# Gamer Reviews 4 Gamers Technical Document v 0.0.1

Contents

[Gamer Reviews 4 Gamers Technical Document v 0.0.1 1](#_Toc150121051)

[Set Up: 1](#_Toc150121052)

[Code Functionality: 2](#_Toc150121053)

[Templates: 2](#_Toc150121054)

[Views: 3](#_Toc150121055)

[Forms: 4](#_Toc150121056)

[Models: 5](#_Toc150121057)

[URLs: 5](#_Toc150121058)

## Set Up:

The following steps assume that you have visual studio code, git, and python installed.

1. Clone the repository from github <https://github.com/TheSpaceViking/CS3300_Web_Project.git>

If using vs code, in the terminal use the following commands:

cd desiredDisk:\path\to\desired\directory

git clone <https://github.com/TheSpaceViking/CS3300_Web_Project.git>

1. Using the terminal create a python virtual environment, preferably named djvenv due to gitignore settings

python -m venv djvenv

.\djvenv\Scripts\Activate

1. Ensure the page can be visualized in your web browser at <http://127.0.0.1:8000/>
   1. In the terminal:  
      python manage.py runserver
   2. To stop running the server use ctrl-c in the teminal
2. Deactivate the virtual environment before closing vscode
   1. In the terminal:

deactivate

## Code Functionality:

### Templates:

Templates define how the page will look in general. These templates are written in HTML and tell the website how to display data from the views and forms created in python. The default template also determines the uniform look of the site throughout the user’s experience.

<!-- Navbar -->

<nav class="navbar navbar-expand-lg custom-navbar">

    <div class="container-fluid">

        <img src="{% static 'images/gr4g\_logo.gif' %}">

        <button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-target="#navbarNav" aria-controls="navbarNav" aria-expanded="false" aria-label="Toggle navigation"><span class="navbar-toggler-icon"></span></button>

        <div class="collapse navbar-collapse" id="navbarNav">

            <ul class="navbar-nav">

                <li class="nav-item">

                    <a class="nav-link" href="{% url 'index' %}">Home</a>

                </li>

                <li class="nav-item">

                    <a class="nav-link" href="{% url 'game\_list' %}">Games</a>

                </li>

                <li class="nav-item">

                    <a class="nav-link" href="#">Menu 2</a>

                </li>

            </ul>

        </div>

    </div>

</nav>

In the above example from the default template, the navigation bar will be exactly the same as the user navigates through the website in order to allow for easy navigation and a fixed aesthetic throughout the user’s experience.

### Views:

Views define the way object data is displayed on screen. Views allow users to absorb and use this information in a few different ways.

#### Detailed Views

Detail Views show full context of an object and connects other objects to enhance the data displayed. The game view example shown below shows the game object data such as the Publisher, release year, overall rating, what platforms it can be played on, and what genres it belongs to. It also sets up the review cards that are related to it. The publisher, the reviews, genres, platforms, and ratings are all separate objects that share different types of relationships to the game and each other. The data for these views are brought up from SQL queries generated using python code and the Django framework.

            <!-- Game Details -->

            <h1>{{ game.title }}</h1>

            <p><strong>Publisher:</strong> {{ game.publisher.name }}</p>

            <p><strong>Release Year:</strong> {{ game.release\_year }}</p>

            <p><strong>Overall Rating:</strong> {{ game.overall\_rating }}</p>

            <p><strong>Platforms:</strong>

                {% for platform in game.platforms.all %}

                    {{ platform.name }}

                    {% if not forloop.last %}, {% endif %}

                {% endfor %}

            </p>

            <p><strong>Genres:</strong>

                {% for genre in game.genre.all %}

                    {{ genre.game\_genre }}

                    {% if not forloop.last %}, {% endif %}

                {% endfor %}

            </p>

        </div>

    </div>

    <!-- Review Cards -->

    <div class="row mt-3">

        {% for review in game.review\_set.all %}

        <div class="col-md-4">

            <div class="card">

                <div class="card-body">

                    <h5 class="card-title">{{review.title}}</h5>

                    <p class="card-text">{{ review.content }}</p>

                    <p class="card-text">Overall Rating: {{review.overall\_rating}}</p>

                    <!-- Include other review details here -->

#### List Views

List views allow users to browse a list of objects to select specific object instances to view details of. These are much simpler than detailed views and can be set up in simple lists or table style blocks as shown below.

<style>

    table {

        table-layout: auto; /\* Automatically adjust column width based on content \*/

    }

</style>

<div class="container">

    <h1>All Games:</h1>

    <a href="{% url 'add\_game' %}" class="btn btn-primary ml-auto">Add Game</a>

    <table class="table table-bordered" id="games-table">

        <thead>

            <tr>

                <th style="text-align: center; vertical-align: middle;">Cover Art</th>

                <th style="text-align: center; vertical-align: middle;">Title</th>

                <th style="text-align: center; vertical-align: middle;">Publisher</th>

                <th style="text-align: center; vertical-align: middle;">Release Year</th>

                <th style="text-align: center; vertical-align: middle;">Overall Rating</th>

            </tr>

        </thead>

        <tbody>

            {% for game in games %}

                <tr>

                    <td style="text-align: center; vertical-align: middle;">

  <img src="{{ game.cover\_image.url }}" alt="{{ game.title }}" width="100" height="150">

                    </td>

### Forms:

Forms allow the users to interact with the database. Following CRUD principles, these forms allow users to Create, Read, Update, and Delete data from the database. These coincide with html files that tell the website how to display the information that the queries generated by the python code retrieved.

class GameForm(forms.ModelForm):

    class Meta:

        model = Game

        exclude = ['overall\_rating']

        labels = {

            'title': 'Title',

            'cover\_image': 'Cover Image',

            'genre': 'Genre',

            'publisher': 'Publisher',

            'release\_year': 'Release Year',

            'platforms': 'Platforms',

        }

        widgets = {

            'title': forms.TextInput(attrs={'class': 'form-control'}),

            'cover\_image': forms.ClearableFileInput(attrs={'class': 'form-control-

file'}),

            'genre': forms.CheckboxSelectMultiple(),

            'publisher': forms.Select(attrs={'class': 'form-select'}),

            'release\_year': forms.DateInput(attrs={'class': 'form-control', 'type':

'date'}),

            'platforms': forms.CheckboxSelectMultiple(),

        }

### Models:

In simple terms models define how database tables are set up and how information is stored. These models create objects. For example:

class Publisher(models.Model):

name = models.CharField(max\_length=100)

website = models.URLField()

    contact = models.CharField(max\_length=200)

This model creates a publisher object. This object contains pertinent information such as the name of the publisher, the website that belongs to the publisher, and the contact details of the publisher. This information can then be displayed by views, and manipulated in forms.

### URLs: