Outlab 4: Django, HTML, PostgreSQL (Bonus: CSS and Bootstrap)

This problem statement is going to be a bit longer than your previous outlabs, so please bear with us :)

But on the brighter side:

- You'll learn a lot of exciting and essential stuff that will be required *(later).*
- As much as **80%** of the outlab can be solved by diligently following and understanding each step!

Topics we wish to touch upon:

Django, Django Templates, HTML, API Calls, PostgreSQL, Heroku **[125 points]**

Bonus: CSS and Bootstrap **[25 points]**

General Instructions

- This outlab carries **125 points** that we expect everyone to make an effort to complete and an additional **25 bonus points**, for those who wish to explore CSS Styling. **Please note: The total marks for this outlab will be capped at 125 and any bonus you receive beyond that won't be carried forward to future outlabs.**

- **NOTE :** This outlab will be **manually graded**. You can choose to code in whichever environment (Linux/Windows, with/without Docker) works for you.

- Please make sure you understand what you've written. You might be asked to explain your code at a later point in time.

- Don't worry too much about the grading. Focus on getting the basic functionalities right. This outlab will be manually graded and we will be very lenient as long as there's honest effort.

- We have provided you with a sufficient set of links throughout this problem statement. Please feel free to rely on those or any other tutorials that you find interesting. You are free to copy bits of code from there and adapt it to fit this outlab.

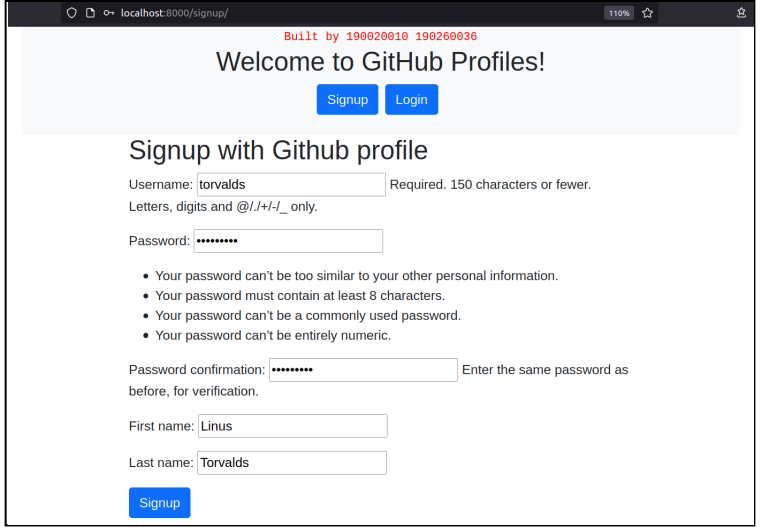
- **Note: Standard plagiarism rules hold and you must cite any major references you used in the `references.txt` as usual.**

- **Deadline for submitting this outlab is 19th September 23:59.**
  Note that there will be NO extensions to the deadline for anyone, as an entire month is a lot more than sufficient to complete the outlab.

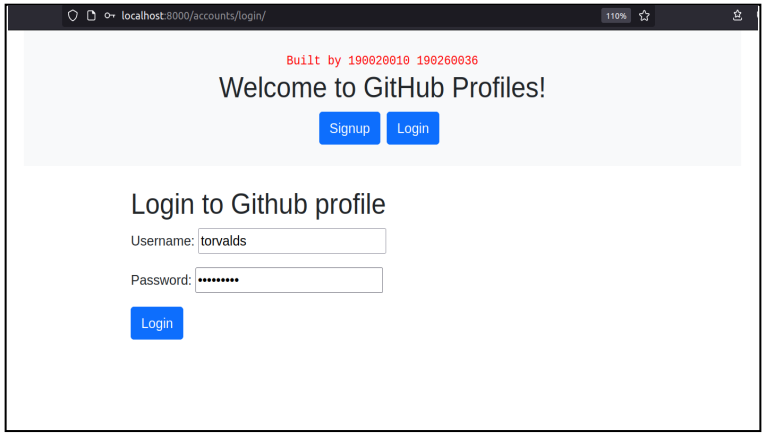
- **[Note: It is necessary to use GitHub for this outlab. You'll submit the link to your repo for this outlab and the link to the deployed Heroku app (more on this later). You'll also need to ensure that the final commit to that repo is before 19th September 23:59]**

0. What exactly are we going to build? (Some motivation to read this long problem statement :p):

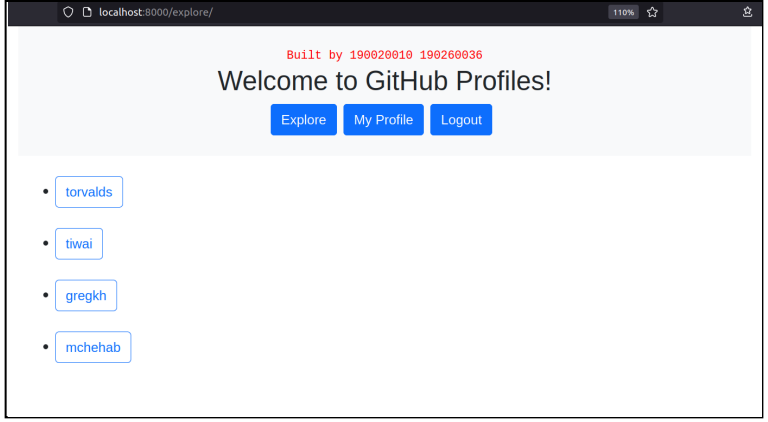
"Github Profiles" is a website where users can share their GitHub account's statistics. Here are the basic functionalities and pages that we need to make :



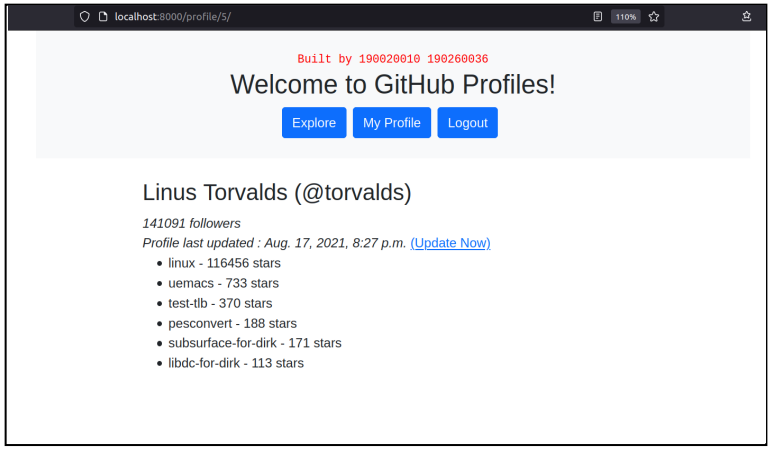
1. A '**signup**' page, where users can enter their (**GitHub) username**, **password**, **password confirmation**, **first name** and **last name** to create an account (you need to ensure that all fields are non-empty before a user account is successfully created, except the **last name** field which may be kept empty).



2. A '**login**' page for logging in users with their **GitHub username** and the **password** they chose during signup.



3. An '**explore**' page that lists all the users of our website. The list should contain usernames and hyperlinks to the corresponding user's profile page.



4. A '**profile**' page lists the user's number of followers on Github, their repositories' names (ordered in descending number of stars) and the last time their profile was updated (synced with Github). When a logged-in user views their profile page, they should have an option to update their profile (i.e. populate their profile statistics with the latest GitHub data).

5. Users should be able to '**logout**' and be redirected back to the login page.
6. A **dashboard** that displays hyperlinks *only* to '**explore**', '**my profile**' and '**logout**' when a user is logged in and *only* to '**signup**'

and '**login**' when the user is not.

[Note: Please ensure you display your roll numbers either in the header/footer of every page. You'll get a more detailed overview in Section 4]

1. First things first (Setting up the environment):

We'll first learn how to set up a virtual environment in python. So what is a virtual environment, and why do we need it?

You'll often use packages that don't come as a part of the python standard library (For example: You must have used NumPy, SciPy, Pandas, etc., in your last outlab). While using these external third-party or open-source packages, you must take care of the changes they introduce with each new release. To summarise, the code that works fine in django==3.1 might throw errors if you upgrade to django==3.2.

Here comes our saviour, virtual environments! A virtual environment allows you to specify the versions of the packages you want to use for a particular project without worrying about which versions of the same packages you are using in a different project. Read more here if you are interested.

Setting up:

1. Open your project folder and setup the virtual environment.
   ```
   $ sudo apt install python3.8-venv
   $ python3 -m venv outlab-env
   $ source outlab-env/bin/activate
   ```
2. Now you are in a virtual environment called **`outlab-env`**, with zero python packages installed. You can start by installing Django. **`pip install django.`** Continue installing whatever you might need on the go.
3. Whenever you wish to work on your project, you'll need to **`source`** the environment using **`source outlab-env/bin/activate`**

[Make sure to update your final `requirements.txt` before submitting. Lookup `pip freeze` to do this.]

[Points: 10]

2. Setting up the Django Project

We would highly recommend going through this set of tutorials alongside solving the outlab.
To setup the initial boilerplate code required for a **`Django`** project, we follow Part 1 from the tutorial linked above. The essential commands are :

1. **`django-admin startproject <your project name>`**
   `[this will create (mainly configuration) files django requires for working & put them in directory named <your project name>]`
2. **`cd <your project name>`**
3. **`python3 manage.py startapp <your app name>`**
   `[this will create an app under the <your app name> directory]`
4. **`python3 manage.py runserver`**
   `[this should start the local django webserver at port 8000. visit localhost:8000 through your browser to check the website]`

After setting up the project and app, head over to **<project_name>/settings.py**. You will notice a variable **SECRET_KEY** assigned to a random-looking string. Django uses this key to sign cookies or other authorization tokens (read more here: link). It is also used to hash passwords while storing them in the database. A leaked **SECRET_KEY** would allow users to impersonate others by forging cookies/session tokens and could possibly even leak user passwords stored in the database.

Follow this link to learn how to store the **SECRET_KEY** without letting the entire world know your **SECRET_KEY.**

[Points: 10]

3. Setting up a PostgreSQL DB

Django, by default, starts with a **`sqlite3`** database. SQLite is a lightweight DB and is suited for small applications. It won't be an issue for a toy project like this outlab, but let's introduce an awesome database that can be used in production level code!

> **PostgreSQL** (>=9.6) : An open-source relational database.

The best thing about Postgres is that Django officially supports it, so you don't have to write any manual SQL queries (which we had to last year, and it was painful). It has a lot of scaling advantages over SQLite, which we won't delve into. But in layman's terms, SQLite locks the entire database during a write operation(so only one write operation can occur at a certain time, which is bad if our user base is large or we want to scale up our application). On the other hand, PostgreSQL does something known as Multiversion Concurrency Control which speeds up things. [Read more here, if interested]

So let's install **PostgreSQL** and **PgAdmin4** locally so that you can actually see what's going on in the database:

1. **sudo curl https://www.pgadmin.org/static/packages_pgadmin_org.pub | sudo apt-key add**
2. **sudo sh -c 'echo "deb https://ftp.postgresql.org/pub/pgadmin/pgadmin4/apt/$(lsb_release -cs) pgadmin4 main" > /etc/apt/sources.list.d/pgadmin4.list && apt update'**
   [this will add pgadmin4 to your repositories]
3. **sudo apt update**
4. **sudo apt install postgresql postgresql-contrib pgadmin4**
   [this should install Postgres 13]
5. **sudo -i -u postgres**
   [switch over to the postgres account]
6. **psql**
   [this opens the postgres shell]
7. **CREATE DATABASE <your db name>;**
8. **CREATE USER <your unix user name> WITH PASSWORD '<password>';**
   [Now we'll do some default settings for our user]
9. **ALTER ROLE <your unix user name> SET client_encoding TO 'utf8';**
10. **ALTER ROLE <your unix user name> SET default_transaction_isolation TO 'read committed';**
11. **ALTER ROLE <your unix user name> SET timezone TO 'Asia/Kolkata';**
    a. In the first line we are setting the default encoding to UTF-8 expected by

Django.
   b. With the second line, we set the default transaction isolation scheme to “read committed”, which blocks reads from uncommitted transactions.
   c. By default, the Django timezone is UTC. We are setting our database’s timezone to Asia/Kolkata :)
12. **GRANT ALL PRIVILEGES ON DATABASE <your db name> TO <your unix user name>;**

We are almost done with our database setup! [If you want to get a feel for **PgAdmin**: Now head over to **PgAdmin4.** You’ll be able to find it in your applications list. Create a new server(name it whatever you want), set the **hostname** to **localhost, username** to **<your unix username>** and **password** to what you set at **step 8** above.]

Our Postgres database is up and running. Now we’ll need to add it to Django somehow!

Head over here and figure out how to add the database credentials to set up the Postgres database you just created.
[**Note**: You must not save the username, password and other sensitive information in **settings.py** while setting up this database. Figure out a hack using the **SECRET_KEY** trick you saw earlier]
[Points: 15]

4. Let’s build it! [60 points]

First, we need some way of authenticating users when they log in, store user’s credentials in a database when they sign up and a way of logging them out. Luckily, Django has a very easy way to do all this. Read up some of these links to figure it out :

1. https://learndjango.com/tutorials/django-login-and-logout-tutorial
2. How to Define a Custom Django Model (Django Tutorial) | Part 11
   Creating the Django User Registration Form (Django Tutorial) | Part 15

If you followed the tutorials linked above, you might have ended up using Django's User Creation Form. However, our requirements are a bit different. We need to ensure that the First Name field is non-empty, which is not the default behaviour of Django's User Creation Form. Read up and figure out ways to do this on the internet.

Design a **dashboard** as mentioned in the 6th point of Section 0 that is displayed in all **views** (except the logout view).

We get down to the main problem now. With the user authentication and dashboard working, our application now needs a way to **`fetch`**, **`store`** and **`display`** github statistics of a user.

`FETCH :` To fetch information like the number of followers of a user, your application will have to issue a web request to **`https://api.github.com/users/<github username>`** and for listing user's repositories[1] a request to **`https://api.github.com/users/<github username>/repos`**. You may want to use the built-in **`requests`** module in python to do this. Here is a tutorial to get you started.
The github urls above return text formatted as a json. You can convert this json formatted text into python dictionaries or an array of them by using the built-in **`json`** module or directly using the method in the tutorial linked above.

As we are performing web requests, they may end up failing for several reasons. If this occurs, you do not want your Django web server to throw the exception back at the user. Perform appropriate exception handling to make sure this does not happen.
(NOTE : number of stars of a repository is the value in the **`stargazers_count`** field of the json)

`STORE :` Create a model named "**Profile**" to store user specific data like number of followers, timings of last update to the profile and any other field that you deem necessary.
You may also want to link the "**Profile**" model to Django's default "**User**" model that you used for authentication. In a way, we want to extend Django's default User model with extra fields. Googling for the same gives us a lot of ways to do it.

Create another model named "**Repository**" with fields such as name, number of stars and profile of the owner of the repository. Set up appropriate relationships between "Profile" and "Repository" models to mimic the real world behaviour that a single user profile might have multiple repositories linked to it.

You may want to have a look at this tutorial to learn more about how **models** work in Django. (Hint: There are examples in that link, which will suit your needs!) You will also need to write code that updates appropriate fields in the database when new data is fetched from github. Something like the "**`Update Now`**" button we have in our "**`profile`**" page above.

DISPLAY : Create appropriate views for **1.** listing all users of the website ("**explore**" page) and **2.** displaying statistics of a particular user ("**profile**" page). Remember to conditionally render an "Update Now" link/button in the **profile** page of a user if the user being viewed is the same as the logged in user. The "Update Now" link/button, when clicked, will go through the described **fetch, store** and **display** procedure.

You may refer to Tutorial 3 (read other parts to get the full picture) for figuring out the basics of how **views** and **templates** work in Django.

5. Heroku Deployment

Now we will move on to deploy our newly created website to Heroku. First things first, head over to Heroku: Cloud Application Platform and create a free account. You'll also need the Heroku CLI. Get it here.

Setting up your project for **Heroku**:

1. To deploy a **Django** project, **Heroku** needs **3** special files:
   Procfile, runtime.txt and requirements.txt
   at the base of your project.
2. Read up **Heroku** documentation and figure out what these files are and what to write in them.
3. While using **PostgreSQL DB** locally, we could directly specify the **DATABASES** credentials in the **settings.py.** But, now that we want to deploy our app on **Heroku**, we will use some cloud-based **PostgreSQL DB**. Read up on how to make a free **Postgres DB** on **Heroku** and link it in your **Django settings.py**. [Hint: Look up **dj-database-url**]

Setting up the **Heroku-GitHub** pipeline:

1. Open Heroku in your browser. You will reach the initial dashboard. On the top right side, you'll see a "**New**" option.
2. Create a new app and connect it to your **private GitHub** repository that you (hopefully) must have been using for this outlab.
3. Enable auto-deployment if you wish. In any case, go through the page you are currently on. Most of the stuff will

be self-explanatory, and it's just a few more clicks to get your app deployed.

4. Also note that you'll need to store **`SECRET_KEY`** somehow as a Heroku environment variable (search for: heroku config vars). You might face some other issues as well. We won't list those out. You'll need to check the logs and figure it out.

You might find **`heroku logs --tail`** and **`heroku run python3 manage.py <something>`** commands helpful for debugging later.

Note: These are just pointers to what you need to do. You are free to follow whatever you find convenient, be it a blog describing **`Django x Heroku`** deployment or some detailed documentation. As long as you submit a working link to your deployed web app, it's fine :)

Submit your deployed Heroku app link here before 19th September 23:59.

[If deployment successful] [Points: 30]

6. [Bonus] Add some flair to your bland website

You must have used Django Templates for creating the interface of your web-app. If you were following our steps, it's quite possible that your website is plain HTML and *bland*.

Let's fix that:

1. Read up about CSS and what it is here. Feel free to explore other resources.

2. Figure out how to add CSS to your Templates. This and this might be a good start.

3. Now that the **`CSS`** you added finally shows up on your browser, here comes the fun part: Go to your **`settings.py`**, turn **`DEBUG`** to **`False`** and redo the **`runserver`** step. And quite possibly, your **`CSS`** styling is gone. **Why?** Django is NOT a file-server and it's not designed to serve files in production. **`CSS`**, **`JS`**, images, etc are termed as **staticfiles** and Django discourages from serving **staticfiles** using the Django development server. They term it as "grossly inefficient".

a. The best solution for serving files is context dependent. **The quickest solution here is to use a python staticfile server. Look up whitenoise and figure out how to configure it with Django.** Once you are done with this, **`CSS`** should be back even with **`DEBUG=False`**. This is particularly important for Heroku deployment, since we don't want to see **`DEBUG=True`** in your deployed website.

Style any **5** entities using CSS. Anything works: lists, buttons, text-colours, fonts, …

Note: `Inline-CSS` doesn't count for this bonus. You must create a separate file for `CSS` and link it in your django templates. Also, `CSS` must be rendered in your heroku website with `DEBUG=False` to get this bonus.

[Bonus points: 15]

Still looks bland and writing CSS is painful? Let's learn about Bootstrap:

Bootstrap is a famous front-end open-source toolkit (open source CSS framework). [To put it simply: A bunch of awesome people wrote customizable CSS code which we can import and use straight away.] Whatever frontend framework you use later (say **Angular, React, Vue**), you'll end up using some styling toolkit one-way or the other.

Let's setup bootstrap:

1. Head over here and figure out how to add bootstrap to your Django Templates. (It's recommended you complete the plain CSS step before this, because as you'll see adding bootstrap is very similar to adding plain CSS).

Style any **5** items with Bootstrap. Everything counts: containers, margins, links, buttons …

Don't worry about the count of 5. As long as you've set it up correctly and shown some usage, that's enough.

[Bonus points: 10]

Submission & General Instructions

- Don't worry about the fine grading done in Section 4. Focus on getting the basic functionalities right. We will be very lenient with grading as long as there's honest effort. You'll get the full 30 points

in Heroku deployment, even if just parts of your deployed app work.

- Please ensure that your deployed Heroku application has **`DEBUG=False`**. To add to this, it should not expose the **`SECRET_KEY`** or **database credentials**. We'll be deducting points for these.
- **IMPORTANT Note: Your GitHub repo should be private at all times. There will be severe consequences if we find a public repo for this outlab. Also, make sure you add "cs251lab4" as a collaborator to your private repository for evaluation purposes and push your final commit to the repo before 19th September 23:59.**

- **For this outlab, you'll need to submit 3 things:**
  - Submit the **link** to your **1. Deployed Heroku app** and **2. private GitHub repo here before 19th September 23:59.**
  - The final directory structure will look something like this :

```
<your project name>
|
`--<your project name>
|        |
|        `-- django related files…
|        `-- django related directories...
|
`--<your app name>
|        |
|        `-- django related files…
|        `-- django related directories…
`-- django related files…
`-- .env (or any file you used for environment vars.)
`-- requirements.txt
`-- references.txt
```

Rename top-level **`<your project name>`** directory to **`<rollno1>-<rollno2>`**
and compress it using the following command :

`tar -czvf <rollno1>-<rollno2>.tar.gz <rollno1>-<rollno2>/`

Submit the **`<rollno1>-<rollno2>.tar.gz`** file generated from the above
command on Moodle from the account of the lexicographically smaller roll number
before 19th September 23:59.

Late Penalty:

- **Standard late penalties apply. You'll get a 2 hour leeway beyond 19th September 23:59 with 1% penalty and no submissions will be accepted after that.**

---

[1] **Requesting https://api.github.com/users/<github username>/repos will give us the lexicographically smallest 30 repositories. Consider only these 30 repositories for the outlab. Other repos can be accessed by appending ?page=2, ?page=3 .. to url.**