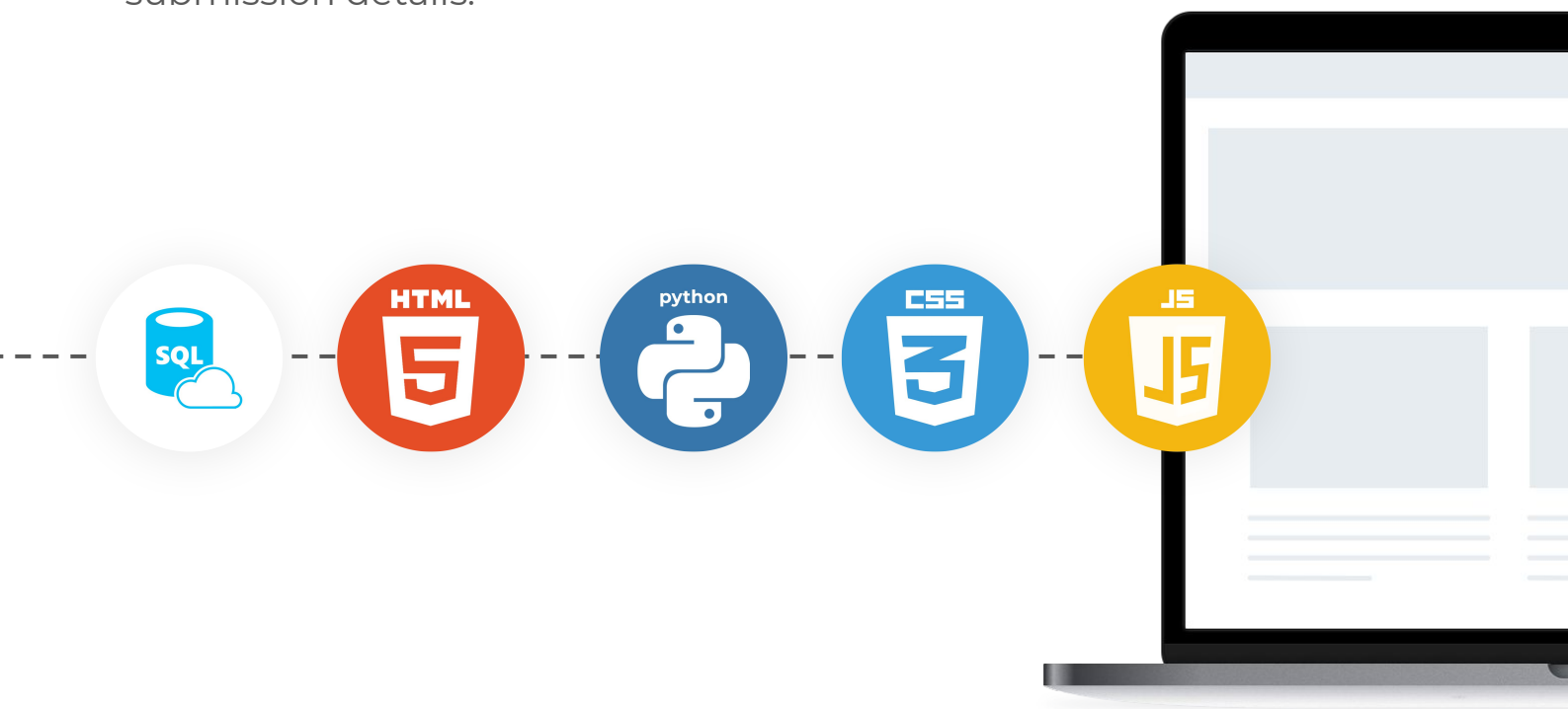
code institute

Assessment Handbook

Your go to guide for all things assessment related including project ideas, assessment criteria and submission details.



Europe's only university credit-rated coding bootcamp

Table of Contents

Milestone Project 1
User Centric Front-End Development

In this project, you'll build a static front-end site to present useful information to users, using all the technologies that you have learned about so far.

Data is presented in a way that helps users achieve their goals, e.g. learning about a product/service that they are interested in. The presentation of this data advances the site owner's goals, e.g. helps them market a product/service.

Main Technologies

Required: HTML, CSS

Optional: Bootstrap and/or other CSS libraries/frameworks.

Project Ideas

You can either choose to build the website based on Project Idea 0, or take inspiration from the other example ideas below.

Project Idea 0

Bring your own idea(s) to life, based on providing value to users to address a specific real or imagined need. Use the relevant project assessment criteria as a guide to the minimum required functionality

Project Example Idea 1

- Build a website for a band (real or fictional)

External user's goal:

- The site's users are fans and potential fans who wish to learn more about the band's history and the band members, and possibly book them for shows.

Site owner's goal:

- The band are interested in selling more of their music/merchandise and getting more gigs.

Potential features to include:

- Showcase photos, audio and/or video clips from the band's catalogue.
- Publicise the band's upcoming shows and/or availability to perform at events such as weddings and corporate parties.
- Provide links to external resources, such as the band's social media profiles (can point anywhere at all).

Project Example Idea 2

- Build a website for a gym

External user's goal:

- The site's users are gym members and potential members, who want to know more about the gym and its procedures.

Site owner's goal:

- The gym is interested in attracting and retaining members.

Potential features to include:

- Showcase photos of people having fun exercising in the gym and any other media to motivate people to come.
- Provide detail on the organised classes in the gym and their schedule.
- Provide information on the gym's location, opening hours, contact details and any external resources.

Project Example Idea 3

- Build a personal portfolio site (potentially for yourself).

External user's goal:

- The site's users are recruiters considering to hire the applicant.

Site owner's goal:

- Present self in best light and get hired.

Potential features to include:

- Include educational history and work experience.
- Outline skills and any other relevant competencies and interests.
- Provide basic personal information and contact information for recruiters.

Advanced potential feature (nice-to-have)

- Showcase portfolio of projects so far (in-lieu of links to real projects that you'd build later, feel free to include links to fake projects or random sites on the internet at this stage)

N.B. When choosing a milestone project, it is best practice to choose a new topic. The reason for this is to avoid duplicating your HTML/CSS code as it can only be graded once. If you resubmit part of a previous project, you cannot receive marks for this. For this reason, we strongly recommend that you create 4 different projects using the ideas provided. Furthermore, you want to display a diverse portfolio to prospective employers. In a small number of cases, if your reason for completing the course is to develop a business idea, you may want to create a single project.

As this is not the recommended approach, please do not proceed in this way before discussing your intention with Student Care, as this will need to be approved in advance of your submission.

User Centric Front-end Development Assessment Criteria

Learning Outcomes

LO1	Design a front-end web application based on the principles of user experience design, accessibility and responsivity
LO2	Develop and implement a static front-end web application using HTML and CSS
LO3	Maximise future maintainability through documentation, code structure and organisation
LO4	Use version control software to maintain, upload and share code with other developers.
LO5	Test and deploy a front-end web application to a Cloud platform

Pass Criteria:

ALL pass criteria must be achieved for a pass to be awarded

LO1 Design a front-end web application based on the principles of user experience design, accessibility and responsive design

1.1	Design a website that incorporates a main navigation menu and a structured layout
1.2	Design a website that meets accessibility guidelines (e.g. contrast between background and foreground colors, non-text elements have planned alt text equivalents to cater for the visually impaired)
1.3	Design the organisation of information on the page following the principles of user experience design (headers are used to convey structure, information is easy to find due to being presented and categorised in terms of priority)
1.4	Ensure that foreground information is never distracted by backgrounds
1.5	Include graphics that are consistent in style and colour
1.6	Design the site to allow the user to initiate and control actions such as pop-ups and playing of audio/video.

LO2 Develop & implement a static front-end web application using HTML and CSS

2.1	Create a website of at least 3 pages, or (if using a single scrolling page) at least 3 separate page areas, to match the design and to meet its stated purpose
2.2	Write custom CSS code that passes through the official (Jigsaw) validator with no issues
2.3	Write custom HTML code that passes through the official W3C validator with no issues.

2.4	Incorporate images that are of sufficient resolution to not appear pixelated or stretched
2.5	Code all external links to open in a separate tab when clicked
2.6	Use CSS media queries or CSS Grid/Bootstrap across the application to ensure the layout changes appropriately and maintains the page's structural integrity across device screen sizes
2.7	Use Semantic markup to structure HTML code
2.8	Present the finished website with clearly understandable site-specific content, rather than Lorem Ipsum placeholder text
2.9	Implement clear navigation to allow users to find resources on the site intuitively.

LO3 Maximise future maintainability through documentation, code structure and organisation

3.1	Write a README.md file for the web application that explains its purpose, the value that it provides to its users, and the deployment procedure.
3.2	Insert screenshots of the finished project that align to relevant user stories
3.3	Attribute all code from external sources to its original source via comments above the code and (for larger dependencies) in the README.
3.4	Clearly separate and identify code written for the website and code from external sources (e.g. libraries or tutorials)
3.5	Organise HTML and CSS code into well-defined and commented sections
3.6	Place CSS code in external files, linked to the HTML page in the HEAD element
3.7	Write code that meets at least minimum standards for readability (consistent indentation, blank lines only appear individually or, at most, in pairs)
3.8	Name files consistently and descriptively, without spaces or capitalisation to allow for cross-platform compatibility.
3.9	Group files in directories by file type (e.g. an assets directory will contain all static files and may be organized into sub-directories such as css, images, etc.)

LO4 Use version control software to maintain, upload and share code with other developers

4.1	Use a cloud-based, git-based, version control system (e.g. Git & GitHub) throughout the development and implementation process
4.2	Document the development process through descriptive commit messages
4.3	Use consistent and effective markdown formatting to produce a README file that is well-structured, easy to follow, and has few grammatical errors

LO5 Test and deploy a front-end web application to a Cloud platform

5.1	Design and implement manual testing procedures to assess functionality, usability and responsiveness
5.2	Document the testing in the README or in a separate file
5.3	Deploy a final version of the code to a cloud-based hosting platform (e.g. GitHub Pages) and test to ensure it matches the development version
5.4	Remove commented-out code before pushing final files to version control and deploying
5.5	Ensure that there are no broken internal links

All Pass criteria must be achieved for a pass to be awarded.

Merit Performance

It is expected that the learning clearly demonstrates characteristics of higher level performance as described below.

To evidence performance at Merit level, a learner will, in general demonstrate characteristics of performance at Merit level as outlined below. The learner must achieve ALL Merit criteria for a merit to be awarded.

The learner has a clear rationale for the development of this project and has produced a fully functioning, well-documented, website for a real-life audience.

The finished project has a clear, well-defined purpose addressing the needs of a particular target audience (or multiple related audiences). Its purpose would be immediately evident to a new user without having to look at supporting documentation. The design of the web site follows the principles of UX design and accessibility guidelines and the site is fully responsive.

Code is well-organised and easy to follow and the application has been fully tested, following a planned, 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.

1.1	Design a website with a flow of information layout, and interaction feedback which are clear and unambiguous
2.1	Implement a website whose purpose is immediately evident to a new user without having to look at supporting documentation.
2.2	Implement a website that provides a good solution to the user story demands and expectations.
3.1	All HTML attribute and CSS rule names are consistent in format, appropriate and meaningful
4.1	Commit often, for each individual feature/fix, ensuring that commits are small and well-defined, with messages that clearly and concisely describe the exact reason for a particular commit.
5.1	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, and user stories for a particular target audience (or multiple related audiences).
5.2	Document testing fully to include evaluation of bugs found and their fixes and explanation of any bugs that are left unfixed.
5.3	Fully document the development life cycle procedures in the README file.

ALL Merit criteria must be achieved for a merit to be awarded.

DISTINCTION Performance

At this level, a learner will have achieved all pass and merit criteria, as described above, and will demonstrate characteristics of high level performance as described below:

The learner has documented a clear, justified, rationale for a real-world application and a comprehensive explanation of how it will be developed. The finished project is judged to be publishable in its current form with a clearly evidenced professional grade user interface and interaction adhering to current practice. There are no obvious 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 resulting application is original and not a copy of any walkthrough projects encountered in the course.

Amplification (craftsmanship):

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 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

Consistency

- evident across all pages/sections and covers interactivity as well as design

Confirmation

- user 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 and class 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 and CSS rules, are consistent in format, follow standards for the language and are appropriate and meaningful

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, 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/attribute names clearly indicate their purpose
- CSS code is split into well-defined and commented sections
- Semantic markup is used to structure HTML code
- HTML and CSS are kept in separate, linked files
- CSS files are linked to in the HTML file's head element

Defensive design

- 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

Robust code

- no logic errors are found when running code
- errors caused by user actions are handled
- inputs are validated when necessary.

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

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.

Version control systems are 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

Milestone Project 2
Interactive Front-End Development

Project purpose: Presentation of interactive data

In this project, you'll build an interactive front-end site. The site should respond to the users' actions, allowing users to actively engage with data, alter the way the site displays the information to achieve their preferred goals.

Main Technologies
Required: HTML, CSS, JavaScript.

Optional: JQuery, any other JavaScript libraries, external APIs.

Value provided:

Users are able to interact with the site in their particular way, to achieve their personal goals and derive answers to their specific questions.
The site owner advances their own goals by providing this functionality, potentially by being a regular user themselves.

Project Ideas

You can either choose to build the website based on Project Idea 0, or take inspiration from the other example ideas below.

Project Idea 0

Bring your own idea(s) to life, based on providing value to users to address a specific real or imagined need. Use the relevant project assessment criteria as a guide to the minimum required functionality

Project Idea 1

- Create a site that calls on the Google Maps API and/or the Google Places API (or similar) to allow users to search for their next holiday destination.

External user's goal:

- Find the best holiday destination for their needs.

Site owner's goal:

- Get users to choose a travel package from this site and/or from its sponsors.

Potential features to include:

- Connect to an API that suits your needs, we recommend Programmable Web as a way to search for APIs.
- Display information about different cities that are potential holiday destinations.
- Allow users to select/search a city and see a map with relevant attractions, accommodation and restaurants.
- Provide results in a manner that is visually appealing and user friendly.

Project Idea 2

- Build a memory game

External user's goal:

- Have fun playing the game

Site owner's goal:

- Same as external users - make a game that you'd enjoy playing yourself

Potential features to include:

- Build a simple single-player pattern-matching memory game.
- We encourage you to create your own game, by choosing a set of visual and/or auditory patterns that increase in complexity over the course of the game and challenge the player. Games that you could look at for inspiration include Simon and Bop It.
- Provide users with clear explanations on how to play the game and with clear feedback on how they're doing at any stage.

Project Idea 3

- Build a front-end website that is themeable and customisable using JavaScript, and that remembers the customisation for the next time.

External user's goal:

- Within reason, be able to modify the website layout and style .

Site owner's goal:

- Produce a customisable website.

Potential features to include:

- Use Web APIs such as Window.localStorage to store data across browser sessions.
- Possible customisation would be dark mode toggle.
- Provide users with clear explanations on how to customise and with clear feedback on what changes are made.

Interactive Front-End Development Assessment Criteria

Learning Outcomes

LO1	Design, develop and implement a dynamic front-end web application using HTML, CSS and JavaScript
LO2	Implement front-end interactivity, using core JavaScript, JavaScript libraries and/or Application Programming Interfaces (APIs)
LO3	Test an interactive front-end web application through the development, implementation and deployment stages
LO4	Deploy an interactive front-end web application to a Cloud platform
LO5	Demonstrate and document the development process through a version control system such as GitHub

Pass Criteria:

ALL pass criteria must be achieved for a pass to be awarded

LO1 Design, develop and implement a dynamic front-end web application using HTML, CSS and Javascript

1.1	Design a web application that meets accessibility guidelines, follows the principles of UX design and presents a structured layout and navigation model, and meets its given purpose
1.2	Design interactivity for a web application that lets the user initiate and control actions, and gives feedback
1.3	Write custom JavaScript, HTML and CSS code to create a responsive front-end web application consisting of one or more HTML pages with significant interactive functionality
1.4	Write JavaScript code to produce relevant responses to user actions
1.5	Implement an interactive web application that incorporates images or graphics of usable resolution, legible, unobscured text, consistent styling, undistracted foregrounds

LO2 Implement front-end interactivity, including user forms, using core JavaScript, JavaScript libraries and/or Application Programming Interfaces (APIs)

2.1	Write JavaScript code, that passes through a linter (e.g. Jshint) with no major issues, and write validated HTML and CSS code.
2.2	Write JavaScript functions that correctly implement compound statements such as if conditions and/or loops
2.3	Write code that intelligently handles empty or invalid input data
2.4	Implement appropriate working functionality for all project requirements.
2.5	Organise non-trivial JavaScript code in external file(s) linked to at the bottom of the body element (or bottom of head element if needs to be loaded before the body HTML) and CSS code in external files linked to HTML in the head element
2.6	Write code that meets minimum standards for readability (comments, indentation, consistent and meaningful naming conventions).
2.7	Name files consistently and descriptively, without spaces or capitalisation to allow for cross-platform compatibility.
2.8.	Write code that does not generate internal errors on the page or in the console as a result of user actions
2.9	Organise code and assets files in directories by file type.

LO3 Test an interactive front-end web application through the development, implementation and deployment stages

3.1	Design and implement manual testing procedures to assess functionality, usability and responsiveness of the web application
3.2	Insert screenshots of the finished project that align to relevant user stories
3.3	Apply test procedures during development, and implementation stages and test to ensure the deployed version matches the development version
3.4	Fully document the results of well-planned manual testing procedures to assess the website's functionality, usability and responsiveness.

LO4 Deploy an interactive front-end web application to a Cloud platform

4.1	Deploy a final version of the interactive web application code to a cloud-based hosting platform (e.g. GitHub Pages)
4.2	Ensure that the deployed application is free of commented out code and has no broken internal links
4.3	Use Git & GitHub for version control of an interactive web application up to deployment

LO5 Demonstrate and document the development process through version control system such as GitHub

5.1	Document the full development cycle, with clear evidence given through commit messages, the README.
5.2	Write a README.md file in English for the interactive web application that explains its purpose and the value that it provides to its users.
5.3	Clearly separate and identify code written for the interactive web application and code from external sources (e.g. libraries or tutorials). Attribute any code from external sources to its source via comments above the code and (for larger dependencies) in the README.
5.4	Use consistent and effective markdown formatting, that is well-structured, easy to follow, and has few grammatical errors, when writing a README file.

ALL pass criteria must be achieved for a pass to be awarded

It is expected that project work submitted for this unit will demonstrate the same knowledge and skills demonstrated in Unit 1 - User Centric Development, across the grading levels and that the learner clearly demonstrates characteristics of higher level performance as described below.

Merit Performance

To evidence performance at MERIT level a learner will, in general, demonstrate characteristics of performance at MERIT level, as suggested below. However, the learner must achieve ALL listed merit criteria

The learner has a clear rationale for the development of this project and has produced a fully functioning, well-documented, interactive, web application for a real-life audience, with specific content rather than placeholders. There is a range of interactive features. Data validation, API handling (when applied) and user feedback are all evident in the code and the working application. Optionally, a range of external and internal APIs (e.g. TMDb, Google Maps, etc.) have been used to produce working features. There are no logical 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). 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.

Code is well-organised and easy to follow and the application has been fully tested, following manual testing procedures, with no obvious errors left in the code. Unit test files are present in code commits.

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.

1.1	Design a web 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
1.2	Design a web application that lets the user initiate and control actions, and gives feedback
1.3	Implement a web application whose purpose is immediately evident to a new user and which provides a good solution to the user's demands and expectations
2.1	Write code such that users who direct to a non-existent page or resource are redirected back to the main page without having to use browser navigation buttons
4.1	Commit often, for each individual feature/fix, ensuring that commits are small, well-defined and have clear, descriptive messages
5.1	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, and user stories for a particular target audience (or multiple related audiences).
5.2	Document the UX design work undertaken for this project, including any wireframes, mockups, diagrams, etc created as part of the design process, and the reasoning behind it. Include diagrams created as part of the design process and demonstrate that these have been followed through to implementation
5.3	Document testing fully to include evaluation of bugs found and their fixes and explanation of any bugs that are left unfixed.
5.4	Fully document the deployment procedure in a section in the README file.

ALL merit criteria must be achieved for the learner to be awarded a merit.

.

DISTINCTION performance

At this level, 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 level:

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, web application. When used, the learner shows a clear understanding of asynchronicity and the timing problems that can arise when accessing shared data values.

The finished project is judged to be publishable in its current form with a clearly evidenced professional grade user interface 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 resulting application is original and not a copy of any walkthrough projects encountered in the unit

Amplification (craftsmanship):

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 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 color, 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
- the user is shown progress indicators where relevant
- errors resulting from user actions are reported to the user

Consistency

- evident across all pages/sections and covers interactivity as well as design

Confirmation

- user 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 name, 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

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, js, 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 and Javascript are kept in separate, linked files
- CSS files are linked to 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 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

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**.

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:

- unit tests are included in code commits

Version control systems are 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 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

Milestone Project 3
Python and Data Centric Development

Project purpose:

In this project, you'll build a full-stack site that allows your users to manage a common dataset about a particular domain.

Main Technologies
Required: HTML, CSS, JavaScript, Python+Flask, MongoDB
Additional libraries and external APIs

Value provided:

Users make use of the site to share their own data with the community, and benefit from having convenient access to the data provided by all other members.

The site owner advances their own goals by providing this functionality, potentially by being a regular user themselves. The site owner might also benefit from the collection of the dataset as a whole.

Important Notes
No authentication is expected for this project.
The focus is on the data, rather than any business logic.

Project Ideas

You can either choose to build the website based on Project Idea 0, or take inspiration from the example ideas below.

Project Idea 0

Bring your own idea(s) to life, based on providing value to users to address a specific real or imagined need. Use the relevant project assessment criteria as a guide to the minimum required functionality

Project Idea 1

- Create an online cookbook

External user's goal:

- Find and share recipes

Site owner's goal:

- Promote a brand of cooking tools

Potential features to include:

- Create a web application that allows users to store and easily access cooking recipes. Recipes would include fields such as ingredients, preparation steps, required tools, cuisine, etc.
- Create the back-end code and front-end form(s) to allow users to add new recipes to the site, edit them and delete them.
- Create the back-end and front-end functionality for users to locate recipes based on the recipe's fields. You may choose to create a full search functionality, or just a directory of recipes.
- Provide results in a manner that is visually appealing and user friendly.

Advanced potential feature (nice-to-have)

- Build upon the required tools field to promote your brand of kitchen tools (e.g. oven, pressure cooker, etc...).
- Create a dashboard to provide some statistics about all the recipes.

Project Idea 2

- Create a jargon glossary/dictionary for a particular domain.

External user's goal:

- Find and share definitions.

Site owner's goal:

- Collect good definitions to eventually publish the dictionary in book form.

Potential features to include:

- Create a web application that allows users to store and easily access definitions for terms in a particular domain, such as web development. You may want to use Wiktionary or Urban Dictionary (potentially NSFW content) for ideas.
- Create the backend code and frontend form(s) to allow users to add new definitions to the site, edit them and delete them.
- Create the backend and frontend functionality for users to search for definitions. You may choose to also provide an alphabetical display of all definitions.

Advanced potential feature (nice-to-have)

- Provide an upvoting functionality for users to express their satisfaction with particular definitions.

Project Idea 3

- Build a book review and recommendation site.

External user's goal:

- Find books they would like to read.

Site owner's goal:

- Earn money on each book purchased via a link from the site.

Potential features to include:

- Create a web application that allows users to upload details of books, including book name, author name, link to cover image and any other relevant fields. Allow users to write comments about any book and upvote it.
- Create the back-end code and front-end form(s) to allow users to add new books and reviews to the site, edit them and delete them.

Advanced potential feature (nice-to-have)

- Add a link such as the following to each book page, such that you could conceivably earn money from people looking to buy the book: https://www.amazon.com/s?tag=faketag&k=alice+in+wonderland

Note that we do not actually encourage you to create an affiliate link, but rather want to demonstrate how this could work. Instead, for this project, we encourage you to just keep the tag value as something fake. Also, note that in general it's better to link directly to the book's page in the store, but that's a bit more difficult.

Data Centric Development

Learning Outcomes

LO1	Design, develop and implement a back-end for a web application using Python and a micro-framework
LO2	Demonstrate competence in modeling and managing non-relational data
LO3	Demonstrate competence in querying and manipulating non-relational data
LO4	Deploy a full stack web application to a Cloud platform
LO5	Identify and apply security features

Pass Criteria:

LO1 Design, develop and implement a back-end for a full-stack 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 non-relational database-backed Flask web application that allows users to store and manipulate data records about a particular domain.
1.4	Design a database structure that is relevant for your domain, consisting of a minimum of one collection.
1.5	Design and implement manual test procedures 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	Include sufficient custom Python logic to to demonstrate your proficiency in the language
1.8	Include functions with compound statements such as if conditions and/or loops in your 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 capitalisation to allow for cross-platform compatibility.

LO2 Demonstrate competence in modeling and managing non-relational data effectively

2.1	Design a data model that fits the purpose of the project
2.2	Develop the model into a usable non-relational database where data is stored in a consistent and well-organised manner.

LO3 Demonstrate competence in querying and manipulating non-relational data effectively

3.1	Create functionality for users to create, locate, display, edit and delete records

LO4 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 in English that also explains the application's purpose and the value that it provides to its users

LO5 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 security sensitive information, 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

ALL pass criteria must be achieved for a pass to be awarded

It is expected that project work submitted for this unit will demonstrate the same knowledge and skills demonstrated in User Centric Front-end Development and Interactive Front-end Development, across the grading levels and that the learner clearly demonstrates characteristics of higher level performance as described below.

Merit Performance

To evidence performance at MERIT level a learner will, in general, demonstrate characteristics of performance at MERIT level, as suggested below. However, the learner must achieve ALL listed merit criteria.

The learner has a clear rationale for the development of this project and has produced a fully functioning, well-documented, Non-Relational Database backend, 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 to 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 in English. 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).

Merit Criteria

1.1	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
1.2	Design a full stack application that lets the user initiate and control actions and gives immediate and full feedback on data processes
1.3	Implement a full stack application whose purpose is immediately evident to a new user and which provides a good solution to the user's demands and expectations
1.4	Create templates, writing code that demonstrates solid understanding of template syntax, logic and usage
1.5	Write robust code that is free of errors in all parts of the application.
1.6	Fully document the results of well-planned manual testing procedures 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
2.1	Describe the data schema fully in the README file in English
2.2	Maintain database configuration in a single location where it can be changed easily
2.3	Maintain a Procfile, requirements.txt file, etc.
3.1	All Create, Read, Update and Delete (CRUD) functionality is present and working
3.2	All Create, Read, Update and Delete actions are immediately reflected in the user interface
4.1	Commit often for each individual feature/fix, ensuring that commits are small, well-defined and have clear descriptive messages
4.2	Fully document the deployment procedure in a section in a README file, written using consistent and effective markdown formatting that is well-structured, easy to follow, and has few grammatical errors
5.1	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

All MERIT criteria MUST be achieved for the learner to be awarded a MERIT grade

DISTINCTION performance

At this level, 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 level:

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):

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 color, 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, js, 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 to 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 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 security-sensitive information 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 documentation
- 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

Milestone Project 4
Full Stack Frameworks with Django

Congratulations on making it this far! For the final project, you're going to build a web application, drawing on the knowledge and skills you have gained as a result of completing all modules.

Project purpose:

In this project, you'll build a full-stack site based around business logic used to control a centrally-owned dataset. You will set up an authentication mechanism and provide paid access to the site's data and/or other activities based on the dataset, such as the purchase of a product/service.

Main Technologies

- HTML, CSS, JavaScript, Python+Django
- Relational database (recommending MySQL or Postgres)
- Stripe payments

Additional libraries and API's

Value provided:

1. By authenticating on the site and paying for some of its services, users can advance their own goals. Before authenticating, the site makes it clear how those goals would be furthered by the site.
2. The site owner is able to make money by providing this set of services to the users. There is no way for a regular user to bypass the site's mechanisms and derive all of the value available to paid users without paying.

Project Ideas

You can either choose to build the website based on Project Idea 0, or take inspiration from the other example ideas below.

Project Idea 0

Bring your own idea(s) to life, based on providing value to users to address a specific real or imagined need. Use the relevant project assessment criteria as a guide to the minimum required functionality

Project Idea 1

- Build a fitness subscription application

External user's goal:

- To join a fitness community and purchase exercise plans and merchandise

Site owner's goal:

- Build an active community around the product based on subscription and individual payments models
- Sell
    - Exercise plans
    - Nutrition Plans
    - Nutrition and exercise products

Potential features to include:

- Now that you're a fully-fledged web developer you've decided it's probably time for you to start your very own cool, modern startup, offering a fitness community portal to a global audience. The exciting thing is the business model that you've decided upon – an active subscription-based community who are eager to purchase your customized exercise and nutrition plans and cool, branded merchandise.

- The primary areas of functionality are:
    - Community
        - Functionality that allows subscribers to update fellow members on their successes
        - User profiles containing information that map to nutrition and/or exercise plans
        - Product reviews
    - e-commerce
        - A subscription-based payment model
        - Individual item purchase capability
        - Authentication and authorisation mechanism for subscribers and administrators

Project Idea 2

- Build a site to sell your graphic design services

External user's goal:

- Users are able to purchase graphical designs to address their needs

Site owner's goal:

- Earn money for doing freelance design work

Potential features to include:

- Showcase prior work for clients, based on different kinds of requests, and the customer's testimonials.
- Allow users to order a particular graphic to suit their goals. The user would fill out a form describing their needs, including fields such as type (e.g. icon, logo, poster), size and description, get an automatic quote and then pay. You may want to include a javascript calculator to display a preview of the quote, but make sure that the final price is determined on the server, and cannot be manipulated directly by the user.
- The site owner, logging in as a special user, would be able to see a list of all orders, and then upload their completed work which would be made available to the customer.

Advanced potential feature (nice-to-have)

- The customer then has the option to either accept the result, or request a round of changes.
- Once the customer accepts, they would write a testimonial and the final graphic would be automatically included in the site's showcase.
- Build a custom js display mechanism for the gallery page - e.g. your own unique kind of carousel.

Full Stack Frameworks

Learning Outcomes

LO1	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.
LO2	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.
LO3	Identify and apply authorisation, authentication and permission features in a full stack web web application solution
LO4	Design, develop and integrate an e-commerce payment system in a cloud-hosted full stack web application
LO5	Document the development process through a git based version control system and deploy the full application to a cloud hosting platform.

LO1 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 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 backend
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	Demonstrate proficiency in the Python language by including sufficient custom logic in your project
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 management within the full web application

LO2 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	All CRUD (create, select/read, update, delete) functionality is implemented.

LO3 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, addressing a clear reason as to why the users would need to do so.
3.2	Implement login 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

LO4 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 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

LO5 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
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

All Pass criteria must be achieved for the Pass to be awarded

It is expected that project work submitted for this unit will demonstrate the same knowledge and skills demonstrated in User Centric Front-end Development, Interactive Front-End Development and Data Centric Development, across the grading levels and that the learner clearly demonstrates characteristics of higher level performance as described below.

Merit Performance

To evidence performance at MERIT level a learner will, in general, demonstrate characteristics of performance at MERIT level, as suggested below. However, the learner must achieve ALL listed merit criteria

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, manual 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.

Merit Criteria

1.1	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 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)
1.2	Produce a codebase that is fully robust
1.3	Follow a Test Driven Development (TDD) approach (for JavaScript and/or Python), demonstrated in the git commits.
1.4	Configure the project efficiently through well-kept Procfile, requirements.txt file, settings files, keep the data store configuration in a single location where it can be changed easily.
2.1	Fully describe the data schema in the project README file
3.1, 4.1	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)
5.1	Use version control software effectively to provide a record of the development process

All Merit criteria must be achieved for a Merit to be awarded.

DISTINCTION performance

At this level, 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 level:

Amplification (craftsmanship)

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 color, 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)

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, js, 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 to 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 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. jslint.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**.

- 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

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

Django:

- 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

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 testing 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, 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
- 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

Recommended structure for a README.md file:

As a mandatory part of the submission, your project must have a readme file named README.md, in markdown format, that will describe all aspects of your project.

The link below provides an official example of an expected structure for your readme file

[Example README.md template]

NOTE:

While it is a requirement for your README to be in English, we also strongly recommend that your site is in English too. The assessment will be conducted in English so to ensure that the functionality and navigation can be fully assessed, we recommend that the site is in English. If the site is in a language other than English, at a very minimum you should ensure you have correctly set the `lang` attribute in the HTML to the character code of the web apps content language. This will allow the browser to attempt a translation into English.

Plagiarism

Plagiarism, as defined by the Oxford dictionary is "*the practice of taking someone else's work or ideas and passing them off as one's own.*" It is a serious academic offence for which there are serious consequences.

It is acceptable to use and reference others' code however it is an academic plagiarism offence if **any** piece of work which is not entirely the student's own is not correctly referenced or acknowledged. All student projects submitted will be reviewed for plagiarism. This includes checking code comparison tools, plagiarism software, review of git commit history and other mechanisms.

It is the responsibility of each student to ensure that any direct or indirect inclusion of the work of others is fully and adequately acknowledged. We appreciate that plagiarism may be unintentional however it will still be treated as an offence. The Tutoring team can answer any plagiarism related queries, but as a general rule, if in doubt, include attribution of all sources.

Students are encouraged to ask mentors, tutors and their peers for advice about their project work but any submission should not include any code written by others, unless it is explicitly credited to them. Failure to correctly credit code that a student hasn't created themselves will be considered plagiarism and will result in a failing grade. Blatant or repeat offences of plagiarism will not be tolerated and will result in stringent penalties being applied, including removal from the course.

Assessment Policy

The programme is assessed by 4 milestone projects. Some modules have challenges for you to complete however these are not graded. All 4 projects must be separate submissions, meeting the specified project brief requirements. All 4 projects must be passed before the Diploma can be awarded.

Your project is due at 12.00 noon (Irish time) on the specified submission date. Please contact Student Care if you are unclear what your submission date is.

Any project not submitted on time will be considered a fail. The project will be accepted up to 10 days after the original submission date, subject to payment of a late submission fee of €150. The grade will also be capped at the pass mark. Code Institute reserves the right not to accept any projects submitted after 10 days. Repeated late submissions will not be tolerated.

Extenuating circumstances are unforeseen significant events which may affect your ability to complete the course/submit your projects on time. Such circumstances include serious illness, hospitalisation, immediate family bereavement etc.

Supporting documentation for such circumstances must be provided for the extenuating circumstance to be considered. Foreseen events such as holidays, work commitments, weddings etc. are not considered extenuating.

If you find yourself in this situation, please contact Student Care to discuss it. These instances are treated on a case by case basis. Supporting documentation e.g. medical certificate will be required for any additional time to be granted.

Resubmissions

If a project submission does not meet the required standard, you may resubmit your project to achieve a pass. Student Care will liaise with you to arrange the timeframe for a resubmission.

The grade for any project resubmitted will be capped at the pass mark. You will not be informed of the actual result. The maximum number of resubmissions per project is two. The resubmission fee is €50. A student whose project has received a pass mark is not permitted to resubmit their project with a view to achieving a higher grade.

Appeals

Where a student feels they have been unfairly marked or that the marking is inconsistent with the assessment criteria, they may appeal their grade. The appeal must be requested in writing to Student Care within 2 weeks of the grade being released. In doing so, you must also outline the grounds on which you are appealing the awarded grade. The project in its previously submitted format will be graded by a different assessor. You may not make any changes to your project (source code or live version) while the appeal process is taking place. The result of the appeal, which may be higher or lower than the original grade will be the final grade awarded. There is a €50 fee for the appeal which will be refunded if the outcome of the appeal is a higher passing grade. The appeal process will be completed within 5 weeks.

Final Grade Calculation

The four projects all contribute to your final grade. This is calculated from the points awarded for a pass/merit/distinction multiplied by the number of credit points per module. These are depicted in the table below. The maximum points available are 480.

.

Module	Credit Points
User Centric Front-end Development	10
Interactive Front-end Development	15
Data Centric Development	15
Full Stack Frameworks	20

Points per Grade	
Pass	4
Merit	6
Distinction	8

Grade Bands	
Pass @ 50%	240
Merit @ 65%	312
Distinction @ 80%	384

Sample	Credit Points	Grade	Points	Total
User Centric Front-end Development	10	Pass	4	40
Interactive Front-end Development	15	Merit	6	90
Data Centric Development	15	Merit	6	90
Full Stack Frameworks	20	Pass	4	80
				300
			Final Grade	Pass

code institute

University Credit-Rated	Industry-Approved Curriculum	Flexible Learning Options	1:1 Tutor & Mentoring Support	High Success & Satisfaction Rating
✓	✓	✓	✓	★★★★½

+353 1 539 7973 info@codeinstitute.net