GE04 Documentation

**Jacob Hartt, Takaiya Jones, Jacob Biles and Zachary Rytting**

Professor Deb Harding  
CS3300.002  
03.19.2024

[**SECTION 4.1 BOOTSTRAP**](#_SECTION_4.1_BOOTSTRAP)

[Issues Encountered](#_Issues_Encountered)

[Useful Resources](#_Useful_Resources)

[SECTION **4.2 CSS STYLING**](#_SECTION_4.2_CSS)

[Issues Encountered](#_Issues_Encountered_1)

[Useful Resources](#_Useful_Resources_1)

[SECTION **4.3 THE HOMEPAGE**](#_SECTION_4.3_THE)

[Issues Encountered](#_Issues_Encountered_3)

[Useful Resources](#_Useful_Resources_2)

[SECTION **4.4 PORTFOLIO DETAIL PAGE**](#_SECTION_4.4_PORTFOLIO)

[Issues Encountered](#_Issues_Encountered_4)

[Useful Resources](#_Useful_Resources_3)

[SECTION **4.5 FORMS**](#_SECTION_4.5_FORMS)

[Issues Encountered](#_Issues_Encountered_5)

[Useful Resources](#_Useful_Resources_4)

[SECTION **4.6 DISPLAY STUDENT DETAILS**](#_SECTION_4.6_DISPLAY)

[Issues Encountered](#_Issues_Encountered_6)

[Useful Resources](#_Useful_Resources_5)

[SECTION **4.7 STUDENT LIST VIEW**](#_SECTION_4.7_STUDENT)

[Issues Encountered](#_Issues_Encountered_7)

[Useful Resources](#_Useful_Resources_6)

[SECTION **4.8 DISPLAY PORTFOLIO DETAILS**](#_SECTION_4.8_STUDENT)

[Issues Encountered](#_Issues_Encountered_8)

[Useful Resources](#_Useful_Resources_7)

# **SECTION 4.1 BOOTSTRAP**

**General description:**

Bootstrap is already integrated into the project in the base template header.  No further action is needed.

## **Issues Encountered**

* **No Issues Were Encountered**

## **Useful Resources**

* None.

# **SECTION 4.2 CSS STYLING**

**General description:**

In this section, developers design the portfolio view page. They can build this webpage with the built-in Django detail views and reusing code from the homepage. Using the detail view, Django automatically adds the fields of a related object to the view's context, streamlining the passing of object data to the template. Next, we reuse code from the previous page to display a list of related projects. The trickiest part of this section is learning how to configure URLs with two object IDs so that project update, view, and deletion pages can be linked from the portfolio view.

## Issues Encountered

**General description:**

CSS integration is much the same as Bootstrap.  The only difference is that CSS integrates through a static local link in the header, while Bootstrap integrates through a remote download link.

A computer screen shot of text

Description automatically generated

## **Useful Resources**

* Resource 1: [Django Static CSS Video](https://www.youtube.com/watch?v=GfyP_MYtLng&t=478s&pp=ygUKZGphbmdvIGNzcw%3D%3D)
  + This resource shows how to integrate static CSS files into django.

# **SECTION 4.3 THE HOMEPAGE**

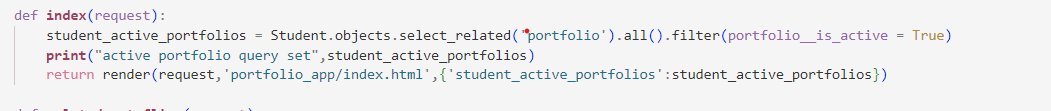
The homepage is the first display seen by users when coming to a site. Homepages are an entryway to expectations, it is important to build and comprehend the details of your site.

## **Issues Encountered**

* Issue 1: The homepage utilizes different aspects of software concepts and principles to understand models, views, HTML styling, and other components crucial for sites. Before you even start designing and setting displays on your screen, you need to understand the relationship between different areas in your code. The models created earlier contain certain characteristics that describe what it is. Your models are needed when referring to the “views.py.” Functions inside of “views.py” create variables that search for specific requirements inside of the models listed earlier. The function then returns the item searched and directs this response to the HTML template. The HTML template is like a skeleton of what the website will display.

* Solution 1: Take time to understand the code you produce before moving on to the next topic. When creating a model, we establish characteristics with needs. Each model contains needs.
* A screenshot of a computer program

  Description automatically generated
* Afterward, we have a view section that searches through the database for required information and saves that information. The information is sent to an HTML link, which will display the required information from active students.



* The “ursl.py” contains all the paths to where the user is led. The “urls.py” and “views” work together to establish routing. Without the URL paths, would not be able to navigate through the site. Be sure when a new page is created to link it in  “urls.py”.A screen shot of a computer

  Description automatically generated

* The HTML document customizes the appearance and displays the required data, active students, on the homepage. A “for loop” iterates across the active student and matches correlating data.

A computer code with text

Description automatically generated with medium confidence

* The last step is the creation of the button. The for loop gets the portfolio specified portfolio and links it to the next page when the button is clicked. Earlier we created a function inside the models named “get\_absoulute\_url”: the “get\_absoulute\_url” function guides the user to more information on the specified portfolio.

A computer code with text

Description automatically generated with medium confidence

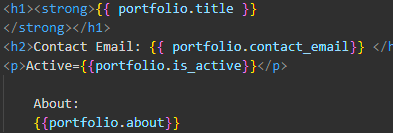
* The “ursl.py” contains all the paths to where the user is led. The “urls.py” and “views” work together to establish routing. Without the URL paths, would not be able to navigate through the site. Be sure when a new page is created to link it in  “urls.py”. A screen shot of a computer

  Description automatically generated

## **Useful Resources**

* [Resource 1](https://www.w3schools.com/django/django_add_main.php)
  + This site shows the bare bones of creating a homepage. It is a helpful guide in creating a server.

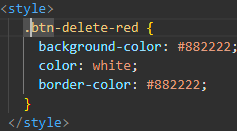
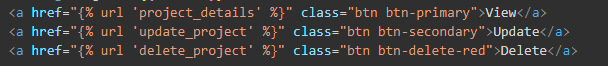
# **SECTION 4.4 PORTFOLIO DETAIL PAGE**

The focus of section 2.4 surrounds the setup of the portfolio .html. In this section I created the portfolio\_details.html. Pulling from the home page, I was able to easily create the main details of contact email, about and active. Using the generic template I implemented the features given from the portfolio model as seen here:

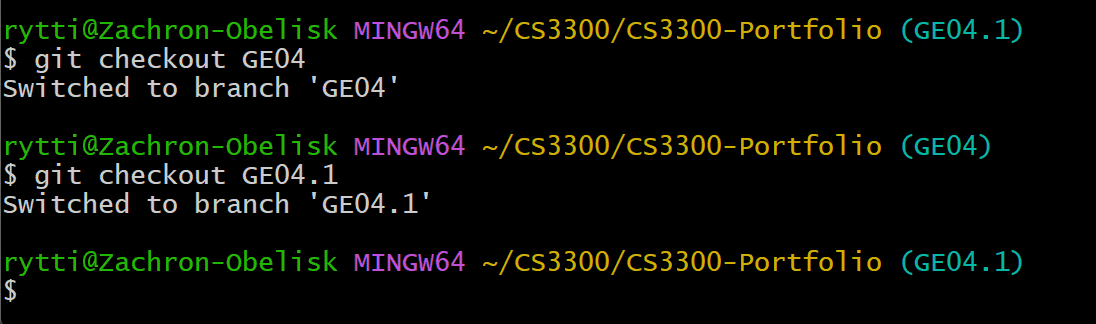
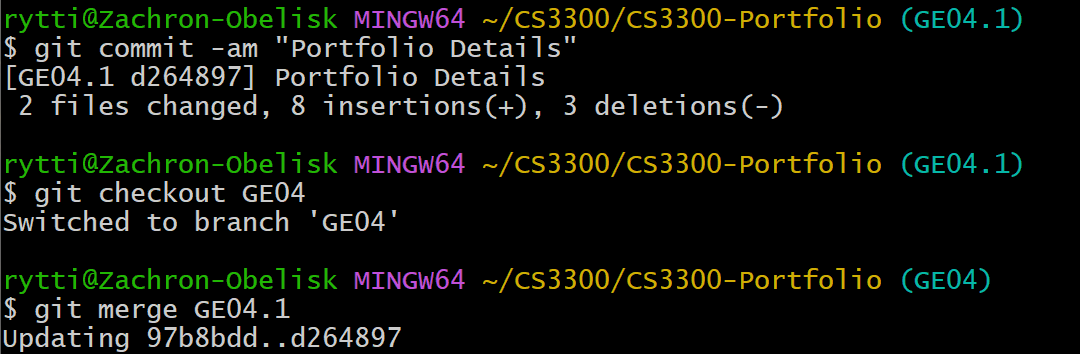
Creating the projects

* creating the projects was the most difficult part. First with the generic form, I had to give it context. This meant that I had to use the context feature of the generic form, so it could access all projects filtered to that portfolio.

Button Shenanigans

* to display the buttons nicely, I kept them all in the same paragraph, and changed their color. For the first two I used the standard templates for a primary button and a secondary button respectively. For the third, I created a custom color using a color wheel to get a more muted red from the vibrant one that came with
  + using 0x882222, I created a better color, and then added a style to the base template named “btn-delete-red”,
  + which I could then use as the class for the delete button
  + 

## Issues Encountered

* I did not initially use the generic templates, causing a massive headache for myself. This was eventually fixed when I formatted it properly, and took the time to reread the lecture notes. I created a new version to undo all of my work that ended up being for nothing, just in case my latest iteration was a failure as well. However I was able to successfully get it to work.
* and was able to merge it back onto the GE04 branch
* 

## Useful Resources

* <https://canvas.uccs.edu/courses/152505/pages/l13-django-model-view-template?module_item_id=2446089>
  + This shows the generic views provided.
* <https://www.canva.com/colors/color-wheel/>
  + this is useful for customizing the various buttons in order
* <https://www.w3schools.com/bootstrap/bootstrap_buttons.asp>
  + brief overview of bootstrap’s buttons
* <https://www.freecodecamp.org/news/css-button-style-hover-color-and-background/>
  + button setup
* <https://docs.djangoproject.com/en/4.2/topics/class-based-views/generic-display/>
  + for generic template aid

# **SECTION 4.5 FORMS**

Forms are one methodology for taking user input. Towards the end of the project, the focus is on updating and creating new forms that will aid in gaining incoming user information. Creating and updating forms is an important aspect of taking user data and applying the changes to the database.

## **Issues Encountered**

* Issue 1: Although filling out forms is a simple task, understanding how they work is challenging. When forms are submitted by the end-user, a “POST ” is made. The form checks for correct date entry that then edits the database, but disregards insufficient. It is complicated to visualize how forms are produced.

* Solution 1:
  + Pictures/diagrams are an amazing way to visualize confusing concepts and break them down into more manageable tasks. To understand the process of creating a form, draw a picture.
    - A form requires information. The model provides the information for the form requirements. For example, the user must enter a title and description to move forward. A valid response involves filling out the necessary data, while an invalid response produces an error message. After the user fills out a valid form, a request is made and sent to “views”. The “views” searches and then executes a request. After the search, a response is returned. The visual display of the response is handled by the HTML, which includes a changing token to protect users.
    - On the other hand, if a user enters invalid data, they are given an error message from the “view.py”. The invalid data does not update or affect the database.
  + A whiteboard with text on it

    Description automatically generatedA whiteboard with writing on it

    Description automatically generated

## **Useful Resources**

* [Resource 1](https://www.geeksforgeeks.org/django-forms/)
  + This resource provides images and code snippets that explain the purpose of each aspect of form creation.

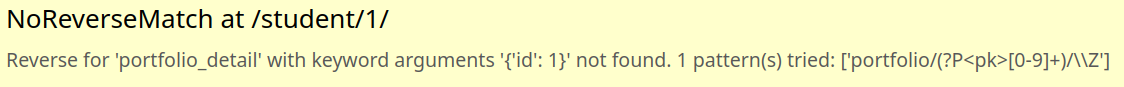
# **SECTION 4.6 DISPLAY STUDENT DETAILS**

**General description:**

Implementing the student details display view involved very little work. It is a barebones detail view with two extra links leading to other pages. This link functionality is the main challenge in this section, as it requires creative use of the ORM to implement. Other than that, the section is quick, as the developer has likely already implemented detail views twice at this point in the project.

## Issues Encountered

* **Issue 1:** I attempted to create a URL to the associated folder using the wrong URL parameter. This issue expresses itself as a match not found error, giving a regex code in the error.



* + Solution 1 :
    - To fix this, I corrected the url argument from id:
    - to pk:
    - After that, the Django URL system substitutes the proper URL.

## Useful Resources

* Resource : [Django Documentation - NoReverseMatch Exception](https://docs.djangoproject.com/en/4.2/ref/exceptions/#django.urls.NoReverseMatch)
  + This resource helps explain what a NoReverseMatch exception is in Django.

# **SECTION 4.7 STUDENT LIST VIEW**

* Creating the student list was fairly straightforward. First, set up the generic list view provided. This allows the url to be set up correctly, giving all needed context,

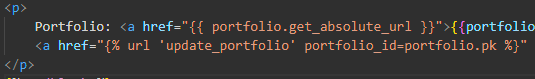
## Issues Encountered

* I had an issue where I set up my one-to-one connection in my student object, so the id it used was actually the portfolio id, and was therefore renamed by django. I had to play around a bunch in order to discover the problem, then renamed the variable to work correctly. 

## Useful Resources

* <https://canvas.uccs.edu/courses/152505/pages/l13-django-model-view-template?module_item_id=2446089>

# **SECTION 4.8 STUDENT DETAIL VIEW**

* This was incredibly easy to set up. The main details is very similar to the portfolio details, so look at that to figure out how to set up most of the issues. The only complicated thing is setting up the absolute url of the portfolio. In the end, the same method used in Student List works here, where the get\_absolute\_url links properly to the url of the portfolio.

## Issues Encountered

None.

## Useful Resources

None.