Project (100 points)

This project shall be done in a group of **2-4** 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.*

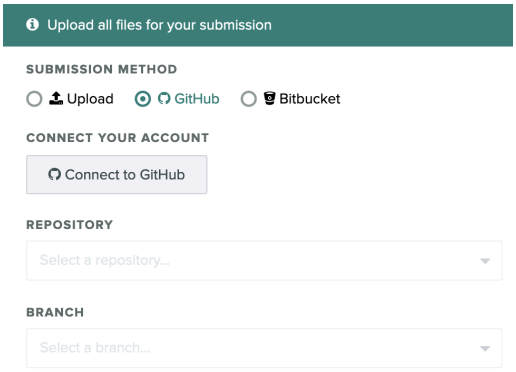
If you haven't already done so, please indicate the group members in the spreadsheet posted on EdStem by Mar 22, 2024.

The project will be a Web application developed using Python **and** Django. If your project need dependencies, then specify them on a `requirements.txt` file. Notice, however, you can only use external libraries that are publicly available and can be installed using `pip install`. You cannot use proprietary libraries or libraries that require a specific operating system. Your code will be tested on different platforms (Windows, MacOS, Linux), so no library should require a specific OS.

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

How to Submit

- Create a **private** GitHub repository and share it with the **professor** and the **TAs'** GitHub accounts, that are linked below:
    - Prof Santos
    - Christian2358 (Christian Northrup) · GitHub
    - Robertdebrus (Robert Wallace) · GitHub
    - TTangNingzhi (Ningzhi Tang) · GitHub
- Add your group's GitHub URL on the spreadsheet (Column A): SP24 Paradigms Course Project - Groups breakdown
- 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.
    - The file `.gitignore` should be placed on the root of your GitHub repository (and should be named exactly as it is).

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 (except for the Demo video, that deliverable will be uploaded on a shared panopto folder). 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 (30 points)

Deadline: Apr 12, 2024 11:59 PM

In the first phase, you will select ***at least three (sub-)features*** from the Project Description section to implement ***(ex: Feature 1.1, 2.1 and 3.2)***. 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 which is available in the course's GitHub repository).
- `./src`: all the code, configuration files, etc., created so far.

Both deliverables listed above should be uploaded on Gradescope and GitHub by the deadline. **Only one person submit on Gradescope, and this person add the names of their team members.**

Phase 2 Deliverables (70 points)

Deadline: May 1, 2024 11:59 PM

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

- `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.

You will also upload a video on **Panopto**:

- `A demo video`: Upload on Panopto a recording (***up to 5min long***) that provides a demonstration of the features for your project (ie, you will record a walkthrough of a user using the features of the web application).

Make sure that all deliverables are in your private GitHub repository & Gradescope and that the video is uploaded on Panopto!

Technical Constraints

- The web application should *not* assume the inputs are well-formed (i.e., the application should validate the inputs).
- The information has to be saved in 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 players, each of which should have played the game at least 10 times using at least 2 different languages.
- The web application should have at least 2 web pages (i.e., it cannot have a single web page with all the features listed below).
- The web application should have at least one CSS file to stylize the website.
- The backend must be implemented using Python and Django.
    - Your implementation is allowed to use public external libraries to the backend (but remember to specify these libraries and their versions on a requirements.txt file).
- The front-end should be implemented using HTML/CSS/JavaScript.
    - Your implementation is allowed to use public external JS scripts and CSS stylesheets.
- 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 data. Instead, your code would make use of generic views (DetailView, ListView, CreateView, etc) or view functions + 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 gaming web applications. The product we envision is called WORNDLY:

At its core, WORNDLY is meant to be a fun game where users get 6 chances to guess a 5-letter word. Each guess provides feedback in the form of color-coded clues, guiding players towards the correct answer. The game combines vocabulary skills with logical deduction, offering a daily challenge that has captured the interest of word enthusiasts around the world.

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

Feature 1: Sign Up / Sign in

1.1: Create new User Profile

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

- Name *
- username

- email
- password

1.2: Log-in

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

1.3: Log-out

The program shall allow a user to log-out.

Feature 2: Gameplay

Once a player is logged in, they have access to the game play.

2.1 Play Game

A player shall be able to play the game whose main goal is to guess a 5-letter word in 6 tries. A user can only play **up to 3 times a day**. If they want to play more than that, they need to pay for each play. See *Feature 4*.

1- The player will first select the desired language:

- 🇩🇪German (de.txt)
- 🇪🇸Spanish (es.txt)
- 🇫🇷French (fr.txt)
- 🇧🇷Portuguese (pt.txt)
- 🇺🇸English (en.txt)

2- Once the user selects the language, this game will randomly pick one of the words from the corresponding language file – as indicated in parentheses above. The files with the words are in the course repository. (location: Project/words/*.txt). Notice that each file has one word per line (each line is divided with a `\n` character - you can download the raw file from GitHub to make sure it has proper newlines).

*To help the TAs to grade, your application should print to the **front-end (JavaScript)** console the chosen word.*

3- Once a word is randomly selected, the game will start. Players have up to **six** attempts to guess the chosen five-letter word. At each turn, the application should provide visual feedback given for each guess in the form of colored tiles indicating when letters match or occupy the correct position. Similar to as shown below, use green for correct letters in the correct spot. Use yellow for a letter that is in the word, but in the wrong position. Use gray for a letter that is not in the word to be guessed.

The letter **W** is in the word and in the correct spot.

The letter **I** is in the word but in the wrong spot.

The letter **U** is not in the word in any spot.

Notice:
- You don't have to make the UI exactly as shown below, but your application should have a way to provide feedback to the user on what letters are correct/incorrect. However, the application must use a color-coding scheme to give feedback (ie., a *green-ish*, *yellow-ish* color, and *gray-ish* color). You are free to pick what shade of gray/yellow/green to use.
- Remember to validate inputs! If the user inputs an invalid word, then your application should give feedback to the user and ask the user to provide it again (this problematic input should **NOT** count towards the maximum number of attempts). A word is invalid if it is a word not in `words.txt`.
- Once the user provides its valid 5-letter word, the program will compare it with the chosen word. If the guess is correct, the application will provide feedback to the user, congratulating them for correctly guessing the word after X attempts.
- Keep in mind that Feature 3 is related to a dashboard. Hence, you will need to keep a history of previous game plays for each player to be able to compute basic stats.
- When the user reaches their maximum daily quota of free games they can play, they should be prompted to buy more games. See Feature 4.

Feature 3: Player's Dashboard

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

- View all the plays they have done in the past
- Basic statistics

3.1 View all prior plays

The player shall be able to view ***all*** of its previously plays, and be able to filter the list of previous plays such that they can view:

- Plays made last week
- Plays made last month
- Plays made last year
- All previous plays

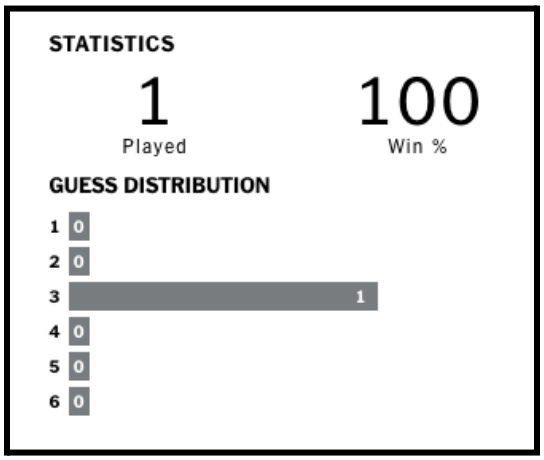
While listing all the previous plays, the following information should be displayed:
- Game Date: when the game was played
- Pass/Fail: indicating whether the user was able to correctly guess or not the word
- Number of Attempts: how many attempts the player took.

3.2 Viewing basic statistics

The player shall be able to see the following statistics about their previous plays:

- How many times they played the game
- How frequently the player won (i.e., guessed the correct word)
- The distribution of number of attempts per game (see screenshot below)

For example, the distribution chart below shows that the player played only once and was able to guess the word correctly in 3 attempts:



***Hint**: you can use Charts.js to show this type of chart. A demonstrative example is in the repository (see file: chart_example.html).*
https://jsfiddle.net/p2nd6v5z/

Feature 4: Buying Game Plays

In this website, users can buy extra game plays per day using our fake currency: Krato$Coin.

4.1 Buying more games

The website shall enable users to purchase additional game plays per day. The cost is **1 Krato$Coin = 1 extra game play a day**.
During the checkout process:
1. The user is presented with their current balance.
2. The user specifies how many games they'd like to purchase, and they will get that charged accordingly.
3. The web application will check whether the transaction succeeded or not, based on the response given by an external REST API.

- 3.1 If the transaction succeeds, the user is provided with a message saying that they have successfully purchased the games and can now play an extra N games. The games accumulate overtime (ie., they should not expire after 24 hours).
- 3.2 If the transaction fails, the user is provided with the corresponding error message (insufficient funds, etc).

Notice:

- To implement this feature, **your web application will make a request to a REST API to charge the user**. Each group was granted access to the REST API through an authentication token (each of you had received username/password that is needed to get your authentication token). The API documentation details are shared below.
- Each group will be able to access the REST server via the admin interface to ***manually*** add credits to users in their game.
- This external REST API identifies users based on their email addresses.

Accessing the REST API

Step 1: Get your access token

You need to get your access token by making a POST request to the /api/token/ endpoint. You can make this POST request using the CURL command:

```python
curl -X POST -H "Content-Type: application/json" -d '{"username": "your username", "password": "your password"}' https://jcssantos.pythonanywhere.com/api/token/
```

Your username and password were provided via e-mail on March 29th. If your username and password are correct, you will receive a response like this:

```
Python
{"refresh":"a-refresh-token","access":"an-access-token"}
```

The token you will use is the one within the `"access"` attribute.

Step 2: Make your requests to the REST API endpoints

The API has the following endpoints:

HTTP Method	Expected behavior
GET	`GET /` - Returns a list of balances per user. `GET /player/email/` - view the balance for a specific player given its email address
POST	`POST /player/email/pay/` - Decrease N coins from the user's balance. The data is passed as JSON in the body of the HTTP request: { "amount": value}. The value has to be a positive integer number.

These endpoints accept only ***authenticated*** requests. Thus, when you make your requests, you need to make sure you pass in your access token.
Moreover, each group has its own dedicated API endpoints. **Thus, you must prefix your requests with the following URL:**

`https://jcssantos.pythonanywhere.com/api/groupN/groupN`

You need to replace `groupN` by your group number (ex: group0). If you try to access another group's endpoint, you will receive a **403** error.

Below are sample Python code snippets showing each endpoint for `group0` (**but remember to replace by your actual group number**):

– Viewing all users' balances:

```
Python
def view_all_coins(access_token):
   # Use the access token to make an authenticated request
   headers = {
       'Authorization': f'Bearer {access_token}'
   }

   # Make a GET request with the authorization header
   api_response =
requests.get("https://jcssantos.pythonanywhere.com/api/group0/group0/",
headers=headers)

   if api_response.status_code == 200:
```

```
        # Process the data from the API
        return api_response.json()
    else:
        print("Failed to access the API endpoint to view all coins:",
api_response.status_code)
```

The request above will return a JSON file that looks like as follows: `[{'player': 'jdasilv2@nd.edu', 'amount': 10}]`

– Viewing the balance for a specific user given the user's email address (ex: jdasilv2@nd.edu)

Python

```
def view_balance_for_user(access_token, email):
    # Use the access token to make an authenticated request
    headers = {
        'Authorization': f'Bearer {access_token}'
    }

    # Make a GET request with the authorization header
    api_response =
requests.get(f"https://jcssantos.pythonanywhere.com/api/group0/group0/pl
ayer/{email}/", headers=headers)

    if api_response.status_code == 200:
        # Process the data from the API
        return api_response.json()
    else:
        print("Failed to access the API endpoint to view balance for
user:", api_response.status_code)
```

The request above will return a JSON file that looks like as follows: `{'amount': 10}`

– Subtracting an amount from a user's balance (used when the user is buying more games):

Python

```python
def user_pay(access_token, email, amount):
   # Use the access token to make an authenticated request
   headers = {
       'Authorization': f'Bearer {access_token}'
   }
   data = {"amount": amount} # non-negative integer value to be
decreased
   # Make a POST request with the authorization header and data payload
   api_response =
requests.post(f"https://jcssantos.pythonanywhere.com/api/group0/group0/p
layer/{email}/pay", headers=headers, data=data)

   if api_response.status_code == 200:
       # Process the data from the API
       return api_response.json()
   else:
       print("Failed to access the API endpoint to pay:",
api_response.status_code)
```

The request above will return a JSON file that looks like as follows: `{'message': 'Coins decreased successfully', 'new_amount': 9}`

All groups have their database initialized with one player (jdasilv2@nd.edu) that has 10 coins. You can go to https://jcssantos.pythonanywhere.com/admin/ and use the admin interface to add more players and coins for each of these players. You will log-in using the username/password sent to you via e-mail on March.

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.