUAL ENVIRONMENT IN CASE OF PYCHARM

Use one of the following:

- Click the Python Interpreter selector and choose Add Interpreter.
- Press Ctrl+Alt+S to open the project Settings/Preferences and go to Project <project name> | Python Interpreter. Click and select Add.

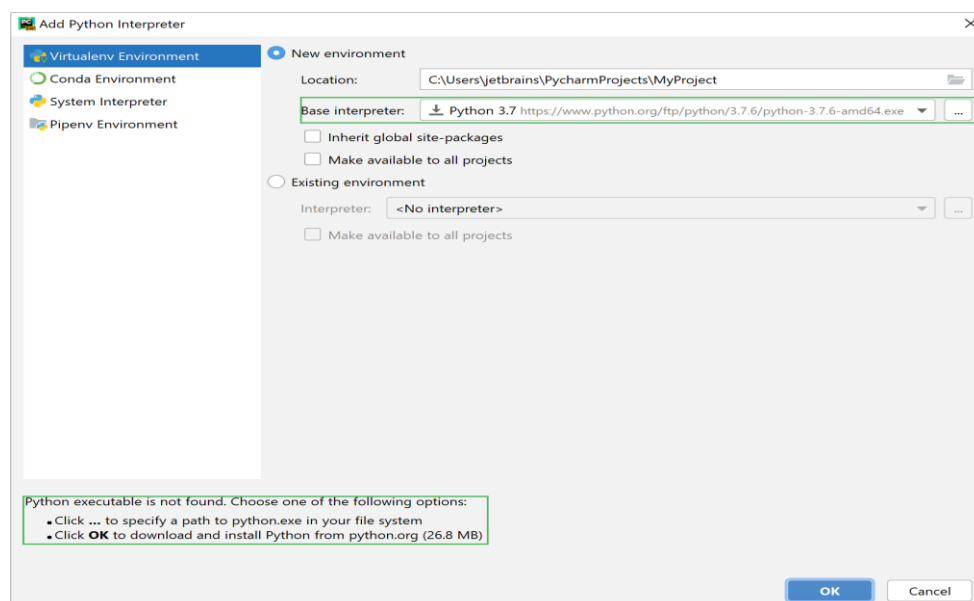
In the left-hand pane of the Add Python Interpreter dialog, select Virtualenv Environment. The following actions depend on whether the virtual environment existed before.



If New Virtualenv is selected:

- Specify the location of the new virtual environment in the text field, or click and find location in your file system. Note that the directory where the new virtual environment should be located, must be empty!
- Choose the base interpreter from the list, or click and find a Python executable in your file system.

If PyCharm detects no Python on your machine, it provides two options: to download the latest Python versions from python.org or to specify a path to the Python executable (in case of non-standard installation).



- Select the Inherit global site-packages checkbox if you want that all packages installed in the global Python on your machine to be added to the virtual environment you're going to create. This checkbox corresponds to the `--system-site-packages` option of the `virtualenv` tool.

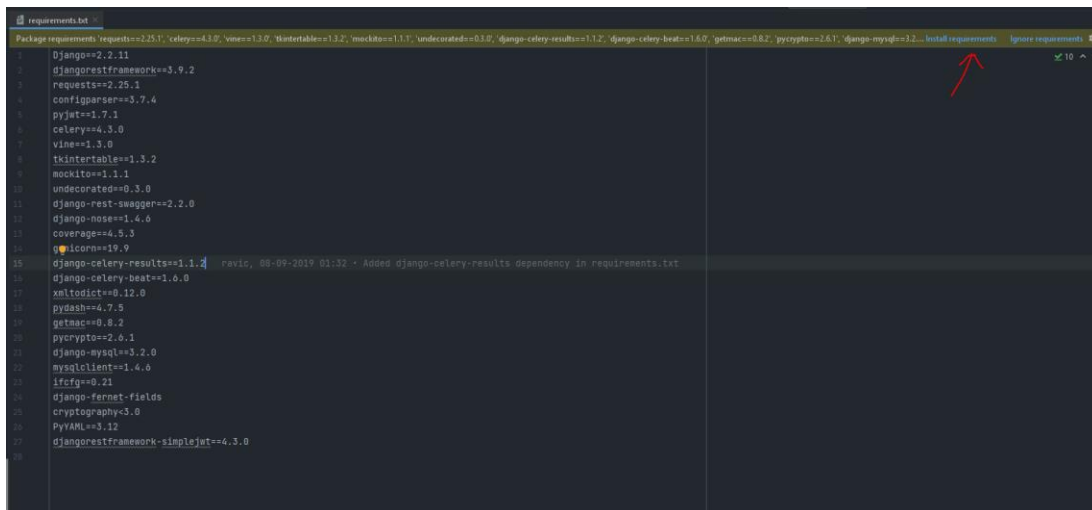
If you select any of the existing virtual environments from the Interpreter list, it will be reused for the current project.

Finally Click OK to complete the task.

Installing all the packages:

You can start installing the packages in interpreter using the two ways.

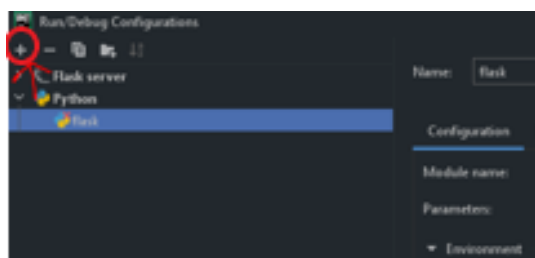
1. Once the interpreter is setup you can start add the packages using the interpreter settings->select interpreter->click on “+” and search and add the corresponding package.
2. You can open “*requirements.txt*” on PyCharm and you can see that PyCharm would start “indexing” the file and would prompt you to install the required packages under the env if not present as shown in the below picture.



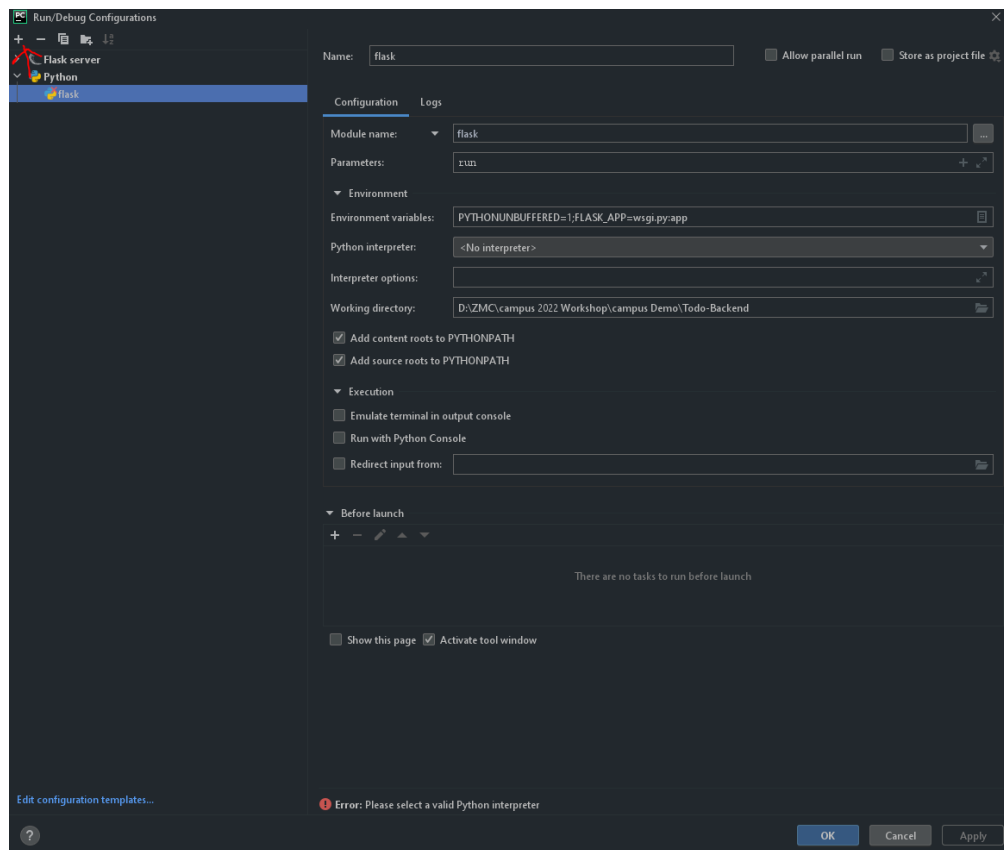
STARTING THE FLASK PROCESS ON PYCHARM

The steps to start the flask process as follows.

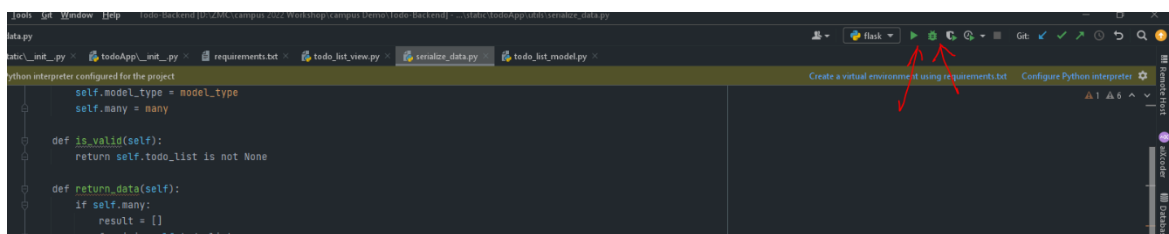
1. Add python configuration for flask.
 - a. Select “Add configuration” on PyCharm and select the “+” icon to select a “python” configuration.



- b. Next in the configuration section make sure to have the options under the configuration filled respectively as given in the image below:
 - i. Python_interpreter: Make sure your interpreter is selected.
 - ii. Environment_variables : “PYTHONUNBUFFERED=1;FLASK_APP=wsgi.py:app”
 - iii. Click on the dropdown and select module as the default parameter. PyCharm would have Script Path as default. So make sure to change it to Module_name
 - iv. Once done you can add the value “flask” for the Module_name field.
 - v. Parameters: “run”
 - vi. Working directory should be the root directory. In our case, it is as shown in the image below. In your case, it should be the root directory where the backend is cloned.



2. Once the configuration is specified, you can click on “Apply” and “Ok”
3. You can click on run or debug which effectively starts the process.



PS: For more details, you can look up the documentation [here](#).

Sam has now hosted the backend on Heroku and tested the APIs using postman. But he doesn't have a UI to interact with and create, list the To Do Items. Hence, he uses ReactJS to build his frontend for the To Do App and host it also on the Heroku so that any user across the internet can access his To Do App.

PREREQUISITES

- NodeJS and NPM (Node - 14.16.1 and npm - 6.14.12)
- Visual studio Code
- Heroku CLI

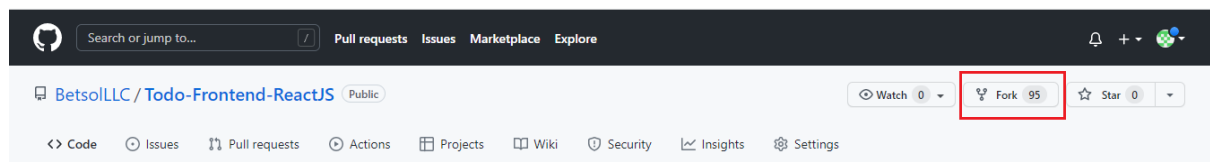
TASKS TO BE PERFORMED

1. Fork & Clone.
2. Update the backend endpoint URL.
3. Deploy frontend on Heroku.

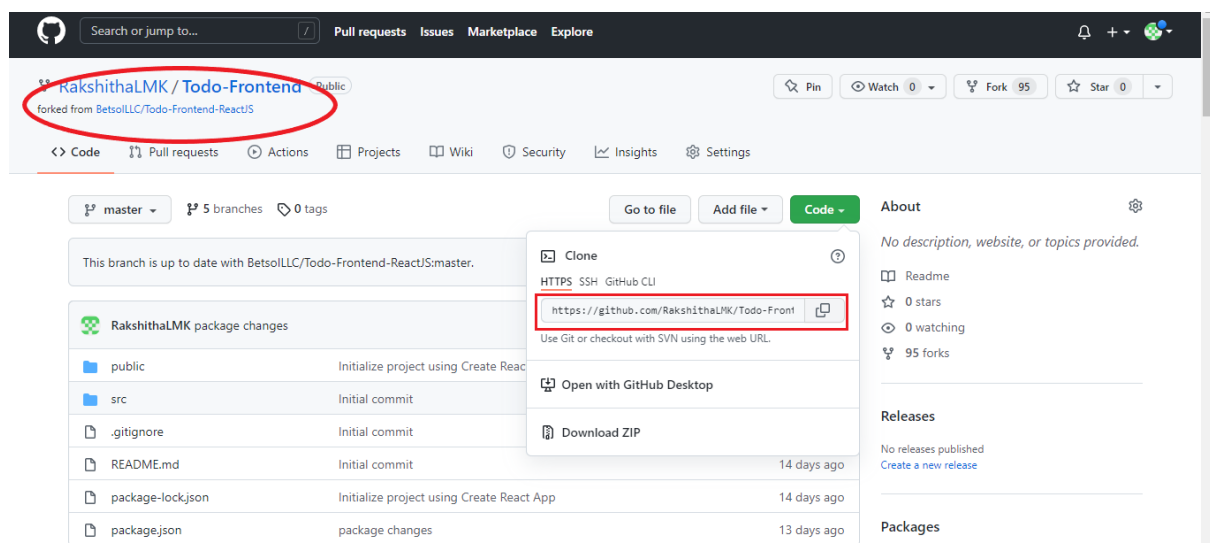
FORK & CLONE

There is already a project template built and ready to use to start with the To Do App.

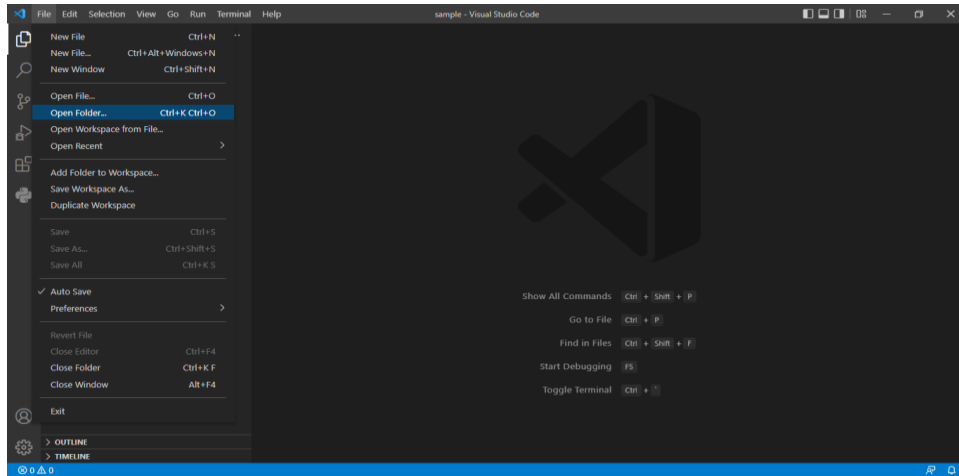
1. Fork the repository from the given URL from GitHub - <https://github.com/BetsolLLC/ToDo-Frontend-ReactJS.git>



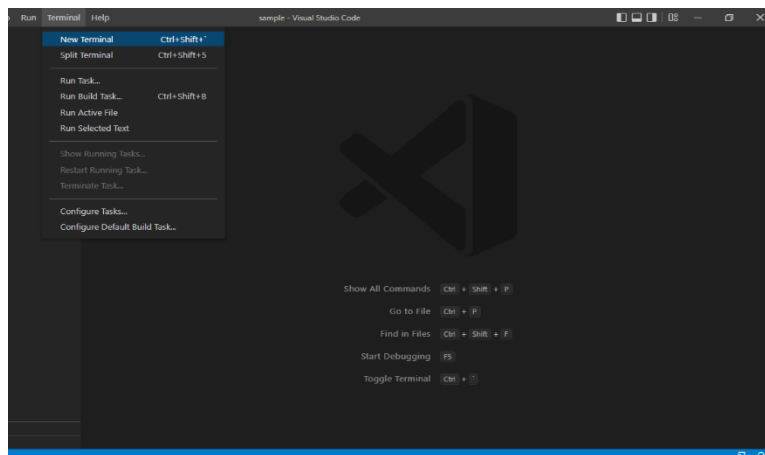
2. Then create a new folder in your system (preferably in D drive) and go into the folder.
3. Then **clone the code from the forked repository under your name**. (NOT the original repository that you forked previously).



4. Open VS Code.
5. Go to File -> Open Folder option and select the “todo-frontend” folder.



6. Then open a terminal here and **checkout/switch** to the branch “**app-1st**” using command
git checkout app-1st



Note: All commands going forward will be executed in VS Code Terminal itself.

UPDATE THE BACKEND ENDPOINT URL

1. Open the file: src -> App.js.
2. Change the value of the variable “url” to the URL of the backend deployed on Heroku.

```

JS App.js x
src > JS App.js > ...
1 import React from "react";
2 import Todo from "../components/Todo";
3 import { useRef, useState, useEffect } from "react";
4
5 export const url = "<Backend Heroku deployment URL>";
6
7 function App() {
  
```


3. Save the changes.
4. Then execute the following commands to push the new changes to the repository.

```
git add .  
git commit -m "changes made to the endpoint URL"  
git push origin app-lst
```

DEPLOY FRONTEND ON HEROKU

Once the above steps are done, we can proceed with deploying frontend on Heroku. For this, follow the steps below:

Assuming that Heroku CLI installation is done already, verify the installation using the command -

```
heroku version
```

```
PS D:\Rakshitha Lakshmi\CAMPUS WORKSHOP\todo\frontend\todo-frontend> heroku --version  
» Warning: heroku update available from 7.53.0 to 7.60.1.  
heroku/7.53.0 win32-x64 node-v12.21.0  
PS D:\Rakshitha Lakshmi\CAMPUS WORKSHOP\todo\frontend\todo-frontend> |
```

Note: If Heroku CLI is not installed, please follow the steps to install it mentioned in the 'backend development' -> 'remote deployment' section.

1. Login to your Heroku account via CLI using the command –

```
heroku login -i
```

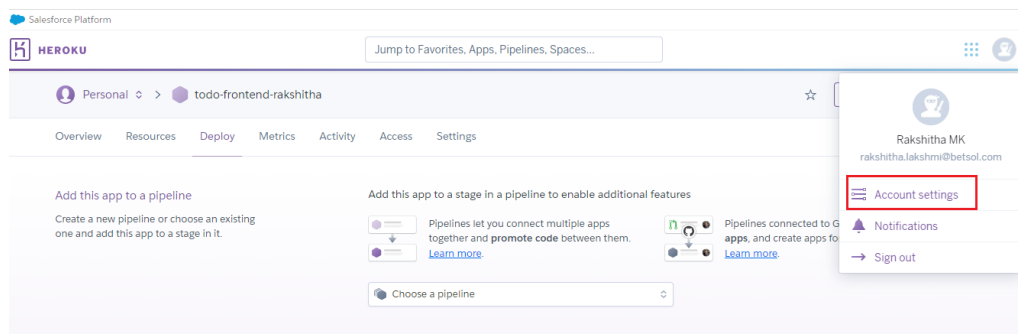
Here it prompts you to enter the following credentials –

- a. Email: <your Heroku login mail id">
- b. Password: <Your Heroku account authorization token for Heroku CLI (mentioned in detail below)>

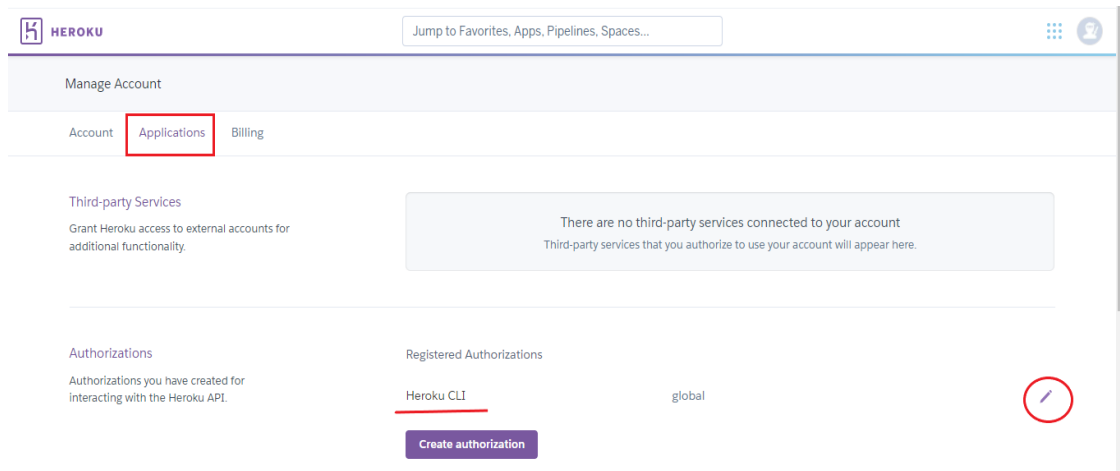
Note: Please use 'right click' to paste the password once u get it.

To get the authorization token,

- a. Go to Heroku Dashboard -> account settings



- b. Navigate to Applications tab and click on the 'edit' icon of the Heroku CLI Authorization.



- c. Copy the **TOKEN** given there and that **would be the Password for login** via the command "heroku login -i" mentioned previously.
- d. Once done, it will show our login like below:

```
PS D:\Rakshitha Lakshmi\CAMPUS WORKSHOP\todo\frontend\todo-frontend> heroku login -i
» Warning: heroku update available from 7.53.0 to 7.60.1.
heroku: Enter your login credentials
Email [rakshitha.lakshmi@betsol.com]:
Password: *****
Logged in as rakshitha.lakshmi@betsol.com
```

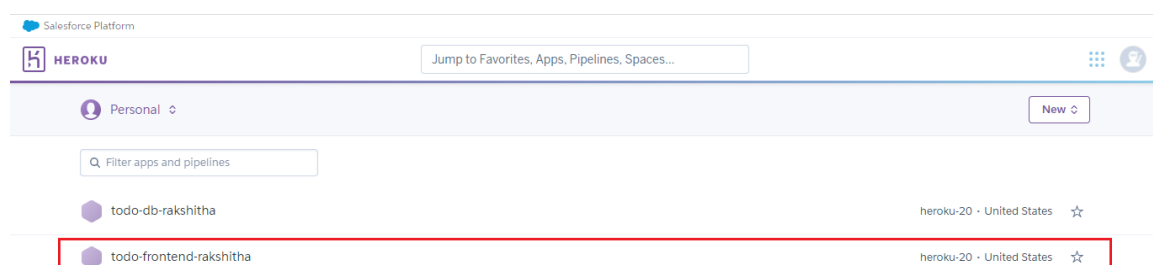
2. Create a Heroku App using the command below: Here, we are using the build-pack suitable for ReactJS i.e., mars/create-react-app.

Note: The app name must start with a letter, end with a letter or digit and can only contain lowercase letters, digits, and dashes.

```
heroku create $APP_NAME --buildpack mars/create-react-app
```

```
PS D:\Rakshitha Lakshmi\CAMPUS WORKSHOP\todo\frontend\todo-frontend> heroku create "todo-frontend-rakshitha" --buildpack mars/create-react-app
» Warning: heroku update available from 7.53.0 to 7.60.1.
Creating todo-frontend-rakshitha... done
Setting buildpack to mars/create-react-app... done
https://todo-frontend-rakshitha.herokuapp.com/ | https://git.heroku.com/todo-frontend-rakshitha.git
```

3. Once done, we can see the app under our personal apps section on Heroku. You can navigate to the URL <https://dashboard.heroku.com/apps> to check it.



- To deploy our app on Heroku now, attach it to the Heroku remote repository that was created for our recently created Heroku app. Use the command –

```
heroku git:remote -a $APP_NAME
```

```
PS D:\Rakshitha Lakshmi\CAMPUS WORKSHOP\todo\frontend\todo-frontend> heroku git:remote -a todo-frontend-rakshitha
» Warning: heroku update available from 7.53.0 to 7.60.1.
set git remote heroku to https://git.heroku.com/todo-frontend-rakshitha.git
```

- Then do a git push to the Heroku remote repository and the necessary branch i.e, **app-lst** using the command – Here, we deploy code to Heroku from a non-main branch i.e, app-lst.

```
git push heroku app-lst:main
```

Note: Do **not** to push the **package.json.lock** file to the repository. This is because the build might fail due to inconsistencies in the lock files when it tries to download the dependencies.

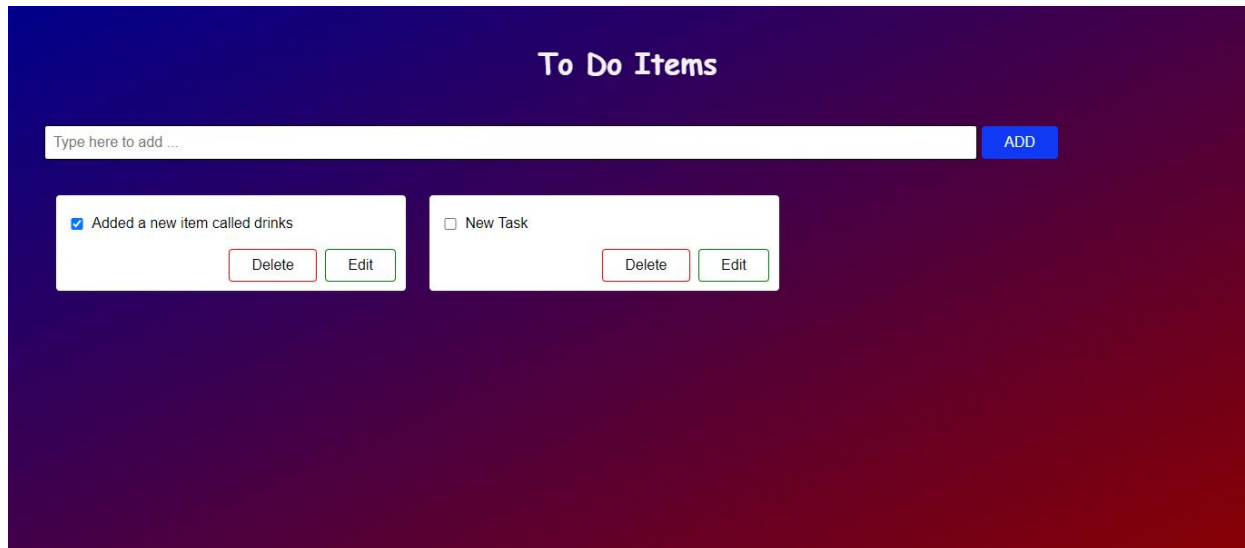
- This deploys the App as well. Hence, we should see the output of the command as below:

```
remote: -----> Compressing...
remote:      Done: 77.7M
remote: -----> Launching...
remote:      Released v3
remote:      https://todo-frontend-rakshitha.herokuapp.com/ deployed to Heroku
remote:
remote: Verifying deploy... done.
Everything up-to-date
```

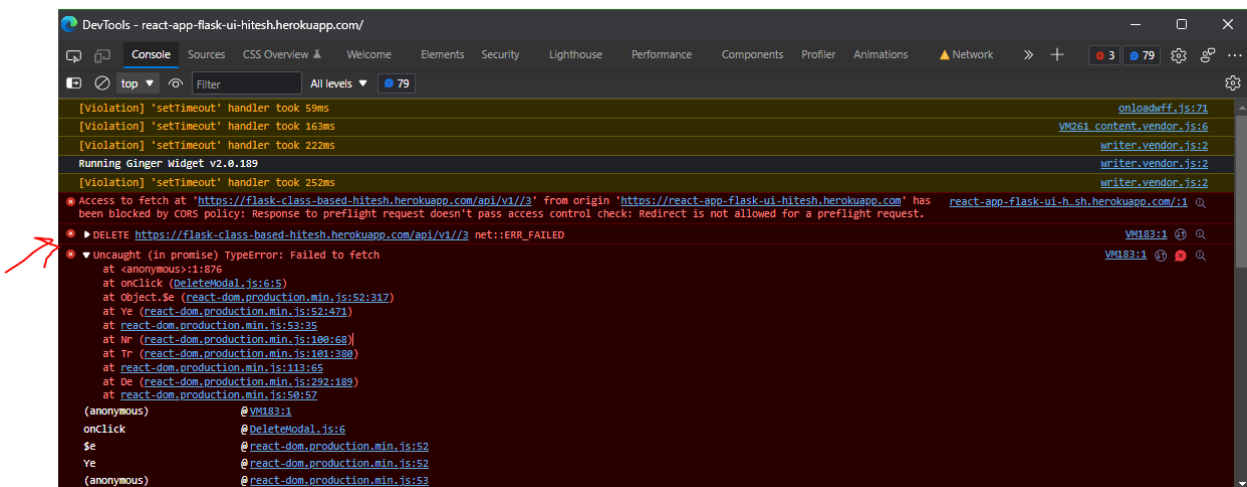
- Go to the Heroku dashboard -> your frontend App and then click on “Open App”

The screenshot shows the Heroku dashboard for the application 'todo-frontend-rakshitha'. The app name is highlighted with a red box. The 'Open app' button is circled in red. The dashboard includes sections for 'Add this app to a pipeline', 'Add this app to a stage in a pipeline to enable additional features', and 'Deployment method' with options for Heroku Git, GitHub, and Container Registry.

It will navigate to the respective URL of your Frontend App where you will be able to see the TO DO APP!



Note: While you are on the react app tab, press F12 or right click->inspect and go to the console tab. Check if there is a CORS issue as mentioned in the image below.



In case there are CORS issues for the API requests, start the chrome session without web security temporarily using any one of the following methods:

Method 1: Open 'Run' tab (press window key + R), type in this command and hit ENTER-

```
chrome.exe --user-data-dir="C:\tmpChromeSession" --disable-web-security
```

OR

Method 2: Open Command prompt and execute this command -

```
C:\Program Files\Google\Chrome\Application\chrome.exe" --user-data-dir="C:\tmpChromeSession" --disable-web-security
```

Then, navigate to URL where your frontend App is hosted. Explore the existing functionalities such as Adding, deleting, and listing the "To Do" items.

EXERCISES:

Sam can now add, view & delete a To Do item, but he wants to be able to edit the To Do Item title as well as be able to mark an item as complete.

EXERCISE 1: MARKING A TO DO ITEM AS COMPLETE

If Sam needs to mark a task as “complete” then he needs to create a *PATCH* method on the backend, which the Frontend can send a request to the API to do the same.

HINTS:

BACKEND - You can refer the Delete API method which is already implemented, for this task we need to create a method to update the “completed” details for each task.

FRONTEND - For complete item, the functionality is implemented from frontend ReactJS code (refer file src -> components -> **Todo.js**), just that a **backend API should be brought up** to handle it properly.

EXERCISE 2: EDITING A TO DO ITEM

If Sam needs to edit a task, then he needs to create the corresponding component logic on the UI, which would send a request to the backend API to update the task.

HINTS:

BACKEND – The API is already implemented.

FRONTEND - Refer to the implementation of delete item for trying out edit item.