CA377 Programming Fundamentals (Project)

Name	Shane Tully
Student Number	16468314
Programme	EC3
Module Code	CA377
Assignment Title	Programming Fundamentals (Project)
Submission date	2018/12/14
Module coordinator	Suzanne Little & Marija Bezbradica

I declare that this material, which I now submit for assessment, is entirely my own work and has not been taken from the work of others, save and to the extent that such work has been cited and acknowledged within the text of my work. I understand that plagiarism, collusion, and copying are grave and serious offences in the university and accept the penalties that would be imposed should I engage in plagiarism, collusion or copying. I have read and understood the Assignment Regulations set out in the module documentation. I have identified and included the source of all facts, ideas, opinions, and viewpoints of others in the assignment references. Direct quotations from books, journal articles, internet sources, module text, or any other source whatsoever are acknowledged and the source cited are identified in the assignment references. This assignment, or any part of it, has not been previously submitted by me or any other person for assessment on this or any other course of study.

I have read and understood the referencing guidelines found recommended in the assignment guidelines.

Name: Shane Tully Date: 2018/12/10

Introduction

EatAtDCU is a website displaying all the places to eat across all of Dublin City University's ("DCU") campuses. These include locations St. Patrick's College, Glasnevin, All Hallows and DCU Alpha. Each location will display and give information on the cafe and restaurants available. This information may include opening times, weekend hours, and a location. Restaurants also feature an additional page showing what their daily special is.

This website was created using the Django web framework. This framework allowed each page to be displayed using a Hypertext Markup Language ("HTML") template, styled using Cascading Style Sheets ("CSS") and additional functionality using Javascript. Python was also used as Django is Python-based.

This report will give an overview of the entire project, including: the system architecture, the user interface design, how Gitlab was used, and the additional functionalities. The goal of this report is to clearly outline the formation of the website and how it was created, and being able to reflect if anything would be done differently.

System Architecture

The website was made using the Python-based web-framework called Django. Django uses the Model-View-Controller ("MVC") as an architectural pattern. "Model" is Django interpreting the classes and objects of a database. The "view" part of this is looking at what data gets presented to the user, such as rendering HTML templates or JSON dictionaries. Controller is the preconfigured settings that are integrated in the Django framework, such as "urls.py". An alternative naming for the architectural could be "Model-Template-View", as "controller" can be considered a redundant part of the acronym[1].

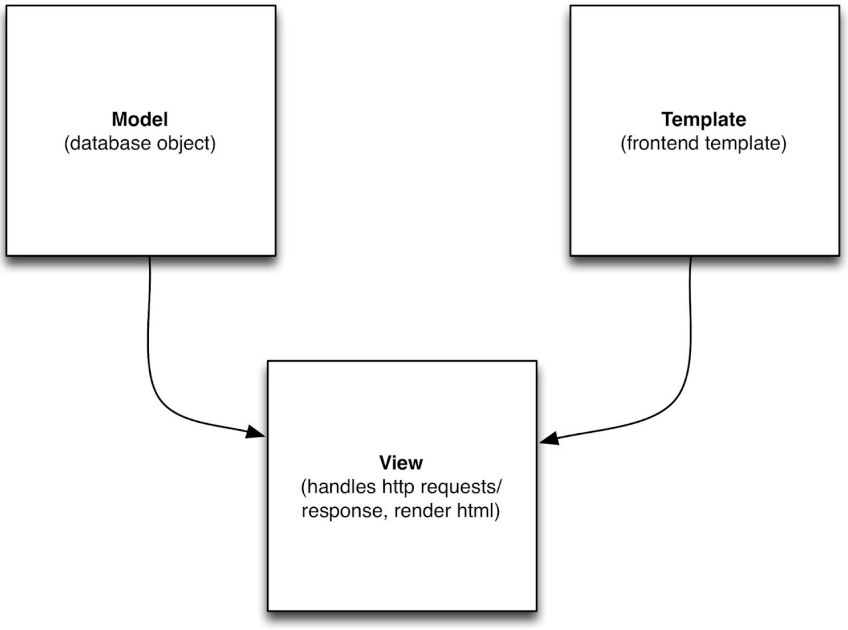


Image from Cornell University [2].

An example of this in action is a database of customers in a csv file. Models.py will read the database and take each entry of the database and assign them to attributes in a class called “Customer”. An attribute could be “customer_id = models.IntegerField(primary_key=True)”. The class “Customer” can now be passed in to “views.py”. Template is a HTML file of how the data from the Model will be portrayed. The template should contain as little information as possible and rely on portraying from data passed from Models. The data on this page would be variables from the Customer model. Both the Template and Model are passed to the View. From views.py the Customers.html template can be rendered and Customer data can be passed through from the imported Models.

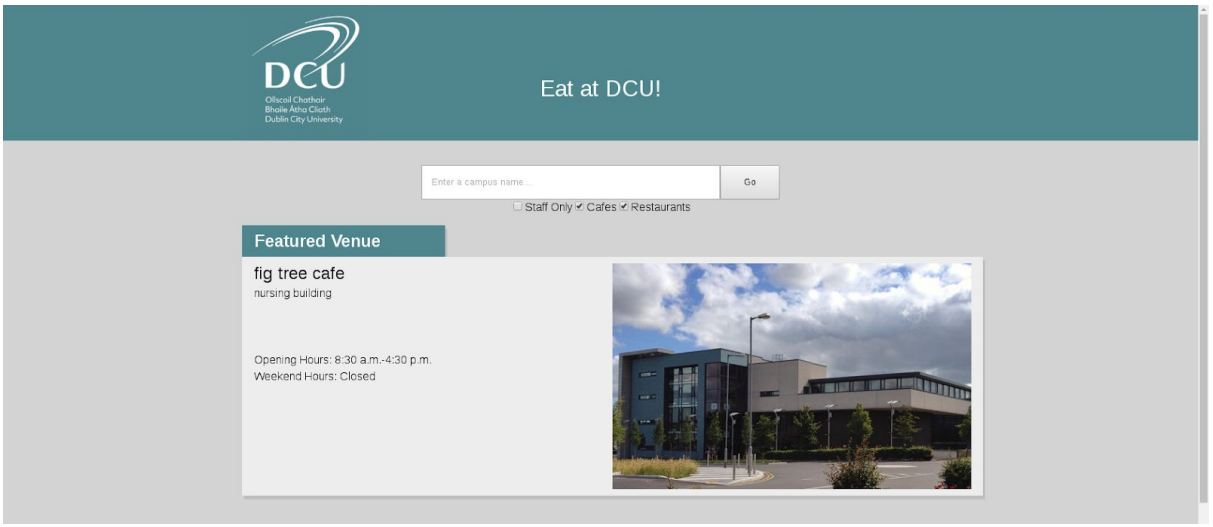
As the templates are made using HTML the styling is done with CSS. The CSS is styled to each element in the HTML through the “ ” tags. The CSS was designed to keep it as consistent across different browsers. An example of this is using percentages of the field of view to measure sizes of elements. Creating consistency across all browsers can be hard to achieve, and the styling may appear incorrect. To combat this alternative CSS styling could be created for different view sizes such as mobile. This can be tedious to do.

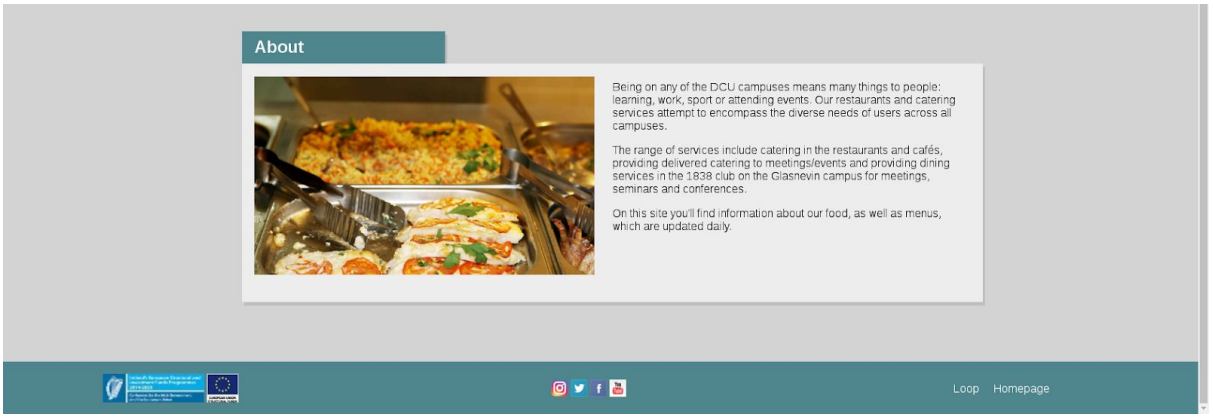
Due to the way Django is set up it is easy to scale up as pages are generated dynamically based on the database contents. Pages are generated when they are needed. Instead of using templates the project could create a HTML page for every venue, however this would require a considerable amount of time. A time this would be convenient would be if more restaurants needed to be added. The website administrator would only have to add the relevant information to the database [3].

Continuing with longevity in mind, Django being Python-based is an advantage as Python’s popularity continues to grow. According to 2018 Redmonk Programming Rankings, Python is ranked the fourth most popular language. These rankings are based on the number of GitHub pull requests and the number of tags for posts on StackOverflow. It is also featured in 70% of introduction to programming classes in American Universities as of 2014 [4]. This means the websites framework is not likely to become outdated and discontinued any time soon.

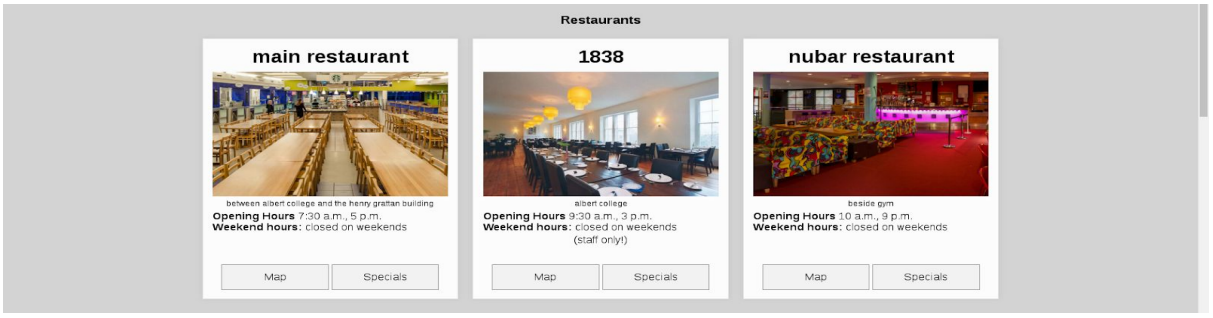
JavaScript Object Notation (“JSON”) was used to generate a dictionary by calling a webservice on pythonanywhere.com for the restaurant specials. This was used by importing it into the Python views.py file.

UI Description

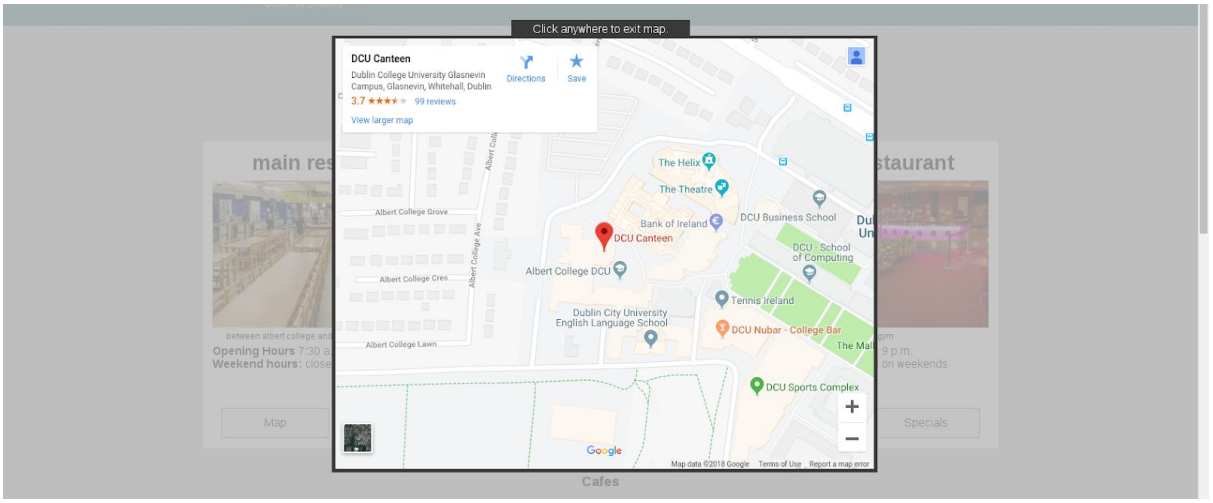




The user goes on to the index page and they are greeted by a header stating “Eat At DCU” along with DCU branding. Below this there is a search bar with a placeholder “Enter campus…” to guide the user to search for a campus. The search bar includes checkboxes “cafes” and “restaurants” that are ticked by default, and “staff only” that is unchecked. This is where the header ends. Underneath this is a box featuring information on a featured venue. This venue is randomised each time the index page is rendered. This box includes all the venue information clearly laid out as well as a picture. Underneath the featured box is an “About” box that gives a description of the food options at DCU. At the bottom of the page is a footer with European Union disclaimers, social media links, and DCU relevant links. The header with the search bar and the footer are featured on every page.

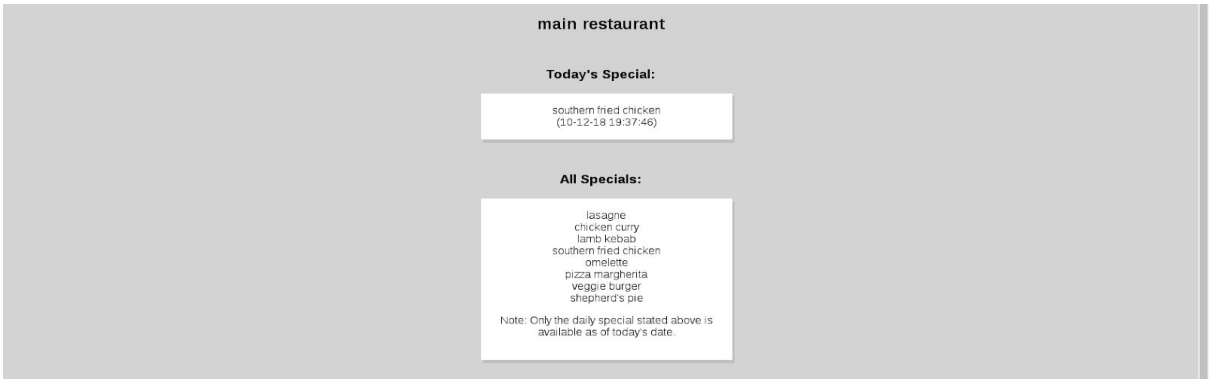


When the user enters a valid campus in to the search bar they are redirected to a page where the query results fade in. The placeholder in the search bar is replaced with what the search query is. There is a category of cafes and restaurants depending on what is requested. The venues are broken up into “location cards” so each venue is side by side in a card design. The location card includes the venue name, picture, location, opening times, weekend times, and whether it is staff only or not. The location card design allows each venue to be easily distinguished and the cards are designed in such a way that the information is spaced out and readable.



Each location card has a button to generate a map using Google Maps API, having it appear as a popup. When the map pops up the whole page is greyed out except for the map which is in the center of the screen. There is a small bit of text above the map that instructs the user to click anywhere except the map to exit the map. The map can be fully interacted with.

If the venue has specials available then a button appears beside the maps button that redirects to a specials.html page relevant to the venue. These two buttons are wide so they stand out and are easy to click.



If the user clicks the specials button they are redirected to a new page that has two boxes with the title of the venue at the top. The first box states the daily special available and the current date. The second box shows all the available specials normally. There is a disclaimer under all the specials saying that only one of these is available.

The style of the website followed the official DCU branding guidelines. This includes the colour, fonts and logos, as well as general design. Arial was used as a font as it is close to the actual font DCU uses called “Objectiv Mk 2”, which requires a paid license [5].

Being user friendly was constantly kept in mind for this project, therefore all the error messages are easy to understand and give clear instruction as to where the user went wrong, example: “Error: you must enter a valid DCU campus (example: St Pats, DCU Alpha)”.

Using Source Code Repository (gitlab)

Gitlab was used for this project by forking from EatAtDCU repository created by the lecturers. This allowed the base code for each assignment and the started code to be pulled from the master repository. Once the repository was forked into the project repository, the project repository could be cloned anywhere. The files in the project repository would automatically update when the master repository was updated. Every time something needed to be added to the project repository, the changes were added using “add” and “commit” to push it to the repository from the local machine.

There were a few issues when working with the project repository. This mainly came down to a lack of experience using a repository that is forked from a master repository.

One week a commit was added that was supposed to contain working code for an assignment. On the Gitlab website it showed the commit was submitted. However, the contents of the commit were never checked. For an unknown reason the changes made were not part of the commit. Despite this, the finished code was found by looking at previous versions of the git files in terminal. As a result of this error, half of the marks for this assignment was lost. In order to avoid this the contents of each commit should be checked, as a commit can be added that contains almost nothing. This is not much of an issue on a small scale project, however on larger projects with very strict deadlines and many contributors, the work must be committed in full correctly.

On another occasion there was a commit published called “A4 finished” followed by a commit called “A4”. The first commit contained the finished code for the assignment, whereas the second commit only contained some migration messages. As a result of this only the second and most recently pushed commit was graded at 2/15. When the second commit was pointed out the grade was changed to 14/15, losing 1 mark as a result of this error. This error could have been caused by unnecessarily pushing when there was nothing substantial that needed to be pushed.

Sometimes when attempting to push changes made locally to the repository, there was an error saying the head of the repository was ahead of the local git. In order to combat this and force the commit to push the local files were pushed to overwrite the files in the repository. This is a bad practice however, as with multiple contributors on a repository then the local files may not be the most up-to-date, and overwriting could remove some recently added work.

Overall, pushing commits to the repository could have been handled more carefully in order to insure they were correctly done. It is not as much of an issue on small projects, however on large project repositories with many contributors it can be a serious issue. The commit messages also could have been more clear as to what has changed, as not everyone will be able to understand some of the vague and shorthand commit messages. In a large project this could lead to a communication error.

Additional Functionality (for +7.5%) [1 page]

Search Checkboxes

Firstly, there was the addition of the “staff only”, “cafes” and “restaurants” checkboxes to refine the query results. “Staff only” was unchecked by default. “Cafes” and “Restaurants” were checked by default. Unchecking either of these means they will not be generated in the query. This was done by adding three new inputs in the form of a checkbox inside the “<form>” that is submitted when the restaurants.html template is being generated. An example of what this changes the URL and queryset to is: “?campus=Glasnevin&staff-only=true&cafe-checkbox=yes”. This means the “staff only” checkbox was ticked to be true, and the “cafes” checkbox was left ticked to true. Therefore only the cafes that are staff only in “Glasnevin” will be returned.

Checking the status of the checkboxes was done in views.py. Before the query is refined the following is checked: “if "staff-only" in request.GET:”. This checks if “staff-only” is in the request (i.e. the URL). If it is, then queries will be filtered using “is_staff_only=1” so only staff only results are returned, and “is_staff_only=0” if it is not. After this it is checked if both "restaurant-checkbox" and “cafe-checkbox” are in the request. If so it generates both cafes and restaurants. If they are not both in the request then it checks if “cafe-checkbox” is present, then it checks if “restaurant-checkbox” is present. This would allow the output to work if only one of the two is requested. The “else” statement is if neither the cafe or restaurant box is checked. When the cafe and/or restaurant is not checked the variable for either is set to “None”. If the box is checked the variable is set to the Restaurant object and filtered accordingly.

Featured Venue

Another additional feature is the “Featured Venue” box on the index page. This is done by adding one line of code to views.py and passes it through when rendering the index.html template:

“featured_restaurant=Restaurant.objects.filter(restaurant_id=randint(1,len(Restaurant.objects.all())))”.

This generates one object from the Restaurants class based on the “restaurant_id”, a unique number assigned to each venue. The ID of the restaurant is assigned by generating a random number between 1 and the number of objects in the Restaurant class”. An alternative would be generating a number between 1 and 12, as there is only 12 venues. However, this means if new venues are added then views.py would have to be manually edited, whereas len(Restaurant.objects.all()) will dynamically update.

Google Maps

On each location card generated there is a button called “Map”. When this is pressed a Javascript function is called that will allow a CSS popup to appear. In the popup a HTML iframe tag of the Google Maps API of the selected location appears. To generate the map for each location the iframe uses “src="{{location_url}}"” as the source. The “location_url” was a variable set as the “map_location” Restaurant object variable. To do this another variable was added to the restaurant.csv with a link to the Google Maps location. This URL was generated by finding the location through a browser and pressing “Share”.

Miscellaneous Additions

- Each location card contains an image of the venue. This is done by naming the image in the static file the exact same as the venue.
- A fade transition effect when the restaurants.html page is loaded showing all the queries. Done through CSS “@keyframes” tag.
- Specials page shows the current special, as well as a list of all the specials that may be available. This is achieved by generating another JSON dictionary and passing it with the original specials dictionary.
- Fixed footer at the bottom of the page linking to relevant links to DCU, such as loop and social media links.
- An about box on the index page with some information on food venues at DCU.

Images included in UI description.

Conclusion

The website has been created to be changed dynamically and can scale easily. The CSS is laid out in a simple format that makes the information distuingable and visually appealing. The templates for each page include no hard-coded data. While the website was originally useful, the additional functionality makes the website much easier to use and overall user experience is improved. Despite this the website is not overly complicated and easier to use for any user. Compared to the DCU websites the branding follows the same theme.

This project taught me how beneficial Django and its “MTV” architecture is. If I was to create this website last semester I would have hard-coded a HTML page for each restaurant and not used any database. I now understand how extremely inefficient this is, and how this can only work on small scale projects. I am heavily considering restructuring my personal portfolio website (www.shaneapt.com) to use Django, as I could have each project page use a projects.html template. Currently I make a new .html file for each project, which is not the best method.

Creating my additional functionalities taught me Django syntax that I would not have learned otherwise, such as using “with” or combining two strings together. I felt it was not hard to learn this new syntax as the theory behind it I understood.

I also learnt the importance of being careful when using Git. I made mistakes that greatly affected my grade on parts of the assignment I felt I did not struggle on. In the future I will be more wary of what my commits to Gitlab contain.

References

[1]: https://docs.djangoproject.com/en/dev/faq/general/#django-appears-to-be-a-mvc-framework-but-you-call-the-controller-the-view-and-the-view-the-template-how-come-you-don-t-use-the-standard-names
[2]: https://courses.ece.cornell.edu/ece5990/ECE5990_Fall15_FinalProjects/Victor_Fei/website/index
[3]: https://hackernoon.com/advantages-and-disadvantages-of-django-499b1e20a2c5
[4]: https://www.techrepublic.com/article/fastest-growing-programming-language-pythons-popularity-is-still-climbing/
[5]: https://www.dcu.ie/sites/default/files/marketing/digitalmedia/overview/index.html