Seneca | FACULTY OF CONTINUING EDUCATION AND TRAINING

SENECA WEB PROGRAMMING PROGRAM
FALL 2018 SESSION

Module Number: WEB 302
Assignment #2 Part 2

Instructor: Tim Lai
Title: Django Application
Assigned: Tuesday July 20
Due date: 11:59pm Tuesday August 10
Value: 40%

Submission details: This assignment MUST be submitted in at least one of 2 ways, as specified below. If the assignment does not follow the specified instructions, it will be considered incomplete. *Failure to upload this assignment will result in a grade of zero*.

1. This assignment should be uploaded to a **GitHub repository** which I will invite you to. When I download this GitHub repository I must be able to access all your source files so that I can see your Python code, HTML/Django template code and connect your site to a database.
2. If you have difficulty uploading to GitHub, then you may alternatively upload your assignment as a ZIP file to **MS Teams** in your **personal channel's Files tab**, or an online cloud storage site such as **Google Drive** or **WeTransfer** and send me a link on **MS Teams** in your **personal channel's Posts tab**. *DO NOT EMAIL ME A ZIP FILE AS AN ATTACHMENT. I WILL NOT RECEIVE IT.*

Instructions: You will be creating a fully-functional **MVC CRUD application** using the **Django** framework. *DO NOT USE A DIFFERENT PYTHON FRAMEWORK SUCH AS FLASK*. The application will allow users to create, read, update and delete **items** which can belong to **models of your choice** (i.e. Spaceship, Dog, Clown etc.) *as long as they are NOT a Musician, Album or Song – be creative*. This application will store the items in a **PostgreSQL database** using the **Django ORM** and **migrations**. You should include a **requirements.txt** file which lists any PIP packages which your project depends on.

Your application must make use of **at least 2 models**. Your models must define at least **1 relationship** which is either a **one-to-many** or **many-to-many** relationship. Your primary model should define at least **3 fillable fields** – a **name**, an **image** and something **unique to your model**. Any strings must be **filtered and validated** using Django's model field validation methods and the **clean()** method. Names should be validated using **CharField()** and formatted with **capital first letters** with the **rest lowercase**. Image files should use **Pillow** and **ImageField()**.

The application should start on a **View Models** page which lists the **names** of all the items which have been added to the database for one of your models. Each item should also have a link or button to a page where you can see more information about that individual item, including

relational data which comes from another model. The user should also be able to view an **image** which is associated with each item, but this could be displayed on either the home page or the individual item's page. There should be a clear **navigation item** or **button** that links to an **Add Model** page which displays a **form** where users can enter **properties** for a new object, including a **name** and an **image**. If the user does not fill out all the required fields, or there is an error sending data to the database, then they should be presented with an **error message** and should not be able to proceed. If the user correctly fills out all the required fields and the data gets sent to the database then they should be taken back to the View Models page where they are presented with a **success message** and can see all the objects in the database, including the one they just added. Each item displayed on the View Models page should have an **Edit button** and a **Delete button** near it. When the Edit button is clicked, the user should be taken to an **Edit Model** page where there is a **form** which is **already filled out** with that **object's attributes**. If the user edits any of the attributes, then they should be taken back to the View Models page where the changes take effect. If the user leaves one of the required fields blank, they should be presented with an error message and should not be able to proceed. When the Delete button is clicked the selected object should be **deleted** from the database. If the object is successfully deleted, they should be presented with a success message. If there is an error, they should be presented with an error message. *I MUST be able to add objects using the form on the Add Model page, edit objects using the form on the Edit Model page, and delete objects using the Deletion buttons.*

Generic class-based views should be used for to transfer data from the primary model to the templates and from the templates to the model using distinct **ListView**, **DetailView**, **CreateView**, **UpdateView**, and **DeleteView** classes. The views for your secondary models do not need to use all the premade class-based views but should make use of **CreateView**, **UpdateView** and **DeleteView** at a minimum. You may display the data for your secondary models on the same pages as your primary model.

The application should have an intuitive and functional **user interface** which uses **Django templates**. The Django templates should make use of a base template and **sections**. They should also make use of Django template **loops** and **conditionals** to display data. Styling can be done using either regular CSS or Sass. It should make use of **buttons**, **alerts** and **colour-coding**. Accessibility considerations should be made for non-standard users. Responsiveness and use of libraries such as Bootstrap, Foundation and Font Awesome is NOT required but is strongly encouraged.

Deliverables:

- At least **2 Django apps**: 1 project-wide app and 1 other app which manages your primary model
- At least 2 models with at least **3 fillable fields** including a **name**, an **image** and something **unique to your model** and at least **1 one-to-many or many-to-many relationship**
- At least 3 Django page types per model: a **View Models** page, an **Add Models** page and an **Edit Models** page (you should rename the pages to match your model i.e. Add Dog)
- A common **base template** created using **Django templates** and **sections**
- An external CSS file and optionally a SASS file
- A functional and intuitive **UI/UX design** which makes use of **buttons**, **alerts** and **colour-coding** (use of Bootstrap/Foundation/Font Awesome is encouraged)
- **Add forms** on the **Add Models** pages which take in, validate and sanitize **model field** data
- N**ew items** which are created, saved to a **PostgreSQL database**, retrieved, and displayed on the **View Models** page every time the **Add Model** forms is submitted
- A **button or link** for each item on the View Models page which allows the user to view more information about an individual item, including **relational data** from a second model

- **Edit forms** on the **Edit Model** pages which update existing **model field** data
- **Delete buttons** for each item on the **View Objects** page which deletes an item when clicked
- **Class-based views** with distinct **ListView**, **DetailView**, **CreateView**, **UpdateView**, and **DeleteView** classes
- A **requirements.txt** file which lists any required PIP packages
- Accessibility considerations for non-standard users (blind, assistive devices, etc.)

Grading breakdown:

- Creation of 2 models which use a Django ORM relationship, sanitization, form classes **(30%)**
- Creation of Django templates which retrieve and display data **(25%)**
- Creation of class-based views which perform CRUD operations with corresponding URLs **(30%)**
- Creation of a usable UI/UX design using HTML, CSS and Django templating **(10%)**
- Good coding/file organization practices (variable names, code organization etc.) **(5%)**

Your files should contain only valid (as determined by the W3C validator) HTML code. You can use invalid code in the HTML but if you do, you must include a comment beside the invalid code explaining your decision. To properly validate your HTML in your Django templates, you MUST do a "view source" of the page in a web browser. The W3C validator does not understand Django template code. *You will lose 2% per type of uncommented validation error*. You should not have any Python errors. *You will lose 2% per Python error which could have been fixed*.

Late Policy

All late assignments will be given a grade of zero.

Plagiarism

There are serious penalties for cheating and plagiarism offences and you are expected to be aware of our Academic Honesty Policy. Please refer to the Academic Policy at http://www.senecacollege.ca/ academic-policy/acpol-09.html for more information.