Unit 4: Full Stack Frameworks with Django

Level:	Level 6
Credit Value:	25
GLH:	190
Unit Number:	F/650/3528
Unit Aim:	This unit aims to provide learners with the knowledge and skills needed to build a Full Stack web application. The unit covers the use of frameworks; APIs; automated testing; persistent storage; user authentication and e-commerce functionality.

This unit has 5 learning outcomes.

LEARNING OUTCOMES The learner will:	ASSESSMENT CRITERIA - PASS The learner can:	MERIT CRITERIA* In addition to the pass criteria, the learner can:	DISTINCTION
1 Design, develop and implement a Full Stack web application, with a relational database, using the Django/Python Full Stack MVC framework and related contemporary technologies.	1.1 Design a Full Stack web application to be built using the Django framework and to incorporate a relational database and multiple apps (an app for each potentially reusable component). 1.2 Design a front end for a Full-Stack web application that meets accessibility guidelines, follows the principles of UX design, meets its given purpose and provide a set of user interactions. 1.3 Develop and implement a Full Stack web application built using the Django framework, to incorporate a relational	M(i) Design and build a real-world Full Stack MVC application with a Front end: - that is easy to navigate and allows the user to find information and resources intuitively. - where the user has full control of their interaction with the application. - where all data (CRUD) actions are immediately reflected in the user interface. - where the purpose is immediately evident to a	**Characteristics of Performance at Distinction

LEARNING OUTCOMES The learner will:	ASSESSMENT CRITERIA - PASS The learner can:	MERIT CRITERIA* In addition to the pass criteria, the learner can:	DISTINCTION
	database, an interactive front-end and multiple apps (an app for each potentially reusable component). 1.4 Implement at least one form, with validation, that allows users to create and edit models in the Back end. 1.5 Build a Django file structure that is consistent and logical, following the Django conventions. 1.6 Write code that clearly demonstrates characteristics of 'clean code'. 1.7 Define application URLs in a consistent manner. 1.8 Incorporate a main navigation menu and structured layout. 1.9 Include custom logic that shows proficiency in the Python language. 1.10 Write Python code that includes functions with compound statements such as if conditions and/or loops. 1.11 Design and implement manual or automated test procedures to assess functionality, usability, responsiveness and data	new user. - which provides a good solution to the user's demands and expectations. - which has a clear, well-defined purpose addressing the needs of a particular target audience (or multiple related audiences). M(ii) Produce a robust codebase M(iii) Follow a Test Driven Development (TDD) approach (for JavaScript and/or Python), demonstrated in the git commits. M(iv) Configure the project efficiently through well-kept Procfile, requirements.txt file, settings files, keeping the data store configuration in a single location where it can be changed easily.	

LEARNING OUTCOMES The learner will:	ASSESSMENT CRITERIA - PASS The learner can:	MERIT CRITERIA* In addition to the pass criteria, the learner can:	DISTINCTION
	management within the full web application.		
2 Design and implement a relational data model, application features and business logic to manage, query and manipulate relational data to meet given needs in a particular real-world domain.	2.1 Design a relational database schema with clear relationships between entities. 2.2 Create at least TWO original custom Django models. 2.3 Create at least one form with validation that will allow users to create records in the database (in addition to the authentication mechanism). 2.4 Implement all CRUD (create, select/read, update, delete) functionality	M(v) Fully describe the data schema in the project README file.	
3 Identify and apply authorisation, authentication and permission features in a full-stack web application solution.	3.1 Implement an authentication mechanism, allowing a user to register and log in, stipulating a clear reason as to why the users would need to do so. 3.2 Implement log-in and registration pages that are only available to anonymous users. 3.3 Implement functionality that prevents non-admin users from accessing the data store directly without going through the code.	M(vi) Demonstrate solid understanding of Django template syntax, logic and usage, placing Django logic in the component where it is best suited (e.g., data handling logic is in the models, business logic is in the views).	
4 Design, develop and integrate an e-commerce payment system in a cloud-hosted Full Stack web application.	4.1 Implement at least one Django app containing some e-commerce functionality using an online payment processing		

LEARNING OUTCOMES The learner will:	ASSESSMENT CRITERIA - PASS The learner can:	MERIT CRITERIA* In addition to the pass criteria, the learner can:	DISTINCTION
	system (e.g. Stripe). This may be a shopping cart checkout, subscription-based payments or single payments, donations, etc. 4.2 Implement a feedback system that reports successful and unsuccessful purchases to the user, with a helpful message.		
5 Document the development process through a git based version control system and deploy the full application to a cloud hosting platform.	5.1 Deploy the final version of your code to a hosting platform and test that it matches the development version. 5.2 Ensure that all final deployed code is free of commented out code and has no broken internal links. 5.3 Ensure the security of the deployed version, making sure to not include any passwords in the git repository, that all secret keys are hidden in environment variables or in files that are in gitignore, and that DEBUG mode is turned off. 5.4 Use a git-based version control system for the full application, generating documentation through regular commits and in the project README.	M(vii) Use version control software effectively to provide a record of the development process.	

LEARNING OUTCOMES The learner will:	ASSESSMENT CRITERIA - PASS The learner can:	MERIT CRITERIA* In addition to the pass criteria, the learner can:	DISTINCTION
	5.5 Create a project README that is well-structured and written using a consistent markdown format. 5.6 Document the full deployment procedure, including the database, and the testing procedure, in a README file that also explains the application's purpose and the value that it provides to its users.		

*Additional guidance for merit

To achieve merit learners, need to meet the assessment criteria outlined above for a pass and a merit.

The following additional guidance describes characteristics of performance at MERIT.

The learner has a clear rationale for the development of this project and has produced a fully functioning, well-documented, relational database backed, Full Stack application for a real-life audience, with a full set of CRUD creation, reading, updating and deletion of data records) features . Data validation, API handling and user feedback are all evident in the code and the working application. Templates have been used correctly produce working features. There are no logic errors in the code and the application functions as expected.

The finished project has a clear, well-defined purpose addressing the needs of a particular target audience (or multiple related audiences) and a particular data domain. Its purpose would be immediately evident to a new user without having to look at supporting documentation. The user is kept informed of progress and actions through feedback and, where large data sets are being loaded, progress indicators.
The design of the web application follows the principles of UX design and accessibility guidelines, and the site is fully responsive.

Data is fully modelled and matches the schema. The schema design is documented in the README. Data store configuration is kept in a single location and can be changed easily. Configuration and settings files are well-organised and there are different versions for different branches.

Code is well-organised and easy to follow, and the application has been fully tested, following a test-driven development approach and a planned, testing procedure, with no obvious errors left in the code.

The development process is clearly evident through detailed commit messages. The project's documentation provides a clear rationale for the development of this project and covers all stages of the development life cycle.

The application is robust and deals with external errors gracefully (user input, API calls, asynchronous processes).

The conventions of the framework are followed, and code is clearly separated with HTML, CSS, JavaScript and Python being kept in separate, linked files.

All logic in the app makes sense in the context of its purpose (e.g. if different users have different permissions, then their access levels are appropriate so that, for example, a non-admin user will not be able to edit another user's data.

**Characteristics of Performance at Distinction

To achieve a distinction, a learner will have achieved all pass and merit criteria, as described above, and will demonstrate characteristics of high-level performance as described below:

Characteristics of performance at distinction:

The learner has documented a clear, justified, rationale for a real-world application and a comprehensive explanation of how it will be developed. The development of the project has

resulted in a fully-functioning, interactive, Full Stack Django application, with well-designed data and a full set of CRUD operations. The learner shows a clear understanding of data modelling techniques and of the relationship between the Back end and Front end.

The finished project is judged to be publishable in its current form with a professional grade user interface and functionality, and interaction adhering to current practice. There are no logic errors in the code. Where there is a clear breach of accepted design/UX principles, or of accepted good practice in code organisation, these are fully justified, appropriate and acceptable to the target user. It clearly matches the design and demonstrates the characteristics of craftsmanship in the code. The database schema is representative of complex user stories and there is a fully documented and full set of data operations which are fit for purpose in relation to the domain.

Each individual app matches a natural aspect of the project, with no app being too small or too big (a common approach is to have an app to encapsulate each single set of tightly connected models). Any data in the models that is relevant to multiple apps is shared, rather than duplicated.

The resulting application is original and not a copy of any walkthrough projects encountered in the unit.

Amplification (craftsmanship)this amplification is only applicable to performance at distinction.

Front end Design

The design of the web application demonstrates the main principles of good UX design:

- Information Hierarchy
  - semantic markup is used to convey structure - all information displayed on the site is presented in an organised fashion with each piece of information being easy to find
  - all information displayed on the site is presented in an organised fashion with each piece of information being easy to find
  - all resources on the site are easy to find, allowing users to navigate the layout of the site intuitively
  - information is presented and categorised in terms of its priority
- User Control
  - all interaction with the site would be likely to produce a positive emotional response within the user. This is down to the flow of information layout, use of colour, clear and unambiguous navigation structures and all interaction feedback
  - when displaying media files, the site avoids aggressive automatic pop-ups and autoplay of audio; instead letting the user initiate and control such actions
  - users who direct to a non-existent page or resource are redirected back to the main page without having to use the browser navigation buttons
  - users are never asked for information that the application already has (e.g. a contact form does not ask a logged in user for an email address).
  - the user is shown progress indicators and feedback on transactions.
  - errors resulting from user or data actions are reported to the user
- Consistency
  - evident across all pages/sections and covers interactivity as well as design
  - consistency across all data operations, including in the reporting
- Confirmation
  - user and data actions are confirmed where appropriate, feedback is given at all times
- Accessibility
  - there is clear conformity to accessibility guidelines across all pages/sections and in all interactivity

Any design decisions that contravene accepted user interaction, user experience design principles are identified and described (comments in code and/or a section in the README)

Development and Implementation

Code demonstrates characteristics of ‘clean code’:

- Consistent and appropriate naming conventions within code and in file naming, e.g.
  - file names, class names, function names and variable names are descriptive and consistent

    - for cross-platform compatibility, file and directory names will not have spaces in them and will be lower-case only
    - all HTML attributes, CSS rules, code variables and function names are consistent in format, follow standards for the language and are appropriate and meaningful
    - app urls are consistent
- File structure
    - whenever relevant, files are grouped in directories by file type (e.g. an assets directory will contain all static files and code may be organised into sub-directories such as CSS, JavaScript, etc)
    - there is a clear separation between custom code and any external files (for example, library files are all inside a directory named 'libraries')
    - files are named consistently and descriptively, without spaces or capitalisation to allow for cross-platform compatibility.
- Readability
    - code is indented in a consistent manner to ease readability and there are no unnecessary repeated blank lines (and never more than 2)
    - id/class(CSS and JavaScript)/function/variable names clearly indicate their purpose
    - all code is split into well-defined and commented sections
    - Semantic markup is used to structure HTML code
    - HTML, CSS, JavaScript and Python are kept in separate, linked files
    - CSS files are linked in the HTML file's head element
    - non-trivial JavaScript code files are linked to at the bottom of the body element (or bottom of head element if it needs to be loaded before the body HTML)
- Defensive design
    - all input data is validated (e.g. presence check, format check, range check)
    - internal errors are handled gracefully, and users are notified of the problem where appropriate
- Comments
    - all custom code files include clear and relevant comments explaining the purpose of code segments
- Compliant code
    - HTML code passes through the official W3C validator with no issues
    - CSS code passes through the official (Jigsaw) validator with no issues
    - JavaScript code passes through a linter (e.g. jshint.com) with no major issues
    - Python code is consistent in style and conforms to the PEP8 style guide (or another explicitly mentioned style guide, such as Google's)
- Robust code
    - no logic errors are found when running code
    - errors caused by user actions are handled
    - where used, API calls that fail to execute or return data will be handled gracefully, with the site users notified in an obvious way
    - inputs are validated when necessary
    - navigating between pages via the back/forward buttons can never break the site, there are no broken links
    - user actions do not cause internal errors on the page or in the console

The full design is implemented providing **a good solution to the users' demands and expectations** and **with consideration for security**.

Real world application

- Clearly understandable site-specific content is used rather than Lorem Ipsum placeholder text
- All links to external pages open in a separate tab when clicked
- The final application is aligned to the user stories presented at the start of the project

Testing procedures are comprehensive, with a good level of coverage, and have clearly been followed. All noticeable errors have been corrected or documented:

Security features and practice are evidenced

- passwords and secret keys are stored in environment variables or in files that are in .gitignore, and are never committed to the repository
- any functionality requiring log-in is available only to logged-in users
- user permissions and levels of access are appropriate (e.g. a non-admin user would not be able to edit another user' post)

Framework conventions are followed and used correctly. Including the following:

Django/Flask:

- Templates
- Apps
- Models
- Views
- Placing of logic in the most appropriate components demonstrates an understanding of the Model-View-Controller (Template) pattern is evident through the placing of logic in the most appropriate components.
- Configuration and settings files are well-organised

Security features and practice are evidenced

- passwords and secret keys are stored in environment variables or in files that are in .gitignore, and are never committed to the repository
- any functionality requiring log-in is available only to logged-in users
- user permissions and levels of access are appropriate (e.g. a non-admin user would not be able to edit another user' post)

Data is well structured

- data is fully modelled and matches the schema.
- data store configuration is kept in a single location where it can be changed easily.
- data is well-structured, organised into logical entities with clear relationships between them
- all CRUD functionality is present and working and actions are immediately reflected in the front end
- any data used across multiple apps is shared and not duplicated.

Configuration and dependencies files are kept up to date. Separate versions/branches of these are commits where relevant. Data store configuration is kept in a single location and can be changed easily. The data store is not accessible to the regular user without going through the code.

Testing procedures are comprehensive, with a good level of coverage, and have clearly been followed. There is clear evidence of a test-driven development approach and this is demonstrated in git commits. All noticeable errors have been corrected or documented:

- navigating between pages via the back/forward buttons can never break the site, there are no broken links
- user actions do not cause internal errors on the page or in the console

Version control software is used effectively:

- all code is managed in git with well-described commit messages
- there is a separate, well-defined commit for each individual feature/fix
- there are no very large commits which make it harder to understand the development process and could lead the assessor to suspect plagiarism
- there are different versions of configuration and settings files for different branches

The full application development process is documented:

- the purpose of the application is clearly described in the README
- the project's documentation describes the UX design work undertaken for this project and the reasoning behind it
    - wireframes, mock-ups, diagrams, etc., created as part of the design process are included in the project
- there is a clear separation between code written by the learner and code from external sources (e.g. libraries or tutorials). All code from external sources is attributed to its source via comments above the code and (for larger dependencies) in the README
- the data schema is clear, comprehensive, and easy to follow.
- the data schema is fully documented in the README file
- the testing procedure is well documented either in the README or a separate file
- the deployment procedure is fully documented in a section in the README file
- clear, well-described, explanatory commit messages describe the development process
- the README is well-structured and easy to follow
- the README file is written in markdown and uses markdown formatting consistently and effectively
- project documentation and the application’s user interface have few errors in spelling and grammar