Project (100 points)

This project shall be done in a group of **3-4** students. Individual submissions are not accepted because the goal is to prepare you to work in teams, which is a skill extremely important for your career. Moreover, this is also important for the following ABET outcomes for this class:

- *Communicate effectively in a variety of professional contexts.*
- *Function effectively as a member or leader of a team engaged in activities appropriate to the program's discipline.*

If you haven't already done so, please indicate the group members in the spreadsheet posted on Campuswire by Oct 29, 2021.

The group may choose to implement the project using *one* of the following:

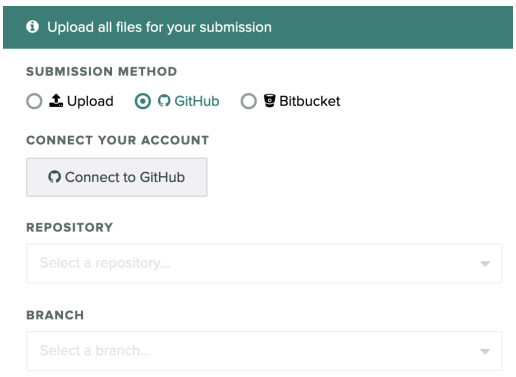
- A Web application developed using Python **+** Django **OR**
- A program with GUI developed using Python **+** TkInter

If your project need dependencies, then specify them on a `requirements.txt` file

- Each line in the file has the library name and the version in the following format `<library-name>==<version>`
    - Example: requirements.txt
- **(For Django projects):** you're *encouraged (but not required)* to use Bootstrap

How to Submit

- Create a **private** GitHub repository and share it with the the **professor** and the **TAs'** GitHub accounts, that are linked below:
    - **Prof Santos**
    - **Latif Siddiq Sunny**
    - **vinilopes03**
    - **Gregory Khvatsky**
- Upload the contents of your GitHub repository on Gradescope



Notice:

- The TAs need to be able to run your code. Thus, please have on your repository everything that is needed to run the code. Runtime errors due to missing files, database setup, etc. will incur points deduction (and potentially a zero, if there was no way for the TAs to run your code and test it).
- **Hint**: make sure to add __pycache__ folders or other folders with binaries (ex: venv) to your .gitignore. This will make your project unnecessarily large, leading to commit errors on GitHub. A sample .gitignore file is included in the course's GitHub repository.

Submissions that do not follow these instructions, will be deducted points. If your repository is public and other groups copy its solution, that would be deemed as **dishonest behavior for all parties involved** (see Honor Code). It is your responsibility to protect your own work. Follow the Academic Honesty policies mentioned in the Syllabus. If you are unsure about whether a specific action is deemed as dishonest behavior, please check with the instructor ***prior to*** engaging in such action. Ignorance of the rules is never an excuse.

Deliverables

This project is split into two phases, each of which with its own deliverables. The templates for the reports, README.md and CONTRIBUTIONS.md are all available on the course's GitHub repository. Make sure that all deliverables are in your private GitHub repository. You can organize your repository in any way you wish, but your README file has to properly explain where things are and how to run the code. Disorganized and/or undocumented repositories will have points deducted.

Phase 1 Deliverables (25 points)

In the first phase, you will select ***at least three (sub-)features (ex Feature 1.1, 2.3 and 3.2)*** from the Project Description section to implement. The deliverables for this first phase are:

- `Phase 1 Report.pdf`: a PDF file that clearly indicates the technology stack used, storage design details, and what features were chosen to be implemented in this first phase (for example: features 1.1, 1.2, 2.2). The report shall also include screenshots of the project demonstrating the implemented features (use the template provided).
- `src`: all the code, configuration files, etc, created so far.

Phase 2 Deliverables (75 points)

In the second (final) phase, you will finalize **all** the features and submit the following deliverables:

- `Final Report.pdf`: a PDF file that includes screenshots of the project demonstrating the implemented features; describes any technical challenges and lessons learned along the way; discusses any paradigm(s) used and how they helped (or not) in developing the project (use the template provided).
- `src`: all the code, configuration files, etc, created so far. This zip file shall also contain the following project documentation files (use the templates provided):
  - `README.md`: it describes how to set up your submission, and run your project.
  - `CONTRIBUTIONS.md`: It describes precisely the specific contributions made by each team member.
- `Peer review questionnaire:`
  - Each team member will *confidentially* fill out a peer-review survey evaluating their team members. If a student does not fill this evaluation, the student's individual project grade will go down significantly. The survey will be released one week prior to the project's final due date on Gradescope and only the professor and TAs will have access. The answers ***won't*** be shared with the team members.

Make sure that all deliverables are in your private GitHub repository & Gradescope!

Technical Constraints

- The system should not assume the inputs are well-formed (i.e., validate the inputs).
- The information has to be saved in a persistent storage:
  - In csv/json/xml/txt file(s) **OR**
  - a SQLite database
- When the project is submitted (Phase 2), the persistent storage should ***not*** be empty. Instead, the persistent storage should include at least 10 recruiters, 10 job postings, and at least 10 candidates.
- The system should have at least 2 windows/web pages (i.e., it cannot have a single window/web page with all the features listed above).
- **[For Django Projects]** The CRUD operations have to be implemented by your solution itself, i.e., the user **will not** interact with the built-in admin interface to create/read/update/delete recruiter/candidates

profiles or positions. Instead, your code would make use of generic views DetailView, ListView, CreateView, etc + templates (HTML Pages) for the user interface.
    - *The project's grade will go down significantly if the project simply uses the admin interface.*

Development Process

- Use a **private** GitHub repository:
    - One of the team members create the repos and grant access to the other members, the instructor, and the three TAs (see the section How to Submit)
    - Each team member should individually push their changes to the repository
- The TAs will use commits history to get a sense of projects' contributions per team member
    - *If it is clear that a team member has not properly contributed to the project, the project grade will be adjusted for that team member*

Project Description

In this project, your team will be acting like a startup company that has been funded by a venture capital firm (Investiny Corp.). Your startup already had done market research, and came up with the elevator pitch below and the core features to be developed as part of the first market release.
Notice:

- *The project scope was intentionally reduced. In real life, features would have been way larger and far more technically complex than the ones listed below.*
- *Your team is welcome to go above and beyond on the features (implementing extra ones), as long as the very basic features listed below are implemented.*

Elevator pitch

We are a startup chartered to create applications for the benefit of recruiters all around the country. The product we envision is called **TinDev**.

At its core, **TinDev** is meant to disrupt and innovate the recruiting process. It aims to facilitate the process of finding the perfect match for a technical position, whether it is a Software Engineering, Quality Engineering, or Full Stack Developer positions.

The successful implementation should make it easy for any candidate wanting to find a job to register and be able to see open positions and demonstrate interest, whereas recruiters can also see qualified candidates within the area.

We want a product whose emphasis is on ease of use, whose navigation is straightforward.

Feature 1: Sign Up / Sign in

The program shall allow a person to sign up to it. On the signup process the user will specify their profile type: candidate or recruiter. Each profile type will request different types of information.
In both profile types, the system will validate the input provided, and, if valid, it generates a unique identifier (ID) for the user. Then, it creates a new user profile, saving it on a persistent storage.

1.1: Create new Candidate Profile

The program shall allow a user to create their ***candidate*** profile. The candidate will provide the following information when creating their profile. The ones marked in * are optional values.

- Name
- Profile Bio (500 characters max) *
- Zip Code
- List of Skills (ex: Java, Python, Git, etc)
- GitHub Username *
- Years of Experience

- Education *
- Username
- Password

1.2: Create new Recruiter Profile

The program shall allow a user to create their ***recruiter*** profile. The recruiter will provide the following information when creating their profile (all of them are required):

- Name
- Company
- Zip Code
- Username
- Password

1.3: Log-in

The program shall allow a user to log-in with their profile by providing their username/password.

1.4: Log-out

The program shall allow a user to log-out.

Feature 2: Recruiter's Dashboard

Once a recruiter is logged in, they have access to their dashboard, where they can:
- View all the posts they have created
- View candidates interested in each job posting
- Hire a candidate

Only recruiters are allowed to create job postings. The program shall allow a ***recruiter*** to create a job post as well as view, update and/or delete the job postings they previously created.

2.1 View all posts

The recruiter shall be able to view all of its previously posted jobs, and be able to filter job postings such that they can view:
- All posts
- Active posts only
- Inactive posts only.
- Posts with at least 1 interested candidates

While listing all the job postings, the following information should be displayed: Position Title, Company, Status, Type, number of Interested Candidates.

2.2 Creating a Post

When creating a job post, the recruiter will provide the following information:
- Position title (ex: "Software Engineer")
- Type (Full Time or Part Time)
- Location (City, State)
- List of Preferred Skills
- Description

- Company
- Expiration Date
    - Date by which the job posting will automatically become inactive
- Status (active / inactive)

2.3 Updating a Post

The recruiter shall be able to select one of the job postings they previously created and update any of its information (except its ID).

2.4 Deleting a Post

The recruiter shall be able to select one of the job postings and **delete** it entirely. Notice that the job posting ID should ***not*** be reused (i.e., the ID is meant to be unique and non-transferrable).

2.5 Make an offer to interested candidates

A recruiter shall be able to select one of their posted jobs and see which candidates are interested in them. The recruiter can select 1 or more candidates to make an offer. The offer should include the yearly salary information and a due date (a deadline by which the candidate has to accept the offer).

2.6 View compatibility with interested candidates

The recruiter shall be able to select one of the job postings and view all the candidates interested in it. The system shall compute a compatibility score, given as a percentage (where 100% means the candidate is a perfect fit, 0% means the candidate's profile does not match the job posting).

- *Your team is expected to come up with its own algorithm to compute this compatibility score (ex: keyword-based match between job's description and candidate's bio or matching the post's required skills vs candidate skills). Feel free to use your creativity. Your report should clearly indicate how this compatibility score was computed.*

Feature 3: Candidate's Dashboard

Once a candidate is logged in, they have access to their dashboard, where they can see:

- Job postings
- Job postings they have demonstrated interest in

3.1 Viewing Job Postings

A candidate shall be able to view posted jobs, and be able to list them based on multiple filtering criteria:

- By job post status (active / inactive only)
- By Location
- By keywords (i.e., post's description contains one or more of the keywords)

3.2 Demonstrating interest / not interested in a job

A candidate shall be able to select a posted active job and demonstrate they are interested 👍 or not interested 👎 in the job.

3.3 Viewing offers

A candidate shall be able to see all the job postings that they had an offer from. For expired offers, the system shall clearly indicate that the offer has expired on day X.

3.4 Accepting/Declining offers

A candidate shall be able to see all accept / decline an offer. Notice that the candidate can only do so if the decision was made timely, i.e., prior to the offer expiration date. After that, the system shall not allow the candidate to accept / decline the offer.

Grading Rubric

The TAs will evaluate the solutions based on the following criteria:

- ☐ **Readability (how easy it is to understand the submitted code)**:
  - Does it have meaningful variable names/functions?
  - Does it have good code comments?
  - Is the code well-formatted?
- ☐ **Functionality (Produced output/answer)**:
  - How well each feature is implemented?
  - Is the solution implementing the features as described in this write-up?
  - Is the code free from runtime errors?
- ☐ **Artifacts' Completeness & Quality**
  - Does the submission include all the required artifacts listed in the "Deliverables" section?
  - How well-written are they?
  - Is the report covering all the technical aspects described in the template?
  - Is the project easy to install & setup? Or was there any step missing in the project's description, which made the submission hard to execute?

If the project has runtime errors which prevents the TAs from running the project and testing it, the project's grade will be zero.