Unit 3: Back End Development

Level:	Level 5
Credit Value:	22
GLH:	153
Unit Number:	D/650/3527
Unit Aim:	This unit aims to provide learners with the knowledge and skills needed to build a Back end web application. Topics covered include data storage and data management using relational and non-relational databases.

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 Back end for a web application using Python and a micro-framework.	1.1 Design a Front end for a data-driven web application that meets accessibility guidelines, follows the principles of UX design, meets its given purpose and provides a set of user interactions. 1.2 Implement custom HTML and CSS code to create a responsive full-stack application consisting of one or more HTML pages with relevant responses to user actions and a set of data manipulation functions. 1.3 Build a database-backed Flask web application that allows users to store and manipulate data	M(i) Design a Front end for a Full Stack application following the principles of UX design which meets accessibility guidelines, is easy to navigate and allows the user to find information and resources intuitively. (Mii) Design a Full Stack application that lets the user initiate and control actions and gives immediate and full feedback on data processes. M(iii) Implement a Full Stack application whose	Please refer to the performance characteristics for distinction.**

LEARNING OUTCOMES	ASSESSMENT CRITERIA - PASS	MERIT CRITERIA*	DISTINCTION
The learner will:	**The learner can:**	**In addition to the pass criteria, the learner can:**	
	records about a particular domain. 1.4 Design a database structure that is relevant to the domain, including relationships between records of different entities. 1.5 Design and implement test procedures (automated or manual) to assess functionality, usability, responsiveness and data management within the Full Stack web application. 1.6 Write Python code that is consistent in style and conforms to the PEP8 style guide (or another explicitly mentioned style guide, such as Google's) and validated HTML and CSS code. 1.7 Write Python logic to demonstrate proficiency in the language. 1.8 Include functions with compound statements such as if conditions and/or loops in Python code. 1.9 Write code that meets minimum standards for readability (comments, indentation, consistent and meaningful naming conventions). 1.10 Name files consistently and descriptively, without spaces or	purpose is immediately evident to a new user and which provides a good solution to the user's demands and expectations. M(iv) Create templates, writing code that demonstrates understanding of template syntax, logic and usage. M(v) Write robust code that is free of errors in all parts of the application. M(vi) Fully document the results of well-planned testing procedures (automated or manual) to assess the website's functionality, usability and responsiveness. Include evaluation of bugs found and their fixes and explanation of any bugs that are left unfixed.	

LEARNING OUTCOMES The learner will:	ASSESSMENT CRITERIA - PASS The learner can:	MERIT CRITERIA* In addition to the pass criteria, the learner can:	DISTINCTION
	capitalisation to allow for cross-platform compatibility.		
2 Model and manage data.	2.1 Design a data model that fits the purpose of the project. 2.2 Develop the model into a usable relational or non-relational database where data is stored in a consistent and well-organised manner.	M(vii) Describe the data schema fully in the README file. M(viii) Maintain database configuration in a single location where it can be changed easily. M(ix) Maintain a Profile, requirements.txt file, settings file.	
3 Query and manipulate data.	3.1 Create functionality for users to create, locate, display, edit and delete records.	M(x) Implement working Create, Read, Update and Delete (CRUD) M(xi) Check that Create, Read, Update and Delete (CRUD) actions are immediately reflected in the user interface.	
4 Deploy a Full Stack web application to a Cloud platform.	4.1 Deploy a final version of the full-stack application code to a cloud-based hosting platform (e.g. Heroku) and test to ensure it matches the development version. 4.2 Ensure that final deployed code is free of commented out code and has no broken internal links. 4.3 Document the deployment process in a README file that also explains the application's	M(xii) Commit often for each individual feature/fix, ensuring that commits are small, well-defined and have clear descriptive messages. M(xiii) Fully document the deployment procedure in a section in a README file, written using consistent and effective markdown formatting that	

LEARNING OUTCOMES The learner will:	ASSESSMENT CRITERIA - PASS The learner can:	MERIT CRITERIA* In addition to the pass criteria, the learner can:	DISTINCTION
	purpose and the value that it provides to its users.	is well-structured, easy to follow, and has few grammatical errors.	
5 Identify and apply security features.	5.1 Use Git & GitHub for version control of a Full Stack web application up to deployment, using commit messages to document the development process. 5.2 Commit final code that is free of any passwords or secret keys, to the repository and to the hosting platform. 5.3 Use environment variables, or files that are in gitignore, to hide all secret keys. 5.4 Ensure that DEBUG mode is turned off in production versions.	M(xiv) Present a clear rationale for the development of the project, in the README, demonstrating that it has a clear, well-defined purpose addressing the needs of a particular target audience (or multiple related audiences), explaining the data, and explaining the security features considered.	

*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, 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. There is a range of features, including creation, location, deletion and updating of data records. Data validation, 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 manual testing procedure, with no obvious errors left in the code.

The development process is clearly evident through 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).

**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 application, with well-designed data and a full set of CRUD data 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. 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.

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 at the bottom of the body element (or bottom of head element if needs 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:

Framework conventions are followed and used correctly.
Flask:

- Controllers
- Models
- Views
- 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
- all CRUD functionality is present and working and actions are immediately reflected in the front end

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.

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

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, mockups, 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
- A manual 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