Full Stack Frameworks with Django Milestone Project

PLANNING DOCUMENT

Paul Bennett

Table of Contents

"Full Stack Frameworks Milestone Project"

This is the milestone project that I have created for the "Full Stack Frameworks with Django" module, which is part of "Full Stack Web Development Course" offered by Code Institute via Learning People.

Full Stack Frameworks Milestone Project Brief

I have listed below the brief of this milestone project.

Congratulations on making it this far!

Now for the final project, you're going to build a web application, and you will draw on the knowledge and skills you have gained as a result of completing lessons in all nine modules. You can either choose to use the example brief below, or you can use your idea for the website.

BUILD AN ISSUE TRACKER

Now that you're a full-fledged web developer you've decided it's probably time for you to start your very own cool, modern startup, offering the extremely awesome UnicornAttractor web app to your users. It's really really amazing, but we don't care about it at all in this project. The exciting thing is the business model that you've decided upon – you chose to offer the service and bug fixes for free, but ask for money from your users to develop additional features.

The primary entity in the Issue Tracker is a ticket describing a user's issue, and similar to Github's issue tracker, you should allow users to create tickets, comment on tickets, and show the status of the ticket (e.g. 'to do,' 'doing,' or 'done'). As mentioned, issues come in two varieties – 'bugs' (which you'll fix for free, eventually), and 'features' which you'd only develop if you're offered enough money. To help you prioritize your work, your users will be able to upvote bugs (signifying 'I have this too'), and upvote feature requests (signifying 'I want to have this too'). While upvoting bugs is free, to upvote a feature request, users would need to pay some money (with a minimum amount of your choice) to pay for your time in working on it. In turn, you promise always to spend at least 50% of your time working on developing the highest-paid feature.

To offer transparency to your users, you decide to create a page that contains some graphs showing how many bugs or features are tended to on a daily, weekly and monthly basis, as well as the highest-voted bugs and features.

Add any additional pages that would help you attract users to the Issue Tracker (and have them pay you well). To make the users participate as much as possible in your online community, make sure that your UI/UX is sublime. Feel free to add additional features, such as a blog, extra perks for active participants, etc.

If you want to have some more fun with this, feel free also to add pages describing your fictional UnicornAttractor application.

And of course, as this project is going to be the lifeblood of your company, it's essential that new developers that join the company will be able to get up and running as quickly as possible. Documentation is the best way to achieve this.

CREATE YOUR OWN PROJECT

If you choose to create your project outside the brief, the scope should be similar to that of the example brief above. If you want some ideas, please ask your mentor for advice and direction.

Full Stack Frameworks Milestone Project Guidelines

I have listed below the guidelines when developing the milestone project.

- Build a web app
  - Must fulfil some actual (or imagined) real world need.
  - May be domain specific.

- Project must be a brand-new Django project
  - Composed of multiple apps.
  - An app for each reusable component.
  - At least one of your apps should contain e-commerce functionality using Stripe.
    (This may be a shopping cart checkout, subscription-based payments or single payments etc)
  - Include at least one form with validation that will allow users to create and edit models in the backend
  - The project needs to be connected to a database (MySQL or Postgres) using
    (Django's **O**bject-**R**elational **M**apper ORM) which interacts with your database.

- Project should include an authentication mechanism
  - Allowing the user to register and log-in.
  - A good reason why user needs to for example to persist their shopping cart between sessions
    (otherwise it would be lost).

- Project Testing
  - Test Project extensively, making sure the following
    - No unhanded exceptions are visible to the users, under any circumstances.
    - Use automated Django tests wherever possible.
    - For JavaScript code, consider using jasmine tests.

- Project logic **must be created** using the following
  - (Css3,Django,Html5,JavaScript,Python3,Templating Language).
  - (Git & GitHub) for version control (new functionality should have separate commit).
  - (Heroku) to deploy final version of code.

- User Interface (UI)
  - Should be responsive.
  - Use either media queries or responsive framework such as Bootstrap.
  - Make sure that website looks well on all commonly-used devices.
  - The app should have a great user experience.
  - Frontend Should contain JavaScript logic to enhance user experience.
  - Whenever relevant Backend should integrate with third-party Python / Django packages such as Django Rest Framework.

- Create a '**Credits**' section in the **Readme.md** file to include the following,
  - **Content** i.e. text copied from xxx
  - **Media** Where they were obtained from.
  - **Acknowledgements**

- Create a '**Deployment**' section in the **Readme.md** file to include the following,
  - How to **deploy project** to hosting platform (Heroku).
  - Detail of differences between Deployed & Development Versions of code.
    - Different values for environment variables.
    - Different configuration files.
    - Separate Git Branch.

- Create a '**Testing**' section in **Readme.md** File to include the following,
  - **Summarizing** your approach to testing
  - Provide **pseudocode** written to develop your TDD tests
  - If **manual tests** are run to ensure websites functionality works correctly.
  - If tests are **not working** as expected document
    - Your expected output should have been.
    - Reasons why the test(s) could have been failing.

Full Stack Frameworks Project Checklist

I have listed below the sections from the Checklist of the milestone project which should be incorporated into the project and checked before final submission using the Checklist.

- **Application Features**
    - App logic
        - App Logic Each of the Django apps that form the project is built correctly and functions well
        - Each app matches a natural aspect of the project, with no app being too small or too big.
        - All files and components in an app make sense within the context of that app.
    - Authentication and Security
        - Any functionality that requires log-in is available only to logged-in users (and anonymous users are redirected to login)
        - The log-in and registration pages are only available to anonymous users (and logged-in users are redirected out automatically)
        - Users are never asked to submit data that the site already has, e.g., if you have a Contact Us form, don't ask for the email of a logged-in user.
        - If different users have different permissions, then check their access levels are appropriate. (e.g., a non-admin user should not be able to edit another user's post).
    - Cross-App logic
        - The Django apps that form the project are designed to work well with one another.
        - Application urls are set up in a consistent manner
        - Any data in the models that is relevant to multiple apps is shared, rather than duplicated.
    - Ecommerce
        - The site contains e-commerce functionality, which works well with a test credit card (4242 4242 4242 4242).
        - Successful and unsuccessful purchases are indicated to the user with a helpful message.

- **Layout and Visual Impact**
    - Colour scheme and Typography
        - There is sufficient contrast between background and foreground colours
        - The colour scheme used on the site consists of a palette of colours that work well together.
        - The typefaces used complement one another.
        - All text is legible; particular attention to legibility is maintained when text formatting effects are in use.
        - Text is never obscured by images or colours.
    - Image Presentation
        - Graphics are consistent in style and colour.
        - The background never distracts from the foreground information.
        - All kinds of multimedia content used in the project work well on the different popular browsers.
        - Whenever needed, multiple alternative file types are used.
        - Images always maintain their original aspect ratio when the screen is resized (crop don't stretch).
        - All images are of sufficient resolution to not appear pixelated.
        - Image files are not bigger than is needed - full-screen images are under 3MB, while smaller images are <500kB.
        - If any larger files are being loaded, there is a progress indicator.
        - For large video/audio resources, prefer an external hosting platform (e.g., YouTube, S3….)
    - Responsive Design
        - All page elements look well on screens as small as 360 pixels wide and as big as 3840 pixels wide (4K).
        - The site uses Bootstrap grid sizes or CSS3 media queries to ensure the layout changes appropriately and reflows when the screen is resized.

- **Code Quality**

  - Appropriate use of Code
    - There is no unneeded complexity, as well as no commented-out code.
    - Code is indented in a consistent manner to ease readability.
    - Functions are used for reuse of identical/similar code sections to encapsulate implementation detail and promote abstraction.

  - Appropriate use of CSS3
    - Your project includes sufficient custom written CSS code to demonstrate your proficiency in the language.
    - Your CSS passes through the official (Jigsaw) validator with no issues.
    - Your CSS is split/indented into well-defined and commented sections.
    - CSS code is kept in external file(s) that are linked to in the HTML file's head element.
    - CSS classes are used effectively and there is no unnecessary duplication of properties.
    - Code is indented in a consistent manner to ease readability.

  - Appropriate use of Django
    - The Django file structure is consistent and logical, following the Django conventions.
    - All logic appears in the component where it is best suited. e.g., data handling logic is in the models, business logic is in the views
    - There is no unneeded complexity.

  - Appropriate use of HTML5
    - our HTML passes through the official validator with no issues.
    - Semantic HTML5 elements are used whenever this makes sense.
    - All non-text elements have a text equivalent for the visually impaired (e.g. alt attributes in img elements).
    - Code is indented in a consistent manner to ease readability.

  - Appropriate use of JavaScript
    - Your project includes sufficient custom JavaScript logic to demonstrate your proficiency in the language.
    - In particular, it includes functions with compound statements such as if conditions and/or loops.
    - Your JavaScript code passes through a linter (we recommend jshint.com) with no major issues.
    - Check that there are no errors in the console during all interaction with the site.
    - The code is indented in a consistent manner to ease readability, and there is consistent use of empty lines.
    - All non-trivial JavaScript code should be kept in an external file(s) linked to at the bottom of the body element (or bottom of the head element if needs loaded before the body HTML).

  - Appropriate use of JavaScript APIs (if used)
    - our project includes JavaScript code that uses external and/or internal APIs (e.g. TMDb, Google Maps, Canvas, dc.js, etc.) in an effective manner.
    - There is a clear separation between input logic, processing, and presentation logic.
    - There are no bugs or unneeded complexity
    - The code is robust against race-conditions - the code never relies on the duration of specific processes
    - The API-related code demonstrates a solid understanding of asynchronicity and call-backs/promises.

  - Appropriate use of Python
    - Your project includes sufficient custom Python logic to demonstrate your proficiency in the language
    - it includes functions with compound statements such as if conditions and/or loops.
    - Your Python code is consistent in style and preferably conforms to the PEP8 style guide
    - Check your Python indentation is valid.

  - Appropriate use of the template language
    - Your template code is valid and well organised.
    - Your template code demonstrates solid understanding of template tags and template inheritance.
    - There is no unneeded complexity

- **Software Development Practices**

  - Comments
    - All code files include clear and useful comments, wherever they are relevant.
    - Consider your intent, the reasoning and any trade-offs behind your code.
    - Your comments explain the “why” rather than the “what”.

  - Data Store Integration
    - The project makes good use of a data store (database, or data files such as json) to maintain data in a consistent and well-organized manner.
    - The contents of the data store are persisted between requests.
    - As Heroku does not support media file upload, have you stored them on an external service?
    - The data store configuration is kept in a single location and can be changed easily.
    - Regular users are not able to access the data store directly without going through your code.

  - Deployment Implementation
    - There are well-kept Procfile, requirements.txt file, settings files, etc., including separate versions/branches of these if relevant.
    - All secret keys are hidden in environment variables or in files that are in .gitignore.
    - Debug mode (e.g. in Flask or Django) is disabled for the deployed version.
    - Your site is deployed online on GitHub Pages, Heroku or any other system with full functionality.
    - The deployed version is identical to the development version except as explicitly mentioned in the deployment section in the documentation.

  - Deployment write-up
    - The deployment procedure is fully documented in a section in the readme file.
    - Any differences between the development code and deployed code are fully explained.
    - Your README describes the deployment procedure including settings files, environment variables, dependencies and any other differences between the dev and live versions.
    - If you have created an automated script to help deploy the project, you should include it (or link to it) in your write-up.

  - Directory Structure and File Naming
    - Your project’s files are named clearly and consistently and located in appropriately named directories.
    - Whenever relevant, files are grouped in directories by file type (e.g., a static directory will contain all static files and may be organized into sub-directories such as CSS, images, etc.)
    - There is a clear separation between your files and any external files (for example, library files are all inside a directory named 'libraries').
    - File names are descriptive and consistent. For cross-platform compatibility, file and directory names shouldn’t have spaces in them and should be lower-case only.

  - Readme file
    - The project includes a readme file named README.md which is intended as an introduction for other developers who would like to use and contribute to this project. Note that submitting a project without a readme is an immediate fail.
    - The readme file describes the project's components, the technologies used and any other important details.
    - The readme is well-structured and easy to follow.
    - Your readme file is written in markdown and uses markdown formatting consistently and effectively.
    - The readme describes the project’s purpose, components, technologies, and all other important details.

  - Testing Implementation
    - You have conducted enough testing to convince the assessor that you legitimately believe that the site works well.
    - You conducted manual testing of your HTML/CSS for usability and responsiveness.
    - If using JavaScript or Python, you have created automated unit tests.
    - If using automated tests in JavaScript or Python are, they effective and meaningful? Ensure no useless tests.
    - If you have used test driven development (TDD) for JavaScript or Python, ensure this is demonstrated in your commit log.

  - Testing Write-up
    - Your testing (both manual and automated) is well documented either in the README or a separate file.
    - Your write-up discusses any interesting bugs found and their fixes.
    - Your write-up mentions and explains any bugs that were left unfixed.

  - Version Control
    - Your code is managed in git, with a separate well-named commit for each feature/fix.
    - You avoid very large commits, because this makes it harder to understand your development process and may lead the assessor to suspect plagiarism.

- **Usability and Real-World application**

  - Defensive Design
    - A customer is not be able to break the site by clicking buttons out of the expected order or by providing any unexpected inputs.
    - All forms intelligently handle empty or invalid input fields.
    - Navigating between pages via the back/forward buttons can never break the site.
    - This includes unexpected actions such as navigating back to the login page after already being logged in.
    - User actions should not cause internal errors in the console
    - Clear feedback to the user is given for any action disallowed by the developer.

  - Ease of Use
    - It is easy and straightforward for a new user to figure out how to use your site without having to read any documentation
    - There are no broken links. Have you had others (for example, family members, friends and/or other students) try out your site and they all said so?
    - The site is intuitive to use and never confuses the user or surprise them in a negative way.
    - The user has full control of their interaction with the project and at no point needs to “fight” it.
    - The site avoids aggressive automatic pop-ups and auto play of audio; instead of allowing the user to initiate such actions.
    - All input elements are clearly labelled and provide placeholders and default values whenever relevant.
    - The project follows common and consistent UI/UX conventions - there are plenty of online resources you can take inspiration from, such as Good UI.

  - Information Architecture
    - All information displayed on the site is presented in an organised fashion with each piece of information being easy to find and none feeling out of place.
    - Headers are used to convey structure - each section has a header that’s easy to see and clear to understand.
    - The written language used on your sites is straightforward for the user to follow.
    - Whenever relevant, the site provides interactivity to make the information easier to consume.

  - Navigation
    - All resources on the site are easy to find, allowing users to navigate the layout of the site intuitively.
    - The site’s navigation is consistent and reasoned.
    - There is never a need to use the Back button to move through the site.
    - For any external links, the target=”_blank” attribute is used.

  - Project Purpose
    - The project has a clear, well-defined purpose addressing the needs of a particular target audience (or multiple related audiences).
    - The project’s purpose is evident to a new user without having to look in the documentation.
    - The project’s documentation provides a clear rationale for the development of this project.

  - Suitability for purpose
    - The site's design, as implemented, provides a good solution to the users' demands and expectations.
    - A regular user would not immediately think "there's a much better way to do this" about any part of the project.

  - UX Design
    - The project's documentation describes the UX design work undertaken for this project and the reasoning behind it.
    - Any wireframes, mock-up’s, diagrams etc... you created as part of the design process are included in the project

What I would like to accomplish

My Project Task List

This is a breakdown of the tasks that need to be performed to achieve the tasks set out in the project brief guidelines.

NO.	PROJECT TASK DESCRIPTION	PROJECT TASK OBJECTIVE
1	**Create** Web Application.	**Create** New Ecommerce multi app Django web application.
2	**Include** Ecommerce functionality	**Create** app(s) to use Stripe (Shopping cart checkout)
3	**Include** Form validation	**Create** form(s) to allow users to Create / Edit models in backend.
4	**Include** Graphs	**Create** graphs of most common parts by category, most common by vehicle, etc.
5	**Include** Up / Down voting	Users could upvote/downvote i.e. parts they like/dislike, sorted by most popular i.e. parts, most popular i.e. cars ...
6	**Include** User registration and authentication	**Allow** users to be able to register and log-in.
7	**Include** Version Control	Use Git & GitHub.
8	**Test** code	Make sure code is extensively tested and documented in README.md
9		
10		
11		
12		
13	Backend Logic	**Create** backend to include (whenever relevant) third-party Django / Python framework.
14	Database connection	**Connect** DB using Django Object-Relational Mapper (ORM)
15	Frontend Logic	**Create** frontend to include JavaScript logic.
16	User Interface (UI)	**Create** UI to include Bootstrap or media queries.
17	**Deploy** Final Version	Use Heroku as hosting platform.
18	**Ensure** there's a README.md	A project submitted **without** a README.md file will FAIL.

My Project Overview

I decided to

- Create a Triumph spitfire Classic car parts Ecommerce Web shop based on project brief.
- Build the charts using Highchart.js
- Build the database using SQLite3 /Postgres as per the Ecommerce mini project in the "Full Stack Framework with Django" module.
- Build User registration and authentication processes as per the Ecommerce mini project in the "Full Stack Framework with Django" module.
- I did some google research and will use https://www.scparts.co.uk/sc_en/british-cars/triumph/triumph-spitfire-mkiii-mkiv-and-1500-1967-1980.html as classic car parts reference site.

Media used

Media	Sourced Location
Bootstrap theme (Sandstone)	https://bootswatch.com/sandstone/
Home page background image	https://commons.wikimedia.org/wiki/File:Triumph_Spitfire_MKIV_colors.svg
Icons	https://fontawesome.com
Parts images	https://www.scparts.co.uk/sc_en/british-cars/triumph/triumph-spitfire-mkiii-mkiv-and-1500-1967-1980.html
Categories Icons	https://www.moss-europe.co.uk/shop-by-model/triumph/spitfire
About Page images	https://unsplash.com/
Home Page images	https://en.wikipedia.org/wiki/Main_Page