Title of the project

Online Hardware Store

Members

Eldin Delic Hodzic - X00153243
Dylan Byrne-Carr - X00155165

Supervisor

Patricia Magee

Table of Contents

Introduction

This document outlines the process of creating the project from Iteration 1 to the final prototype. All pages included in this document elaborate on each member's contribution to the project and what they did in detail. Each iteration will show the progress of the project and how much we improved on each iteration.

In this document, we will also talk about the problems that we encountered and how we overcame the problems. Also, what we could have done better and what we can improve on in the future.

Eldin’s part

My part mainly consisted of the authentication and user interaction. When we created our project, the first thing we planned on doing was accounts, authentication and security. I took this role and created the sign up, sign in form. The sign up form was made using django-allauth which was a pip that allowed us to create the forms.

In the sign up form, a user is required to enter an email, username and password. The password has some security, for example if a user typed in a password that is similar or common to the username or email, the user would get an error saying that it’s too common, or if a password was simple such as 12345, it told the user to add letters, symbols etc.

Once the user created an account, we added a 2FA (2 Factor Authentication) where the user would have to verify the email they entered, for security reasons. Due to the firewalls at the time when were in college doing this, we couldn’t send it to the actual email that the user entered but instead made it so that the email, and link to verifying the email would display on the running terminal in Visual Studio Code. Once the user copied and pasted the code, they were prompted to verify the email.

Sign in form was quite simple. Django-allauth also played a part in this, as we added a feature, ‘forgot password’. Thanks to Django-allauth, users forgot their passwords, they can request to get their password changed. Same as the email, the link to changing the password will be displayed in the terminal once the user clicked to change it. The user will then be redirected to a new page and prompted to enter a new password and re-enter it.

Now that, authentication and accounts were done, we moved onto user friendly functions, such as a comment section. We used a software known as Disqus, which allowed people to have a comment/messaging system for their websites. We created an account and wrote the code for it to display on the website. Users can only post messages or comments if they were signed in. Users could also add reactions.

Lastly, I added an email contact form. The purpose of this form was that if customers/users had questions about our service or products, they can email us directly. This contact form was created using our administration database, Wagtail. Wagtail provides a unique user interface, which allows users to create web pages and forms. Using Wagtails forms, I created the email contact form. When I created the form on Wagtail, I had to add some code to the views and templates for it to work. When a user sends an email, the email, along with their message is displayed in the terminal and on the database. On the database, we can filter the emails using the dates of when the emails were sent. We also get an option to download the emails as .csv files so that we can do data analysis on them.

Dylan’s Part

My role in the project consisted mostly of adding the CMS(content management system) Wagtail. That allowed us to manage the site with ease and really got us going. It was the first big step we made and allowed us to create pages, products etc with ease. The best part of it was that it also went into the sqlite database which would allow us to use that for stuff like data analysis. Wagtail also came with its own stylesheets which made designing the page so much easier and we didn’t have to worry as much about CSS. It worked just like bootstrap.

The Wagtail admin interface allowed us to create users with specific permission, manage and public live and draft versions of pages we were about to submit and eventually lead us into our integration with Snipcart. Snipcart allowed us to make products, manage the cart and payments with ease. All you need to make a product is a button with a few classes in it which go back into the snipcart scripts which handled order validation and even a domain crawling system which tracked our products. However I had a lot of trouble with the products. The domain crawler sometimes wouldn’t work on localhost meaning i had to host the site live somehow. I found a handy program called ngrok which gave us a url which could be accessed online as long as the server was running. In the Snipcart configuration, I had set it to find the URL 127.0.0.1 but being localhost it wouldn't work. The new URL which was a forwarding of our localhost allows snipcart to validate items but more importantly allows anybody to access the site. I tested it on my phone from inside my network and outside and it worked perfectly. The link only stays active for 8 hours however so i have to refresh it every now and then but its only one command to do so.

Finally i added a live chat feature using asynchronous websocket, meaning a full duplex TCP connection between two browsers to send messages to each other. I set up the rooms to be named after the username of the person who initiated the conversation and all it takes it to put the username at the end of the URL and you're talking to them. I needed to make a routing.py file which basically acts as django's urls.py. I made some tests on this and most of the time they would succeed but sometimes failed, up to now i'm unsure why but the site does work as intended.

The products work by creating a button and inside it creating classes such as “data-item-id”. Snipcart takes the id and scans your site for a match of it and pulls all the information including the price for it. I can add custom images and customisation options to a product using custom fields which can work as a dropdown, slider, checkbox etc but for simplicity of ordering and in a real world scenario more conversions i chose to just use drop down.

It was mainly used to customise colour but our main product is the build your own PC part where you select different choices of components from drop downs and add it into one product.

Iteration 1

The first iteration of our project was the planning stage. We planned on what we were going to start on first in the project and what were the essential functions that had to be done for Iteration 1. The first SCRUM Workshop helped us with this as we were told what were the main functions and what had to be done. Accounts, payment and cart were the essentials.

We started off by making the accounts. We used Django's default forms to make our sign in and sign up forms. We used crispy forms to make the forms look neat and tidy. At the time of making the forms, we didn't have 2FA (2 Factor Authentication) on our sign up forms because we mainly concentrated on just creating accounts and signing in and out.

As we were doing our project, we started to run into problems. For the cart and payments, we found a new system called Snipcart. Snipcart was a unique cart that had its own unique cart system. It allowed the user to change the quantities, remove items and also had its own voucher system. The voucher system was provided by Snipcart which was an asset of our database, Wagtail CMS. When we finished coding Snipcart along with the payment and vouchers, we tested it to see how it would work.

We created a product, gave it a random name and price. The product was made using Wagtail CMS, it gave us all the options such as image, name, price, description and more. When we added the product to our cart, everything was going well. The quantity, removing the product until we proceeded to the checkout. The payment information all worked but Snipcart couldn't detect the product. We didn't know at the time how to fix the problem, so we spent hours trying to find a solution, and we couldn't. We had no choice but to move on because of the amount of time we had left before the deadline.

As we were approaching the deadline, we decided to do a test run. As we did a test run, we started at the accounts. We went to test to see if sign in works, and we ran into an error. The error didn't allow us to sign in, some of the text boxes for filling in your username were duplicated. We designed a page so that when the user created an account, the page would display, 'you successfully created an account', but it went wrong. When the user created the account, they would be stuck on that page and every time they try to sign up a different account, they would be brought to that same page. So, the sign up was broken.

Sign in was still working as all it required was a username and password and you were redirected to the website. We didn't know how to fix the sign up either. We looked everywhere to try and find a solution and we couldn't. Until we discovered Django-allauth. Django-allauth is a django asset that concentrates on authorization, accounts and security. We also figured out that Wagtail can provide forms for sign in and sign out. So we decided to change our forms completely. We went from django forms to Wagtail CMS forms.

The forms provided by Wagtail were more detailed and layed out better than django forms. We also found an option to include authorization, such as 2FA (2 Factor Authentication) and Forgot Password, which we did later in other iterations. Changing to Wagtail CMS forms solved our problem and sign up was working again. As we were running out of time, we couldn’t do much testing but we still included as many tests as we could for the functions using Python. We created a test function for each functionality in our project that we did so far, and it all ran successfully.

Iteration 2

We wasted a lot of time during iteration 2. We were completely stuck on something as fundamental as getting the payments working that we did not have a lot of time to work on other stuff. We tried multiple different methods including Skrill, Stripe and PayPal but were unable to get them to work with Snipcart. Eventually we decided to work on other stuff for the time being and that was when Eldin started to work on a comment system. We started to prioritise customer interaction on the site and gave users as much flexibility as we could. Eldin incorporated a comment system that allows people to comment on a page, we used it under every product we had at the time being, at the bottom of our product.html page.

After that i found out that snipcart had a new GUI version 3 and i tried it out but wasn't able to get it working. I reverted back to version 2 and we were basically back to square one. On many occasions I would fire up vs code to add a new feature but would fall into trying to fix payments again so during iteration 2 my only piece of functionality that I added was the finishing touches to django-allauth which allowed authorization and authentication for Eldin as he was having trouble with it. Finishing this finally allowed us to create a user without the infinite redirect loop they would get stuck in.

After that we looked further into the payments system and discovered that Snipcart has an order validation method for security reasons, to stop users from changing values such as the price. We knew what we had to do but didn’t know how to do it,

Iteration 3

Iteration 3 is the final iteration where we absolutely need everything to be working perfectly. Before working on payments we needed more features. Eldin created a contact form which would send emails to the admin for them to reply to. This would show up in the admin page on the website. It was done using forms and models and worked perfectly although it was hard to see when someone had sent an email without manually going into it. That isn't the biggest issue however so we let it go.

I worked on a live chat which was intended to work as a help desk / customer support chat. It uses websockets which allow for full-duplex TCP communication between two users. The way it works is that when you go into the live chat, you are met with a text box containing your username. My plan was to have this text be automatically submitted but wasn’t able to do that so the user just has to press enter. They are put into a chat room where the id of the chat room is the initiator’s username. Admins again would have to find this manually but inside it they could connect and talk in real time to customers. It was asynchronous meaning it would not wait for a response code before continuing which allowed for much faster chat times. It utilised some JSON functionalities to allow for this although no JSON files were needed.

After finishing on this re-made the entire product system. Instead of being a page made inside the Wagtail admin, it was simply a button containing snipcart classes allowing for much simpler programming. The classes would activate the cart and give it information such as price and unique ID of the product. It was here that we realised what the issue with the payments was. Wagtails validation would request the exact url of where the buy/add to cart button was located. The biggest issue was that the validation could not actually reach the site because it was only available locally. I found a neat little program called ngrok which forwarded my localhost address to a live url; accessible online for anyone to use. This url expires every 4 hours but it’s very easy to refresh so no issues there. However i will need to keep this going manually during the grading period. Fixing this led me to an issue with authorisation.

Snipcart’s validation process would scan the website for the id we specified but we had that behind an “{{ if request.user.is_authenticated }}” block so the validator wasn't able to see it. Therefore the only way we could get the payments to work was through allowing purchasing without being logged in. I kept the chat and contact forms behind the block so that we could respond to the right people.

After adding that i made some custom images, only slightly based off real products, to add some realism to the products and tried to fix up the design but came short on time and had to leave it as it was. That led the site into its final state and how we will present it.

Conclusion

In conclusion, given the circumstances we were in, we still managed to do our best in completing this project. There are some parts of the project where we could have done better or where we could have added a little more but nevertheless, we managed to get most of the project to a decent standard.

While doing this project, time management is definitely something we both picked up on, and working under pressure. As we both couldn’t work on the project at the same time, we had to cooperate at times and inform each other when we were finished with our part so that the other can start.

Problem solving played a big part in this project. When we added new code or functions in our project, we got familiar with the types of bugs and errors that would come up frequently, so we already knew what the problem was and we could fix it as soon as possible. It was challenging but we overcame that challenge and managed to complete this project. It is safe to say we both put in an equal amount of effort and time into the project.

Roles in the project document

Introduction - Eldin Delic Hodzic

Eldin’s part - Eldin Delic Hodzic

Dyaln’s part - Dylan Byrne Carr

Iteration 1 - Eldin Delic Hodzic

Iteration 2 - Dylan Byrne Carr

Iteration 3 - Dylan Byrne Carr

Conclusion - Eldin Delic Hodzic

Student Compliance with Academic Integrity

All students are expected to complete their courses in compliance with University regulations and standards. No student shall engage in any activity that involves attempting to receive a grade by means other than honest effort, for example:

1. No student shall complete, in part or in total, any examination or assignment for another person.
2. No student shall knowingly allow any examination or assignment to be completed, in part or in total, for himself or herself by another person.
3. No student shall plagiarize or copy the work of another and submit it as his or her own work.
4. No student shall employ aids excluded by the instructor in undertaking course work.
5. No student shall knowingly procure, provide, or accept any materials that contain questions or answers to any examination or assignment to be given at a subsequent time.

6. No student shall procure or accept assignments from any other student from current or prior classes of this course.
7. No student shall provide their assignments, in part or in total, to any other student in current or future classes of this course.
8. No student shall submit substantially the same material in more than one course without prior authorization.
9. No student shall alter graded assignments or examinations and then resubmit them for regrading.
10. All programming code and documentation submitted for evaluation or existing inside the students computer accounts must be the students original work or material specifically authorized by the instructor.
11. Collaborating with other students to develop, complete or correct course work is limited to activities explicitly authorized by the instructor.
12. For all group assignments, each member of the group is responsible for the academic integrity of the entire submission.

N.B to be filled out by each member of the team.

By including my name in the form below, I declare that I understand and will abide by the University Regulations and Policies covering Academic Integrity. I accept that each member of the group is responsible for the academic integrity of the entire submission. I will retain a copy of this agreement for future reference.

Student Name (1):	**Eldin Delic Hodzic**	**Student No.:**	**X00153243**
Student Name (2):	**Dylan Byrne-Carr**	**Student No.:**	**X00155165**
Student Name (3):	**N/A**	**Student No.:**	**N/A**
Student Name (4):	**N/A**	**Student No.:**	**N/A**
Module title:	**Project**		
Programme Title:	**Computing with Software Development (Hons.)**		
Date: DD/MM/YYYY	**27/04/2020**		