E-14a

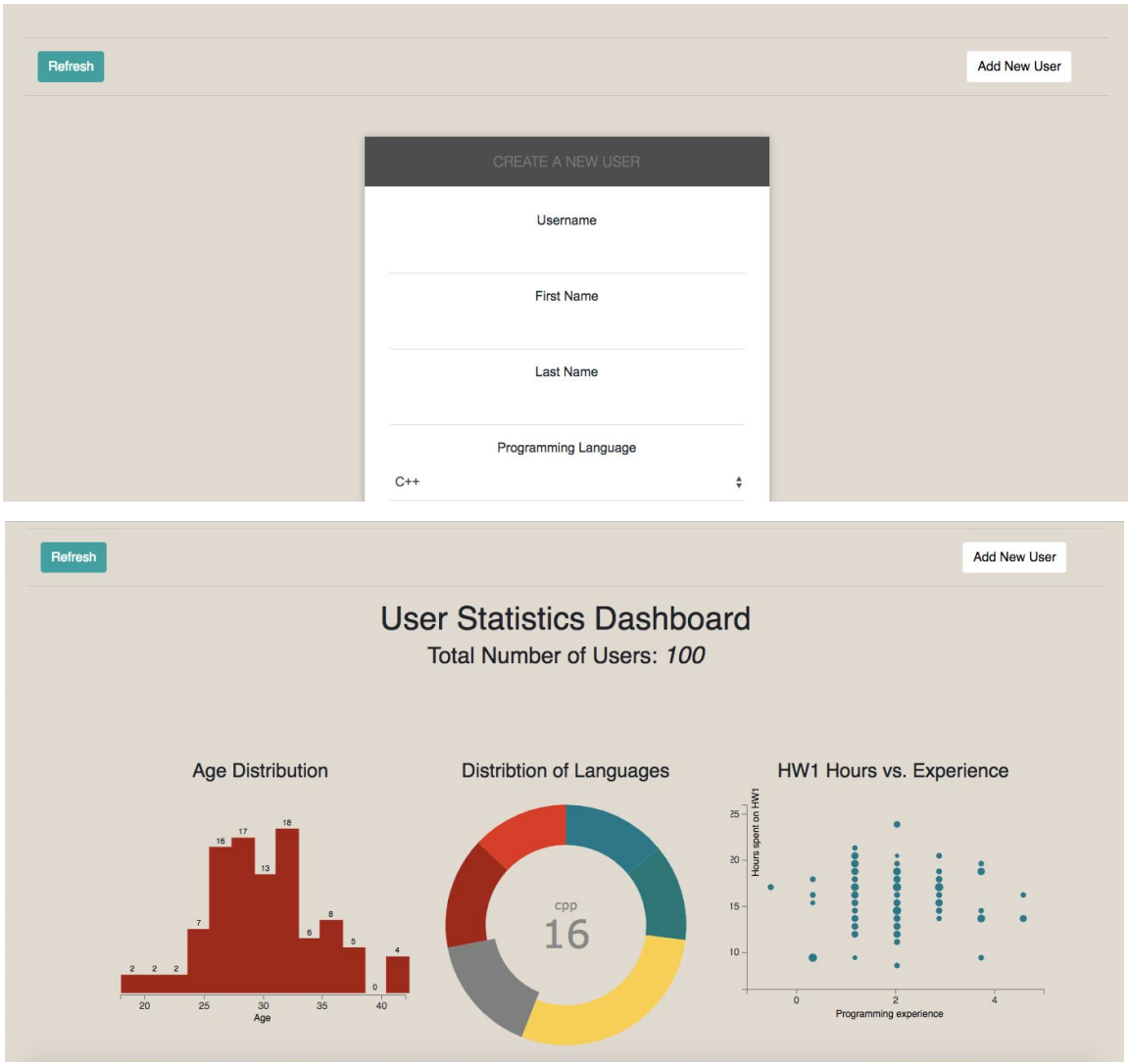
HW 2: Instructions

Learning Objectives

- Database modelling
- Working with DDL and DML
- Use Flask extensions to connect database with Flask
- Use Flask extensions to work with Web Forms
- Connect frontend and backend
- Use d3.js for data visualization
- Create dynamic web app
- Link visualizations
- Deploy your app using Heroku

HW2 PROJECT OVERVIEW

In September of 2018, a new course named *E14a - Web Apps for Data Analysis* was launched at the Harvard Extension School. After beginning enthusiasm, students started putting more and more time into their daily tasks - participating in lectures, labs, and working on their HW assignments. E14a staff was under the impression that given assignments might not be challenging, despite the time invested. However, feedback were drawing different picture. In order to understand the difference in individual achievements, we decided to inspect some of the possible relationships and use visual exploration tool to potentially spot any given patterns.

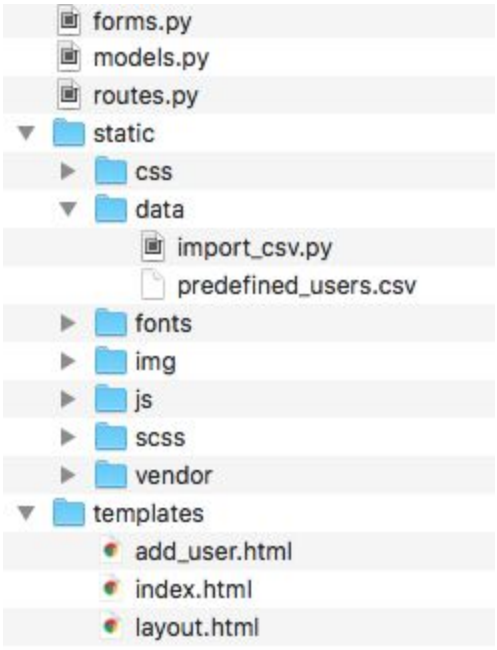


HW2 Preview

Homework 2 description

This homework includes many sequential steps. Please, read the instructions carefully.

Go to *Canvas* > *Modules* > *Homework 2* and download ".zip" file called *hw_2*. Unzip your folder (check the picture below for the folder structure) and follow the next steps.



HW2 Folder structure

Part 1: Working with databases (5 pts)

In the first part of your Homework you will work with databases.

Question 1: Open your Postgres terminal and create database called ***homework_users.*** *Make sure to use this exact (****homework_users****) db name! Connect to the* ***homework_users*** *database and create a new table called* ***users****. The output should look like this:*

```
homework_users=# SELECT * from users;
 uid | username | first_name  |  last_name   | prog_lang | experience_yr | age | hw1_hrs
-----+----------+-------------+--------------+-----------+---------------+-----+---------
```

Hint: check the csv file "predefined_users" to determine attributes data types.

Next, we would like to populate our database with some data. Instead of typing ~100 observations, we will import "csv" file - list of predefined users - into *homework_users* database. Open up your terminal and navigate to the file called *import_csv.py* :

The data folder structure

Open *import_csv.py* file and copy/paste the below code in it.

Question 2: Make sure you go through and understand each line of the code.

```
conn = psycopg2.connect("host=localhost dbname=PUT_YOUR_DB_NAME user=postgres")
cur = conn.cursor()

df_users = pd.read_csv('predefined_users.csv', index_col=0)
for idx, u in df_users.iterrows():
      cur.execute('''INSERT INTO users (username, first_name, last_name,
      prog_lang, experience_yr, age, hw1_hrs) VALUES (%s,%s,%s,%s,%s,%s,%s)''',
      (u.username, u.first_name, u.last_name, u.prog_lang, u.experience_yr, u.age,
      u.hw1_hrs))

      conn.commit()

cur.close()
conn.close()
```

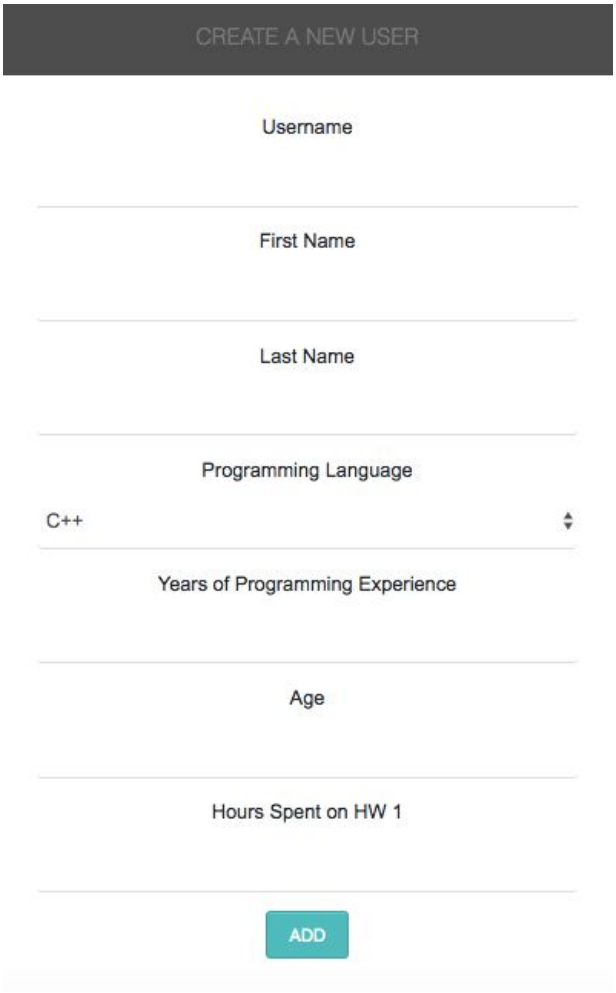
The above code will help you easily populating the databases with record from the “csv” file. How to run it? With your terminal navigate to the folder where *import_csv.py* is located and type:

```
python import_csv.py
```

Use SELECT SQL statement to check is your database populated with 100 names from the “csv” file.

Question 3: Create “Add New User” Web form

We would like to allow new users to participate with adding new information. We want to store that information in the database and later on, to read it directly in order to update the visualizations dynamically. What are the steps you need to accomplish here (*note: if you struggle with accomplishing this part, we recommend going over the Lab 4 again*):



- Open *models.py* and write the model data structure
- Open *forms.py* and write the form
- Open *add_user.html* template and create form elements (*hint: for styling the* `first_name` *field, for example, use* `form.first_name(class="form-control")`)
- Open *routes.py* and follow next steps:
    - Import Python libraries and Flask extensions
    - Configure connection from Flask to Postgresql database
    - Initialize the Flask app for using this database setup
    - Define the secret key
    - Use route() to bind a function to a URL adding a keyword arguments for *index.html* (`'/index'`) and *add_user.html* (`'/add_user'`) pages. Do not forget to add `methods=['GET', 'POST']` to the `route('/add-user')`
    - Passing the values from the Web form into the *Users* object
    - Insert the new user object into the *users* table and save it
    - Redirect users to *index.html* after pressing the ADD button

Part 2: D3 visualization (5 pts)

In this part we will create 3 visualizations: *BarChart*, *PieChart*, and *ScatterPlot*. Before we jump into the implementation, let us work on how to effectively supply the data from the database. Instead of using static files, we want to update our visualizations dynamically.

Open your *routes.py* file and add these few lines of code (**bold**):

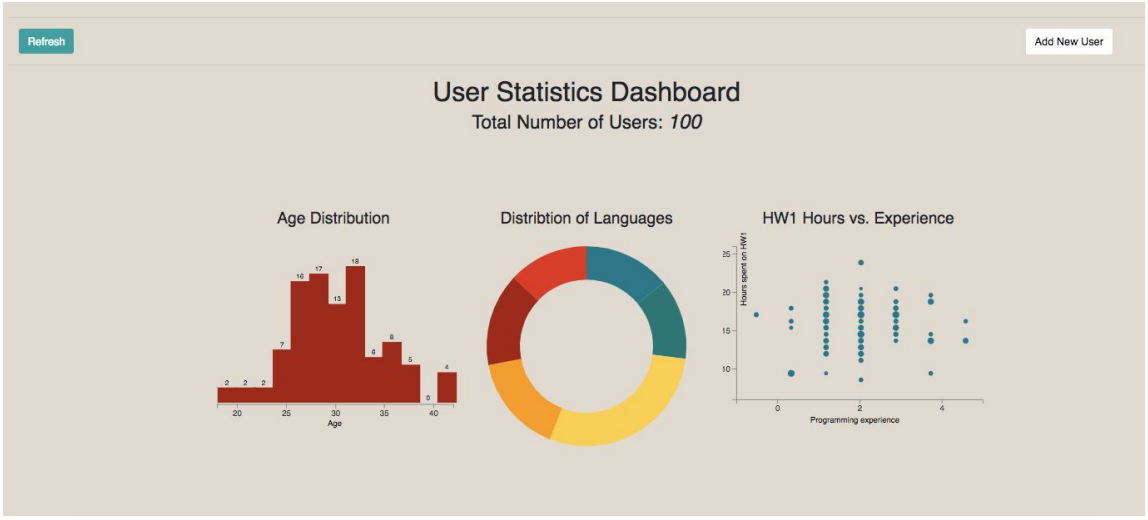
```
@app.route('/')
@app.route('/index')
def index():
    users = User.query.all()
    return render_template('index.html', title='Home', users=users)

.....
@app.route('/load_data', methods=['GET'])
def load_data():
    users_json = {'users': []}
    users = User.query.all()
    for user in users:
        user_info = user.__dict__
        del user_info['_sa_instance_state']
        users_json['users'].append(user_info)
    return jsonify(users_json)

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

The above code reads the data from the database and supply them to the d3 visualizations. It exports the database as json file and pass it directly to d3. It first converts the database table to python dictionary and then makes a json response in return statement. The method *jsonify* is a method of Flask that creates a Response with the JSON representation.

Next, navigate to your js folder where your visualizations are going to be defined (*note: if you struggle with accomplishing this part, we recommend going over the Lab 5 again*):

IMPORTANT: All 3 js files are commented `/* */` to prevent errors. You can uncomment them once they are opened (delete `/*` at the beginning and `*/` at the end of the code). Do this before you continue with next steps!

This is how your visualizations should look like:



Question 4: Create BarChart

Open *barChart.js* file and follow the steps:

- Part 1: Visualize the number of users in the database:

  Total Number of Users: *100*

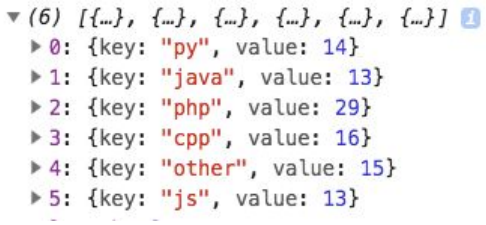
- Part 2: Follow the comments inside the *barChart.js* file to create the histogram of the age distribution

  *Hint: Check **map** to see what data is used here*

Question 5: Working with PieChart (DonutChart) data

Open *donutChart.js* file and follow the steps to create the distribution of programming languages:

- Part 1: Use d3.nest() to aggregate the data. This is the expected output:



- Part 2: Define ordinal scale to map the above data keys to different color values:

```
'#1b7688','#1b7676','#f9d057','#f29e2e','#9b0a0a', '#d7191c'
```

Question 6: Create ScatterPlot

Open *scatter.js* file and follow the steps:

- Part 1: Convert `experience_yr`, `hw1_hrs` and `age` into numerical values.
- Part 2: Follow the comments inside to file to create the *ScatterPlot* that shows the relationship between the number of hours spent on HW1 and previous programming experience.

Extra Credit Questions

The Dashboard hasn't provide any exploration functionality yet (e.g., filtering, subsetting, aggregating … etc).

1. It would be great to have this visualization updating based on programming language. For example, when the user clicks on the *PieChart* arc, both *barChart* and *ScatterPlot* should update accordingly, displaying the users that falls into that specific, programming language category. (2 pts)
2. Use scatter-plot brushing to select the areas of interest and update the entire Dashboard accordingly. For more info: https://bit.ly/2N6ytt6 (2 pts)

You already have developed the C (create) of CRUD functionality in your Flask application. Now, you should extend the application with Read, Update, and Delete functionalities. That way, you'll be able to manipulate the data in the "users" table without having to type an SQL command. (2 pts)

1. READ functionality should be realized as an HTML table that contains users' data, as well as links to update/delete each record. Also, that page should contain a link to (already existing) CREATE page (add-user).
2. UPDATE page should display a form
3. DELETE link on READ page should lead to a page that deletes selected record from database and redirects user to the READ page

   (code: return redirect("http://www.example.com", code=302))