CS551Q Database-Driven Django Application Design and Development Solo Report

Ho Pok Nga 52210984 (MScIT)

A database-driven Django web application with video game sales[1] is developed. In this web application, video games with the top 3000 sales are chosen to be included in this database. There are more than 10000 records of video game details with their sales included in the original dataset[1]. A limited amount of video games (3000) is included in this web application to minimise the web application deployment speed. I did not choose the year of release as the major parameter as the dataset is based on sales will be greatly disturbed. I chose this video games dataset as the sales-related data can be well presented using graphs. The data structure in this database is also simple but has enough complexity for breaking up into multiple tables but does not consist of over-complexed tables which require further data selection or analysis before usage.

There are nine linked tables in this web application where five of which are video games related and the other four are e-commerce related. In the index.html, a video game list is displayed. Any web application user can add items to the basket by clicking the “Add to Basket” button with quantity on the bottom of the video game detail page. Multiple items can be added to the basket. However, only registered customers including staff or admin can move to the next phase for the purchase by making a payment. A comparison bar graph is added for users to compare any two games’ sales data side by side. A pie chart and a bar graph are used in the dashboard to show the proportion of registered account types in the web application and the number of orders from each customer. Dashboard features can be viewed by staff and admin account users. The authentication system is set up. Three account types: customer, staff and admin have different permission types. Error handling is used when video game data are parsed into the database. Two base templates are created: index.html is the base for game_full_list.html and game_filter_list.html shows the full or filtered list of video games where base.html is the base for all e-commerce-related web pages. So head and navigation bar parts are shared preventing duplication. Different measures are done to keep the manageability of the code. The data parsing part for video game data is separated from e-commerce data into two Python files. Templates for video games are put in the base folder whereas templates for e-commerce are put in the shop folder with the registration part put in another registration folder. Tests are separated into different files based on their testing aim in the tests folder. Views are also separated into different Python files in the views folder base on their responsible part. So it is separated into base.py (video games) and shop.py, basket.py, customer.py and order.py (e-commerce). Search functions are implemented in three places. In the comparison page, the graph shows sales data for two user-chosen video games with a filter base on the video game’s rank. In the list of video games, advanced search is applied with multiple parameters including platform, year, genre and publisher. A different number of parameters can be used in the same search. Behave, a behavioural-development-driven (BDD) test method is used to cover different parts of the web application in the features folder. The process of adding the video game to the basket is tested through Behave. Cloud deployment is successfully deployed to Render[2] and Codio’s virtual environment[3]. The Sqlite3 database is used during cloud deployment. Three example account types are created with included permissions. Account details are listed in README.md. A game console icon (favicon.ico) is added to the web application as the icon.

There are five steps in the web development process assessing, planning, developing, testing and deploying[4]. In the assessing and planning stages, I repeatedly studied the requirements from the professor and design the models with a three-week limit for the whole project.

With limited developing time, Agile Development Methodology[5] with continuous integration (CI) and continuous deployment (CD) were used to implement iterative and incremental development and delivery[6]. Only working code is pushed to GitHub and thus auto-deployed to Render which is developed when the base of the web application by showing data stored in the database is done.

The developing stage is separated into two major parts which are the design and implementation of the database and frontend pages respectively. To satisfy the 2000-7000 data rows requirement, the top 3000 video games with the highest sales from the Comma-Separated Values (CSV) file are chosen. Year, genre, platform and publisher fields have separate CSV files and separate models to save every unique value. Simple HyperText Markup Language (HTML) & Cascading Style Sheets (CSS) templates are created to verify items parsed into the database. For the e-commerce part, customers are created using Faker and orders are randomly generated and added to the database.

For the front end, HTML templates are kept simple and the same CSS is shared among the web pages to keep the look and feel throughout the web application. Tables are mostly used for displaying video games related data and lists are for displaying e-commerce-related data such as customers and orders. Filtering features are also implemented twice using forms. Graphs are also implemented three times throughout the web application.

Model-template-view (MTV) pattern, the Django version of the model-view-controller (MVC) pattern, is applied in the implementation. Model is the database of 9 tables, templates (frontend HTML) are created in the templates folder in the games application following the Django convention and view is the controller (files in views folder) controlling all functions implemented[7].

For the implementation stage, we created a Django project (mysite) and application (games), configured Django settings to include Chart.js library, created models for video games and e-commerce with Django ORM, created views for pie chart and bar graphs using Chart.js library, created templates and URLs for the table and bar chart using HTML and Chart.js.

For the testing stage, unit tests and behave are used. Fixtures are used by using game_test.json. Six test types (for filters, models, login, templates, views and graphs) are implemented in different files. First, the templates test tests whether different pages can be accessed and connected. The main method uses response.status_code=200[8] to verify the page connectivity. Second, model testing is for six base models: Year, Genre, Platform, Publisher, Game and Cart. Models are tested by retrieving different data items, with importing models and fixtures at the beginning. Third, the web view test focuses on the successful loading of the form and checking the form content, by creating test data and using assertContains to determine whether the form contains the test data. Fourth, the filter test tests if the correct data items can be retrieved with different parameter filters working in forms. Fifth, the graph test tests the generated graph if the graph is present and consists of the graph labels. Sixth, login and logout with created customers are tested. Behave tests are done to test if the behaviour of the application works as expected.

For the deployment stage, Render is used to deploy the Django application; a cloud platform that offers a simple way for web application deployments. Render deployment starts after the web application base is completed for CD. GitHub is used throughout the development process and everything is pushed to GitHub when finishing a version of the current feature, implementing CI.

In conclusion, the whole process is learnt from assessing to deploying a database-driven Django web application with a filtering function and showing visual graphs. Data filtering and visualisation are studied and applied. Unit tests and behave are applied and Render is used for website deployment.

Website links

Submission folder in Pok Nga Ho's Solo Assignment (Master): cs551q_solo

Render deployed website: https://cs551q-games.onrender.com/

Reference

[1] GregorySmith, "Video game sales," Kaggle. https://www.kaggle.com/datasets/gregorut/videogamesales (accessed May 04, 2023).

[2] "Video games 🎮." https://cs551q-games.onrender.com/ (accessed May 04, 2023).

[3] "Video games 🎮." https://wheelpioneer-bananashock-8000.codio-box.uk/ (accessed May 04, 2023).

[4] N. Beacham and K.Musa (2023). CS551S - Web Development Lecture day 1 - Introduction to the Module & How the Internet Works [PowerPoint slides]. Available: https://abdn.blackboard.com/ultra/courses/_56779_1/outline/edit/document/_3587470_1?courseId=_56779_1&view=content

[5] A. Ahmed, S. Ahmad, N. Ehsan, E. Mirza, and S. Z. Sarwar, "Agile software development: Impact on productivity and quality," in 2010 IEEE International Conference on Management of Innovation & Technology, 2010. Accessed: Feb. 10, 2023. [Online]. Available: http://dx.doi.org/10.1109/icmit.2010.5492703

[6] B. Scharlau (2022). CS551A - Software Engineering Week 1 Lecture 4 - Scrum and XP for Your Team [PowerPoint slides]. Available: https://abdn.blackboard.com/ultra/courses/_56808_1/outline/edit/document/_3570484_1?courseId=_56808_1&view=content

[7] B. Scharlau (2021). CS551Q - Enterprise Software Development Week 2 Lecture 6 - MVC with Django [PowerPoint slides]. Available: https://abdn.blackboard.com/ultra/courses/_56777_1/outline/file/_3572168_1

[8] B. Scharlau(2023). CS551Q - Enterprise Software Development Week 2 Lecture 7 - Testing Models in Django[PowerPoint slides]. Available: https://abdn.blackboard.com/ultra/courses/_56777_1/outline/file/_3574334_1