DCU EEN1037 -Assignment 3

Server-Side Programming

Name: Mohammed Al Shuaili

Date: 15/04/2025

Table content

# Introduction

# Implementation

# Django Models / SQL Database Tables

1. 4 or more SQL database tables managed by Django ORM Model classes,
2. You must have at least one Model/Table with a ForeignKey relation to another
3. You source code should include the “migrations/\*.py” o

# Forms for typical CRUD (Create, Read, Update, Delete) operations

4 or more POST request forms. These can be regular HTML POST forms or AJAX POST forms. You must have at least one or more forms for each of the following: ◦ (A) Creating a new entry in the database ◦ (B) Updating an existing entry in the database ◦ (C) Deleting an existing entry in the database • After using these forms, the changes should be visible somewhere on the website. • All POST request forms should include server-side validation of every uploaded field. • At least 2 POST request forms should also include client-side Javascript validation.

# Client-side AJAX Functionality

1 or more JSON-based AJAX GET Request. This can initiated from any kind of event, e.g. a button press or on page load. If the call fails, it should show an error but the rest of the page should continue working. If the call succeeds, it should update part of the page. • 1 or more JSON-based AJAX POST Request that perform some action to an entry in the database, i.e. Create, Update, or Delete. If the request fails, an error should be shown on the page. If the request succeeds, it should result in some part of the page being updated, either using the JSON results returned directly from the AJAX POST request, or by triggering a seperate AJAX GET Request

# User registration & login functionality

User registration. There should be a sign-up or registration form somewhere on your site that allows unauthenticated users to create new User accounts. • User login. There should be a simple login page somewhere on your site that takes a username and password and authenticates the current user. • User logout. There should be a way for a user to log out of the site To implement this section, you are recommended to take advantage of the Django authentication system built-in “User” objects. For additional user profile information, you can create an associated UserProfile model in your models.py which has a ForeignKey relation to the built-in User model you just created, and which also contains additional fields relevant to your application, for example Address, Telephone Number, any user preferences etc. • “staff” users which can perform additional functionality on the site, e.g. the ability to add products to the catalogue, or make new posts. For project demonstration purposes, these staff users can be created as normal users using your existing registration functionality, then manually converted into staff accounts using the built-in Django Admin UI by setting the user’s “is\_staff” field to true. Then in your project HTML templates, you can use “{% if request.user.is\_staff %}Something only staff should see{% endif %}” and in your View functions check for “request.user.is\_staff == True” before allowing staff restricted form operations to continue.

# Docker container, SQL Database connectivity & migrations

It should be possible to build your source code into a Docker container and run the resulting image with the following commands: docker build -t myapp docker volume create myapp-storage docker run -ti -e DATABASE\_URL="mysql://myappdbuser:myappdbpass@host.docker.internal:3306/ myappdb" -v myapp-storage:/app/storage -p 8000:8000 myapp • The Web Server should be accessible at "http://localhost:8000" • The project should be able to connect to the database specified in DATABASE\_URL environment variable (i.e. using the “dj-database-url” plugin). In the commands above, it will connect to a running MySQL server running on your local computer, to the database "myappdb", with credentials "myappdbuser", "myappdbpass". • The SQL migrations should automatically run on project startup against a clean fresh database. The "Django Example Project" already implements all the required Dockerfile and dj-database-url configuration for this section, but you are required to test and make sure it is still working before submitting your project, as this is how we will test your project

# Submission:

• 1x Zip file of project folders copied from Visual Studio Code with all relevant code files from the project directory ◦ If you are using git for source control, you can use the following command to export your folder, which will exclude any unnecessary binaries. ▪ git archive --format=zip --output project-export.zip HEAD • You are required to add comments in your code wherever possible, as you would explain to someone new to the project what a particular section does and how it works and why you did it that way. You don’t have to comment every line, just the major sections of work that you have done. • 1x ReadMe file containing installation and configuration instructions • 1x Briefing document (word doc or pdf)

Conclusion