**WEB DEVELOPMENT WITH PYTHON**

**Prerequisites:**

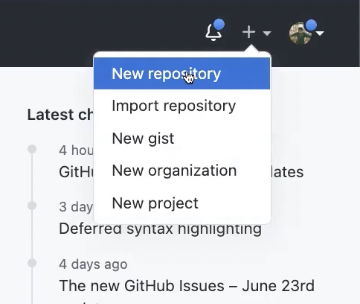
- HTML tutorial: https://htmldog.com/guides/ ,https://www.w3schools.com/html/

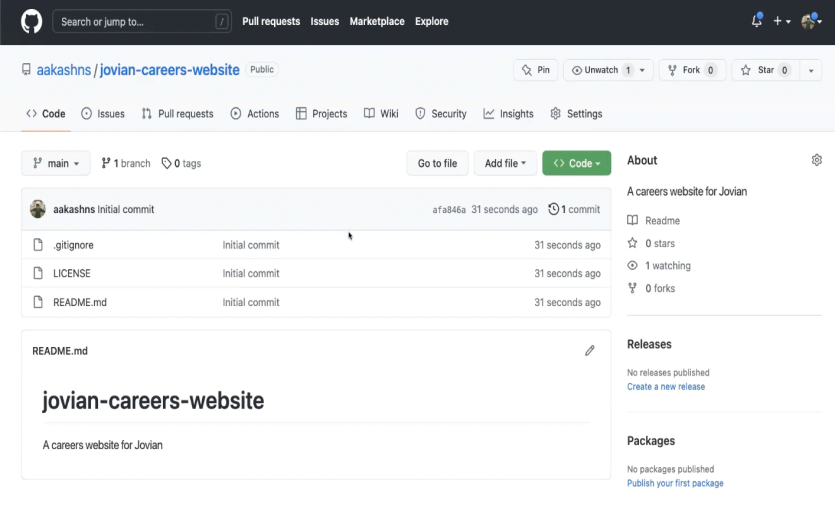
- CSS tutorial: https://htmldog.com/guides/css/ ,https://www.w3schools.com/css/

- Basic HTML, CSS templates : https://www.squarespace.com/blog/what-is-a-website-template

**Step 1** - **Project Setup & Flask Basics**

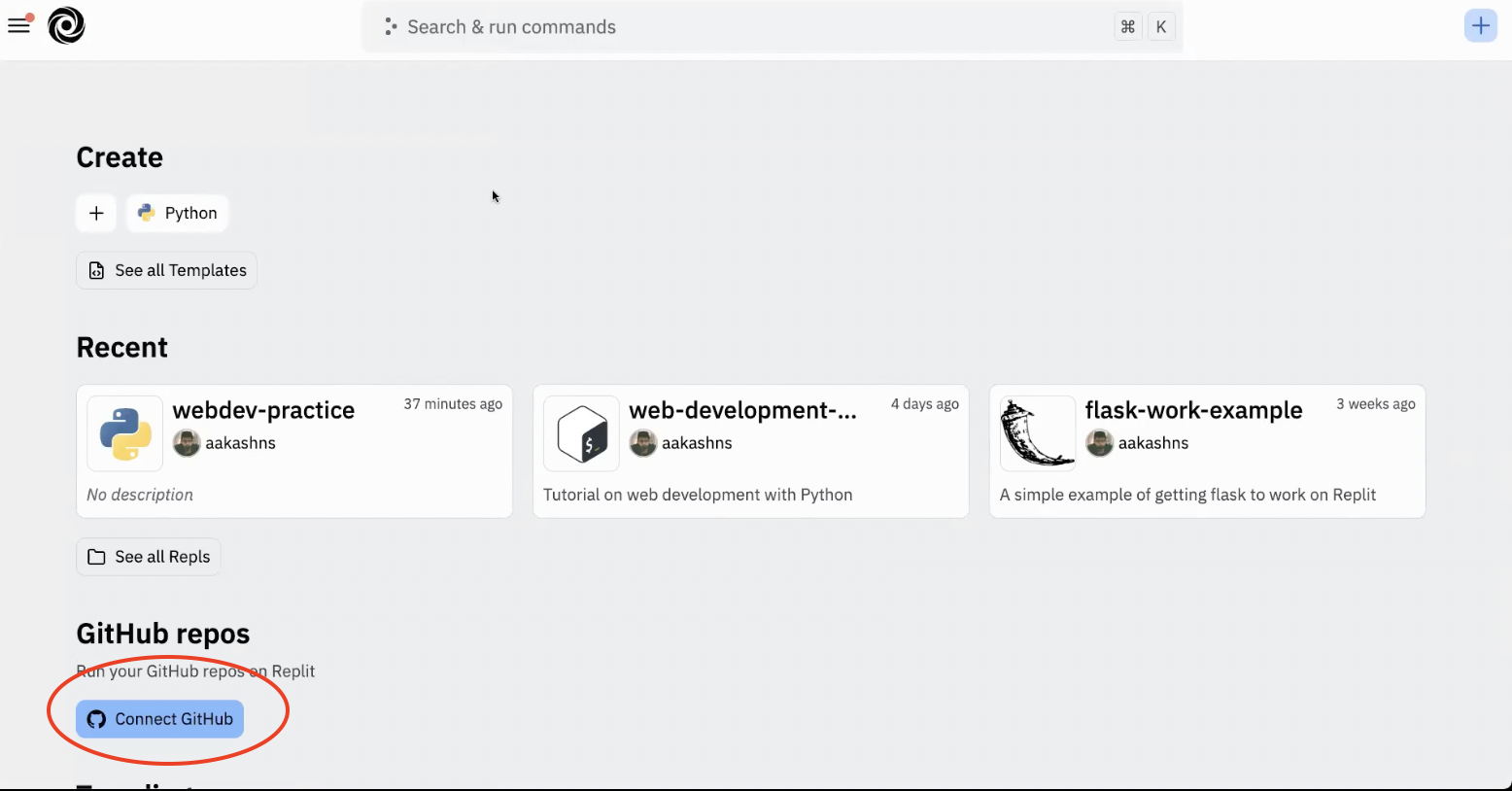
**Create a project on GitHub :-**Create a github repository,Then shift to a free resource for developing i.e replit





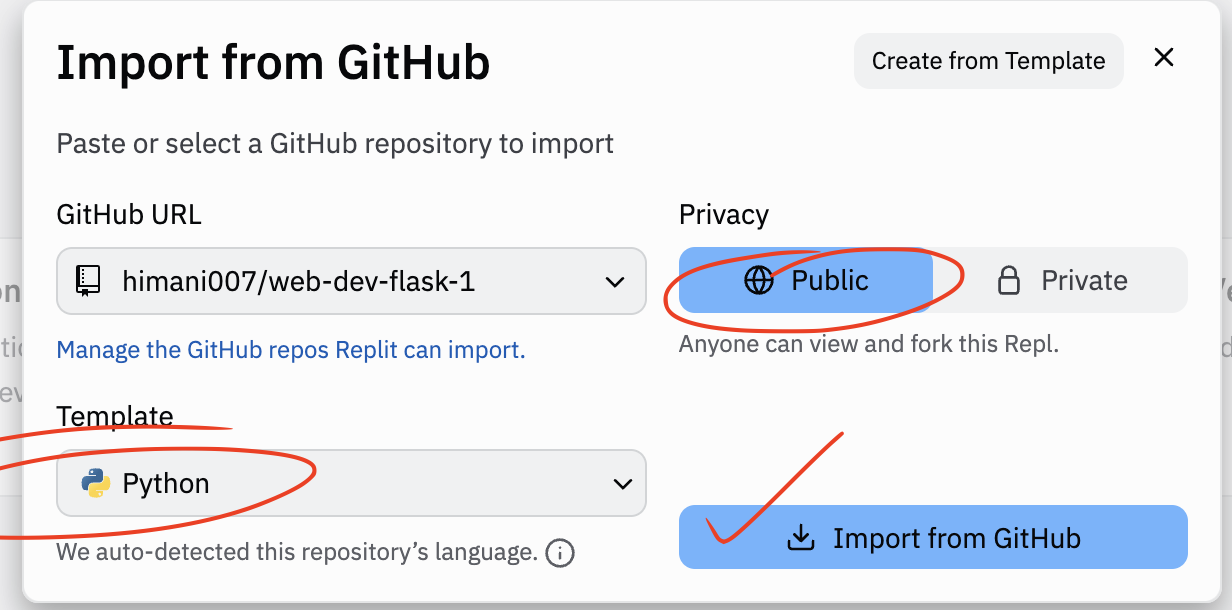
**Open up the Project on Replit:**  Replit allows us to integrate our code on GitHub ,i.e we can make edits on the cloud and the code will directly be synced on the cloud.

* To get started, open https://replit.com/ and SIGN UP to create your replit account .
* Once you've created your account, you will be able to view your replit dashboard. You can directly create new projects from here but ideally you should you github to keep a track of your projects. You can do this by connecting to GitHub:

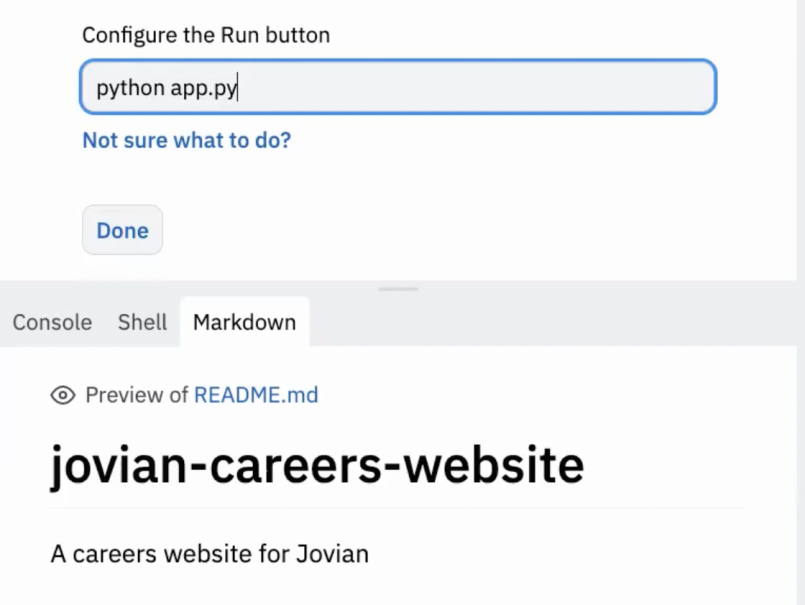


* After this step, you will be able to see a Github Repos section on your dashboard.
* You can select your repo and import it:

Which is now going to show all the repo files on replit.



* Now that you have your project open, you will be asked to configure a run command which will be executed every time we click on the run button.



* Head to the files tab and add a new file (example`app.py`.)

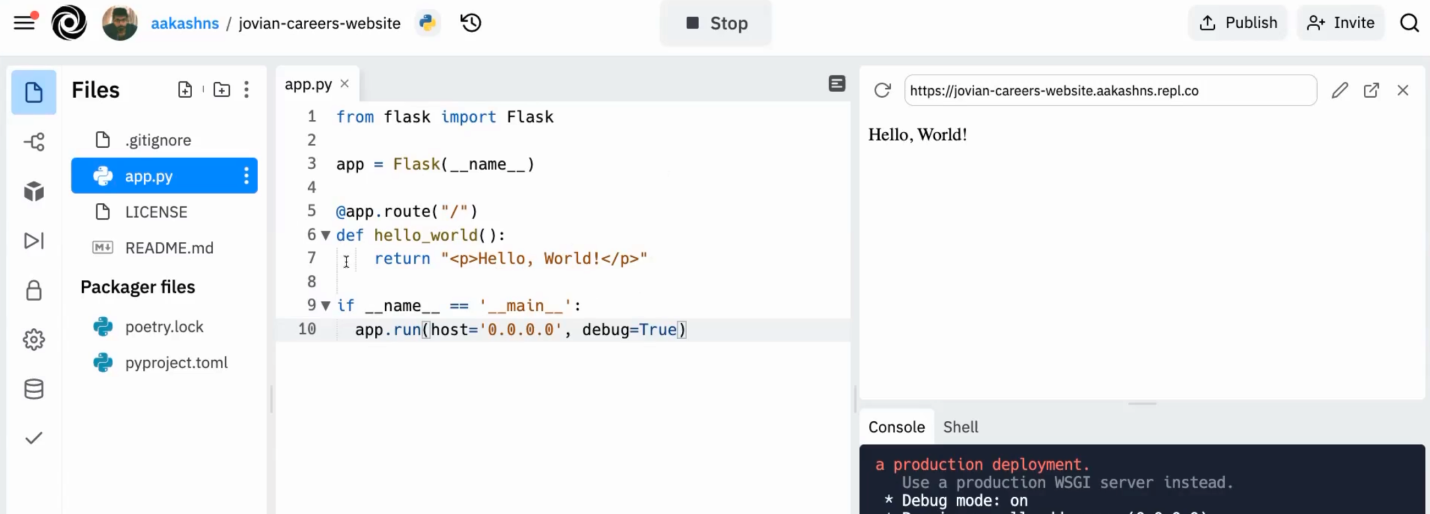
**Running a Flask Web Server :-**

For Building web application in python we use a framework called flask.

**Flask:** Flask is one of the most popular web frameworks in python. It is lightweight and provides useful tools that make creation of web applications in python easier.

Installation: If you're using flask locally, you can install it according to your operating system using this tutorial: <https://phoenixnap.com/kb/install-flask>

Because we are already using a cloud enviroment, we can directly install flask using the following command on our shell: ’pip install Flask’

**Creating a Simple Flask Application :-** We begin with importing the Flask class from the flask module. ‘from flask import Flask ‘**,**Even though these are both called flask, inside the **module flask** there is a class called **Flask** who's instance will be our WSGI application (Web Server Gateway Interface). **WSGI:** A Web Server Gateway Interface describes how a web server communicates with web applications🞟**NOTE**:Replit is not great with production workload so we should not be sending lots of traffic to this website.

**Pushing Changes back to Github**

To save our progress on githiub, we can head to the \*Version Control(GIT)\* tab. We can review the changes and then we can simply commit our changes and push them.

**References**

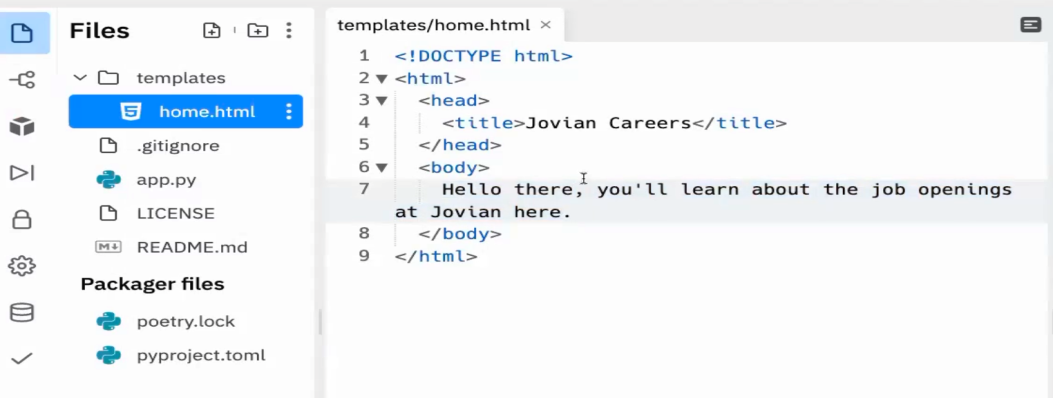
* GitHub tutorial: <https://docs.github.com/en/get-started/quickstart/hello-world>
* Replit tutorial: <https://docs.replit.com/tutorials/overview>
* Flask tutorial: <https://flask.palletsprojects.com/en/2.1.x/quickstart/>

**Step 2 - Web Pages with HTML & CSS**

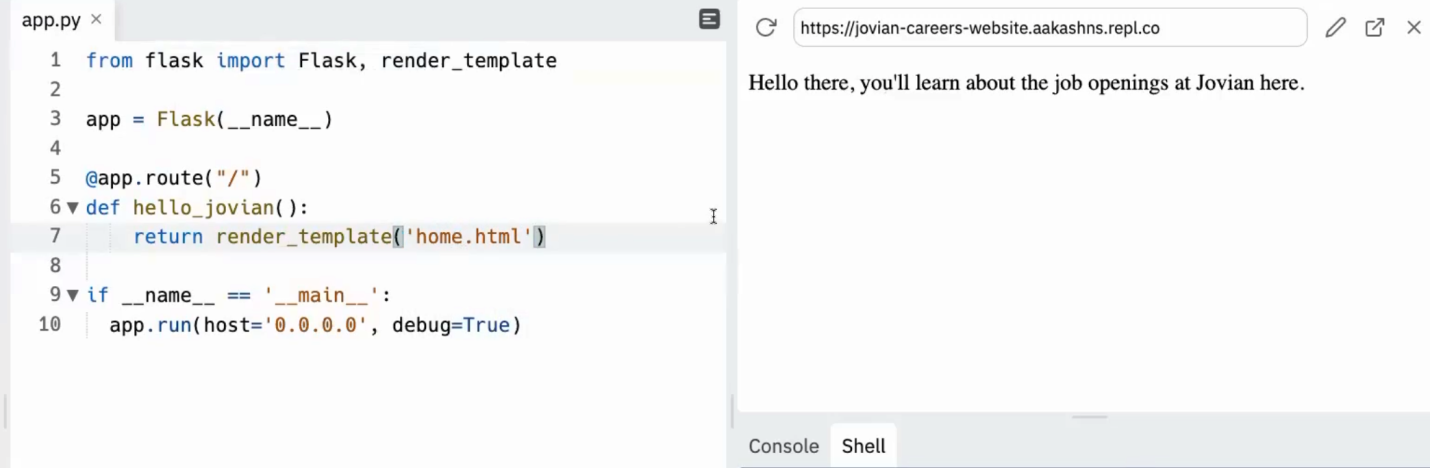
**Rendering Templates and Static Assets :-**Now that we have created a basic flask application it is time to embellish it with the help of HTML and CSS.

- We will head over to the files tab and create a folder called `templates`.

- Within our templates folder, we initialize our html file - `home.html` which contains the information and layout of what we want to show on our route page.

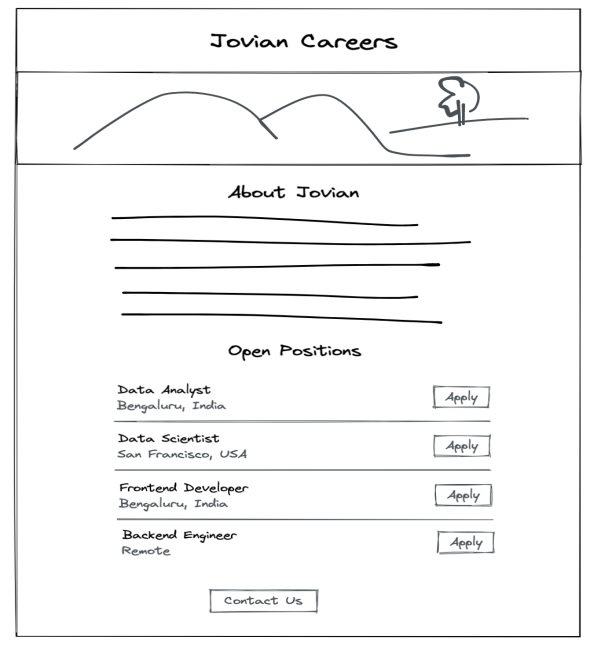


Now that our basic html template is ready, we need to render this template into our main application app.py.For this we use the render\_template function which we need to import from flask.And instead of returning Hello World, we return render\_template('home.html).

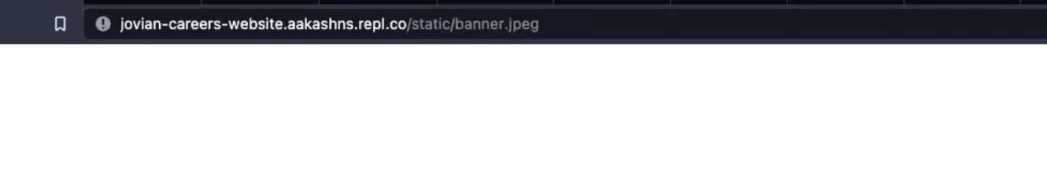


**Creating Wireframes:**

To have a gist of what we are creating we will create a rough outline of our website using a whiteboard or a notebook. Use virtual whiteboard https://excalidraw.com/ to create our wireframes.



**Adding an Image to our Website:** \*\*Static assets\*\* are a term for objects we render in our website that when sent to the server remains unchanged like an \*\*image\*\*. For this purpose we will head over to the files section and create another folder - `static`. So if we upload a file inside this, we can access it using the url of the server where we are rendering the `route('/')`, by adding `/ static/ image\_name`.

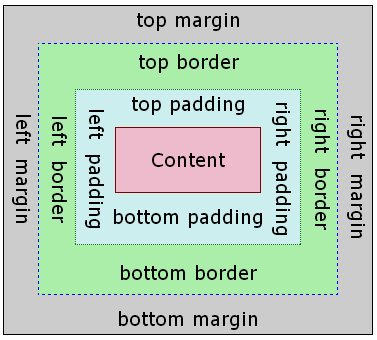


**Styling the page using CSS**

CSS - Cascading Style Sheets, is used for styling your web pages. CSS usually functions around a few major concepts:

-Selectors: `Id` and `Class`(https://web.dev/learn/css/selectors/ )

- The Box Model -(https://web.dev/learn/css/box-model/)



* The best way to add CSS is to do a quick CSS tutorial, you can refer to: <https://web.dev/learn/css/>

All code for CSS can be added to the `<style>` tag of your html document. For eg, we are using the code below to add styling to our website as per our wireframe.

<DOCTYPE html>

<html>

<head>

<style>

h1{

font-family: Roboto;

text-align: center;

color: rgb(180, 180, 100)

}

h2{

font-family: Roboto;

color: rgb(180, 180, 100)

text-align: center;

}

color: rgb(80,90,90)

}

#container{

max-width: 720px;

margin: 0 auto 0 auto;

}

</style>

</head>

<body>

</body>

</html>

**Using Bootstrap for faster Development**

Instead of typing whole CSS manually for every website, you can use a pre existing set of styles that have been created by some good designers.

You can head to Bootstrap's quickstart guide: https://getbootstrap.com/docs/5.2/getting-started/introduction/,

and follow the steps below to add some extra tags within the `<head>` tag to ensure the proper working of bootstrap:

* Add the follwing meta tags:

```<meta charset="utf-8">

<meta name="viewport" content="width=device-width, initial-scale=1"> ```

* For CSS add a link tag:

```<link href="https://cdn.jsdelivr.net/npm/bootstrap@5.2.0/dist/css/bootstrap.min.css" rel="stylesheet" integrity="sha384-gH2yIJqKdNHPEq0n4Mqa/HGKIhSkIHeL5AyhkYV8i59U5AR6csBvApHHNl/vI1Bx" crossorigin="anonymous"> ```

This contains some pre-existing styles that immediately get applied to our page. reload and see the changes straight away. So now, all we need to do is add appropriate classes to style our website according to preexisting styles by bootstrap developers. Let's look at a few classes we will add to make our website look more presentable and ready.

* For Centre Aligning your text: `class = "text-center"`

- Margin Top: `mt-margin\_size`, Margin Bottom: `mb-margin\_size`

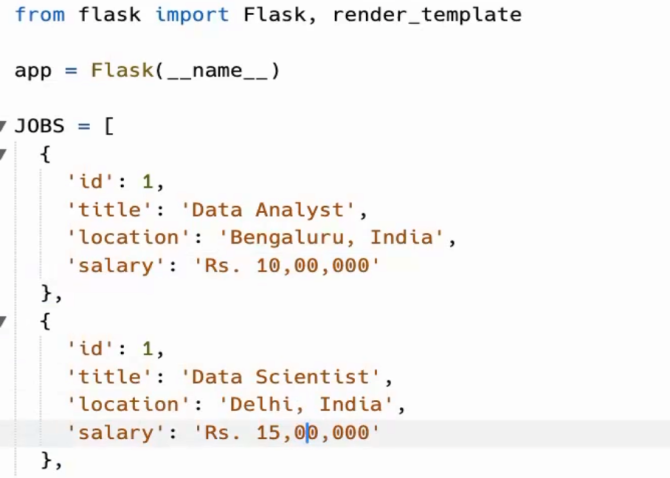
**Step 3 - Dynamic Data & Cloud Deployment**

* Render dynamic data using Jinja template tags ,
* Add an API route to return JSO,
* Deploy the project to Render.com  **,**
* Connect a domain with Render deployment

**Dynamic Data**

it is not a good idea to list all the jobs in HTML file format, because everytime we have to add a new job, we might have to go to the html file and make changes. So data is stored somewhere like a database where administrators can directly make changes, then information is fetched from their to display on the website.

* We will create a dictionary of jobs that we want to pass into our website.



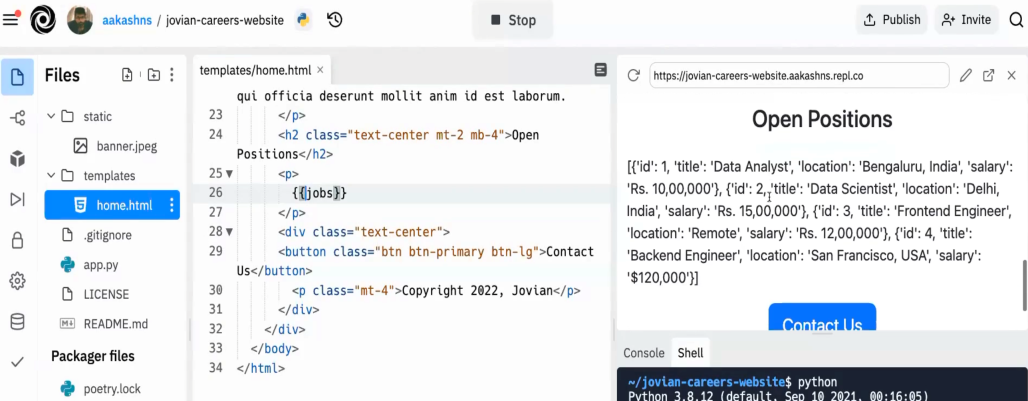
* Then we will add an argument in the `render\_template` as `jobs` and then we can head to html to add add this with the help of a special template called \*\*Jinja Template\*\*.

```def hello\_jovian():

return render\_template('home.html', jobs=JOBS) ```

**Jinja Template : -**It is a text-based template language and thus can be used to generate any markup as well as source code. use jinja template to customize html tags. That is exactly what we are going to do here. Here's a cheatsheet on using Jinja Template to add fields into your flask app: https://www.codecademy.com/learn/learn-flask/modules/flask-templates-and-forms/cheatsheet

* We will head over to `home.html`, to where we want to insert these jobs and add: `{{jobs}}`. You'll notice that whatever information we put inside our `jobs` dictionary will appear as a string:



* We can also use this to pass other variables like `company\_name = 'Jovian'` and use jinja template to render the company name wherever within the `home.html`.

**For Loop Inside Template**

These templates support a special syntax to select particular information from python data structure. We can access the elements within a dictionary just how we would in python. So let's get the `title` from the `jobs` dictionary.

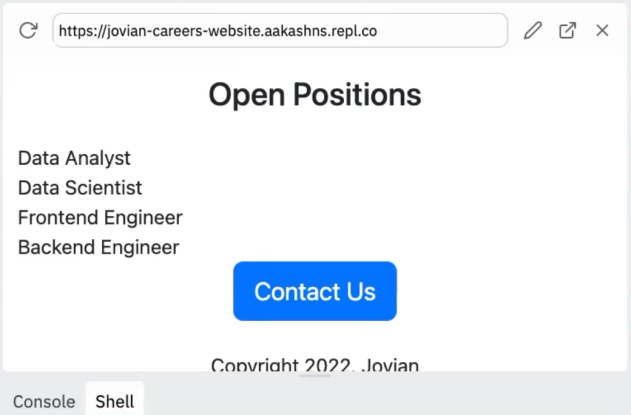
**Syntax**:

{% for job in jobs %}

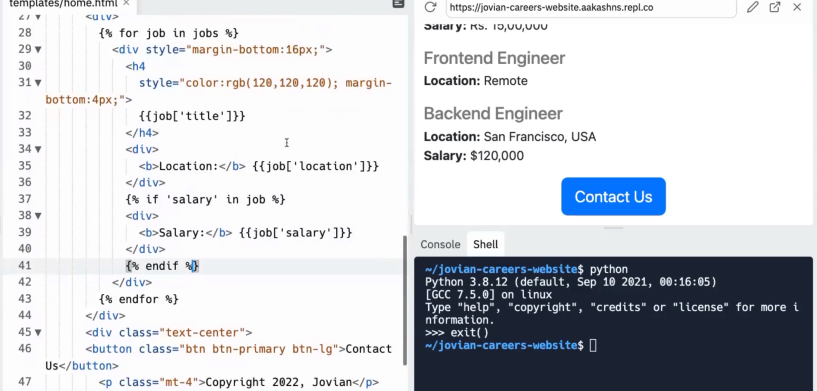
<div>{{job['title']}}</div>

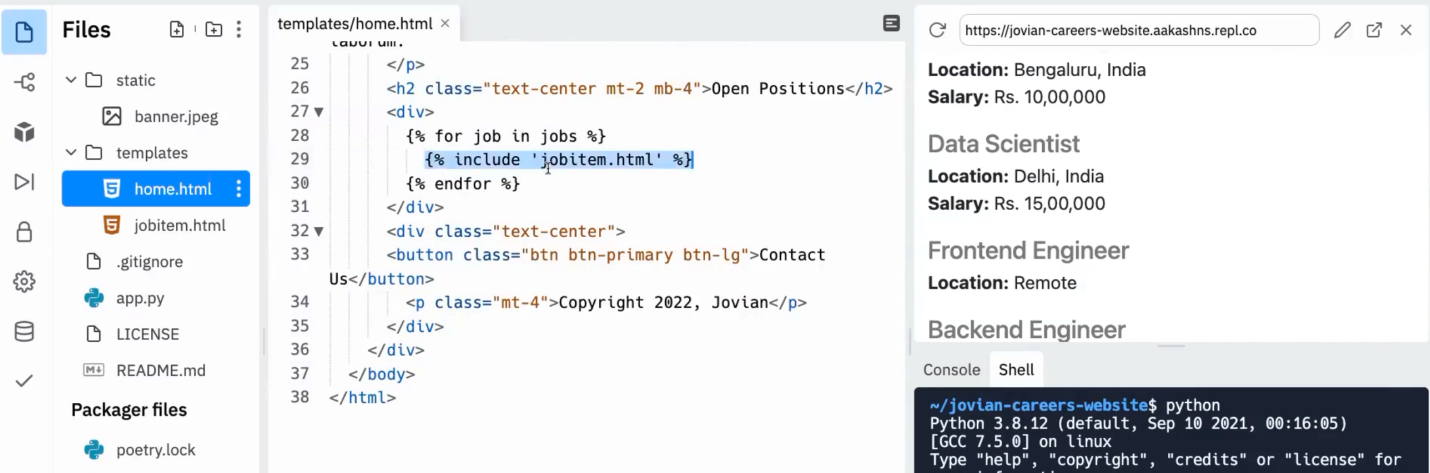
{% endfor %}

This is what the above piece of code will result to:



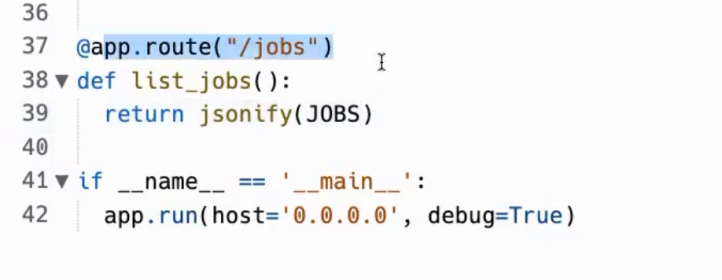
Final code and output looks like this:



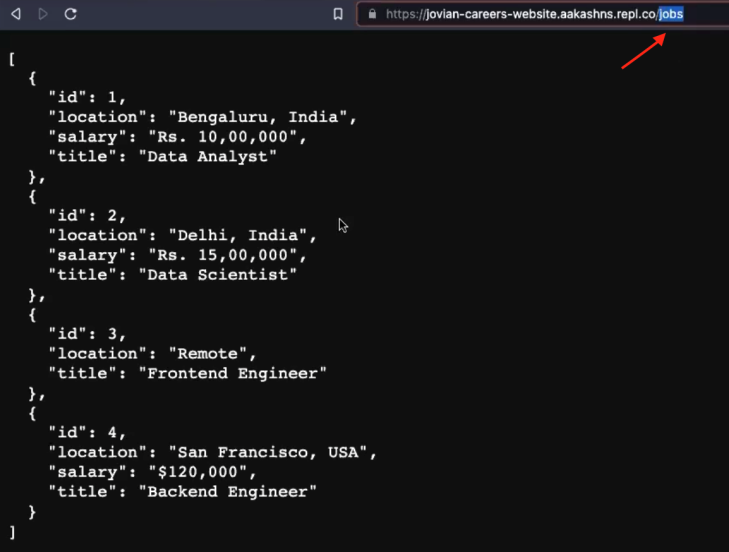
Another thing we can do here is, to add all the jinja template code to another file called `jobitem.html`, and add `{% include 'jobitem.html' %}` instead, which should look something like this: 

**Adding API Route to Return JSON**

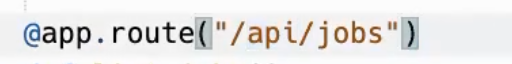
Another way to represent dynamic data is using API - Application Programming Interface. For this, instead of rendering a tempplate, we can simply call a JSON file. JSON is simply \*\*JavaScript Objects\*\*. For returning a JSON string, we will use the function `jsonify`, and instead of routing `('/')`, we will route `('/jobs')` and return our json string. We can do this with the help of a function.



Now we have added another URL to our website, so if we go ahead and add extend /jobs on our home url we will be able to see:



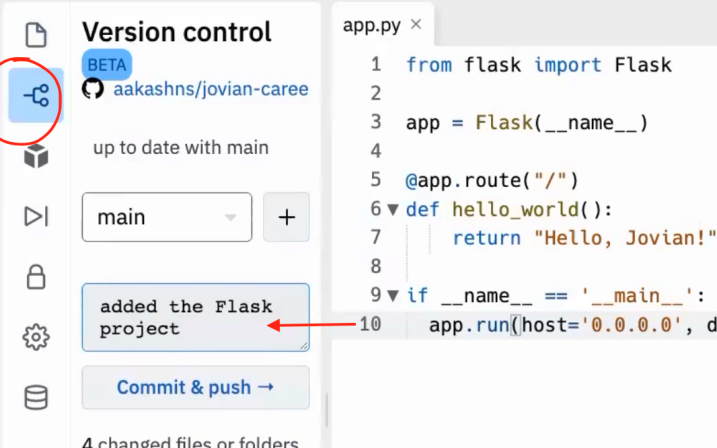
This is exactly what Rest API or JSON API stand for. i.e our webpage is not just returning html but the same information is accesible with JSON.

We generally use the below route to differentiate this from html pages. 

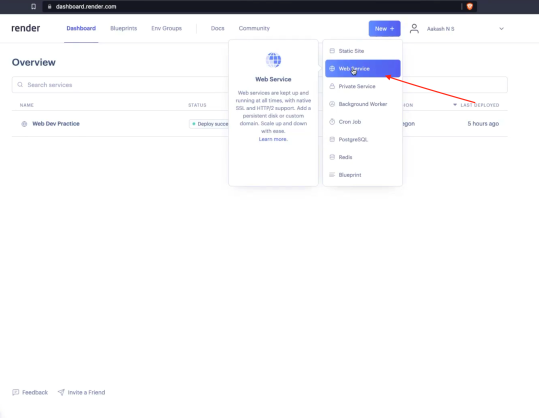
**Deploying Project**

To put your website into production you need to figure out some cloud platform to deploy your website. We will be using https://render.com/ for deploying our websites. There are more platforms like AWS, Heroku, Google Cloud etc... but Render is very easy to work with.

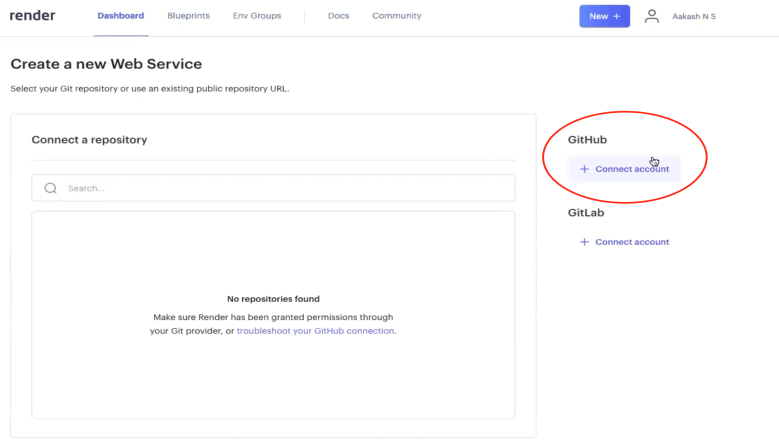
The first step is to head to the \*\*Version Control\*\* tab in the left sidebar to commit your work to GitHub. Like we saw earlier, we will simply write down the changes we made, and press Commit&Push.



* We can head to https://render.com/ and create an account. Render provides a free plan to get started.
* Once you have created an account you will be able to access a dashboard. Go ahead and press New+>WebService:



* Again, we need to connect to github. We need to allow render to pull our code fom github by clicking on 'Connect Account'.



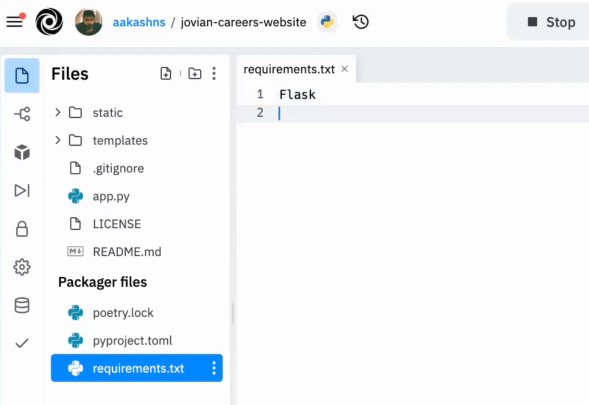
* We will click on 'Connect' and now, we have to configure our code which is present on GitHub. For this, we need to provide the following details:

**Name**: You can name it anything, its for internal reference.

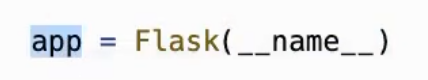
**Enviroment**: python3

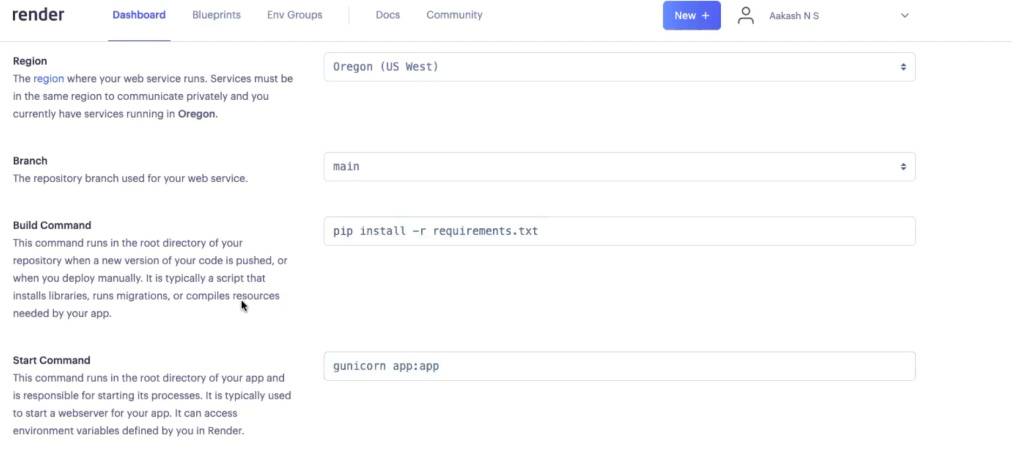
**Region**: It reflects where in the world the server is going to be. You might want to choose a place with less earthquakes😆.

**Built** **Command**: The build command is an important piece of code/command that needs to run to keep track of the libraries that need to be installed to run your project. For this we create a `requirement.txt` file and add `Flask` to it.



This is an accepted standard in the python ecosystem wherein we put our requirements in a `requirement.txt` file for our python project to run.

Another library we will add to our `requirement.txt` file is `gunicorn`. `gunicorn` is a helper library for Flask for production with python, you can look up for it. And then we can commint the changes**. Build command**: `pip install -r requirements.txt` ,**Start Command**: For starting the server we will call `gunicorn`, the name of the file we want to be executed i.e `app:`, and then the name of the Flask app. 



This deployment process differs for each site. You can always look into the documention of the website you want to deploy your website in.

**Conecting a Domain :** To add a custom domain, you will need to buy a domain.

**References**

- Render.com deployment docs: https://render.com/docs/deploy-flask

- Google Domains: https://domains.google.com/

- Heroku flask deployment: <https://devcenter.heroku.com/articles/getting-started-with-python>

**Step 4 - Functional & Aesthetic Improvements**

* Add a Navbar and Footer from Bootstrap
* Add `mailto:` links for the buttons
* Make the website mobile-friendly (responsive)
* Refactor templates into reusable components

**References**

* Bootstrap examples: <https://getbootstrap.com/docs/5.2/examples/>
* Mailto Link Generator: <https://mailtolink.me/>
* Mailto Link tutorial: <https://www.freecodecamp.org/news/mailto-link-how-to-make-an-html-email-link-example-code/>
* Bootstrap docs: <https://getbootstrap.com/docs/5.2/getting-started/introduction/>
* Replit Account: https://www.freecodecamp.org/news/how-to-use-replit/