

Investigation into the Omnichannel Experience that Post Pandemic Education can Present

LM118 – Bachelor of Engineering in Electronic and Computer Engineering

Final Year Project

Final Report

Emmett Lawlor

18238831

John Nelson

<Submission Date>

Abstract

Replace this text with the final report abstract. This is to be a one page summary of the report that identifies:

1. The background and motivation for the project,
2. What was achieved in the project,
3. The structure of the report.

The exact format of the report is to be discussed and agreed with the project supervisor. In this template, the following is used:

1. 1.5 line spacing apart from the title page,
2. In the main body of the text, 12-point Calibri font is used,
3. The use of underlining text is avoided,
4. The report is primarily aimed at being read on a computer screen,
5. Text in main body is justified left and right,
6. The report presented so that if it was printed, it would be printed 2-sided. This means that to ensure a new section is presented on the right side of the paper, blank pages may be required,
7. The document is to be checked using the in-built Accessibility Checker in Microsoft WORD ( <https://support.microsoft.com/en-us/office/improve-accessibility-with-the-accessibility-checker-a16f6de0-2f39-4a2b-8bd8-5ad801426c7f> ),
8. Consider accessibility of the document:
   1. <https://support.microsoft.com/en-us/office/video-improve-accessibility-with-heading-styles-68f1eeff-6113-410f-8313-b5d382cc3be1>
   2. <https://support.microsoft.com/en-us/office/accessibility-video-training-71572a1d-5656-4e01-8fce-53e35c3caaf4>
9. The Table of Contents is generated with the “formal style” with Calibri 12-point font,
10. The IEEE citation style of referencing is to be used. Endnote may be used to aid the creation and use of references,
11. Note the use of page numbering and the numbering style,
12. Replace the chapter titles with appropriate wording according to the project,
13. Remove sections that are not required,
14. Add header 2, header 3, etc. styles for chapter sections and sub-sections, and suitably format,
15. Figure captions are to be placed below a figure (to the left of the page), and table captions are to be placed above the table (to the left of the page),
16. The project Gantt chart and final presentation slides are to be included as appendices at the end of the report.
17. If a weekly progress report was kept during the project, include an appendix that presents each weekly progress report (summary of actions in each week of the project).

Declaration

**This report is presented in part fulfilment of the requirements for the LM118 Bachelor of Engineering in Electronic and Computer Engineering Final Year Project.**

**It is entirely my own work and has not been submitted to any other University or Higher Education Institution or for any other academic award within the University of Limerick.**

**Where there has been made use of work of other people it has been fully acknowledged and referenced.**

|  |  |
| --- | --- |
| **Name** |  |
| **Signature** |  |
| **Date** |  |

Table of Contents

Replace this text with a table showing the section/sub-section numbers, titles, and page numbers (such as shown below).

[Abstract i](#_Toc96019634)

[Declaration iii](#_Toc96019635)

[Table of Contents v](#_Toc96019636)

[List of Figures vi](#_Toc96019637)

[List of Equations viii](#_Toc96019638)

[List of Tables x](#_Toc96019639)

[Chapter 1 Introduction 1](#_Toc96019640)

[Chapter 2 Frameworks and Methods of Interest 4](#_Toc96019641)

[Chapter 3 Architecture (Tech Stack) 7](#_Toc96019642)

[Chapter 4 Execution (If perfection perfected itself) 8](#_Toc96019643)

[Chapter 5 Problems and Solutions (Where there’s a will there’s a way) 9](#_Toc96019644)

[Chapter 6 Background Interests, Hypothesis’ and Analytics 10](#_Toc96019645)

[Chapter 7 Conclusions and future work 11](#_Toc96019646)

[References 12](#_Toc96019647)

[Appendices - 1 -](#_Toc96019648)

[Appendix A: Updated project Gantt chart - 2 -](#_Toc96019649)

[Appendix B: Final presentation slides - 3 -](#_Toc96019650)

[Appendix C: Project poster - 4 -](#_Toc96019651)

[Appendix D: Project progress reports - 5 -](#_Toc96019652)

List of Figures

Replace this text with a table showing figure numbers, captions and page numbers

List of Equations

Replace this text with a table showing equation numbers, captions and page numbers.

Notable Events

Introduction

This Final Year Project was created to be an experimental way of looking at the various software’s and what data they will collect in a world where online interaction and work environments are getting more and more digitalized.

For one, when I started my FYP, the various “verses”, like Meta’s metaverse and Nvidia’s Omniverse were not a thing, now the next frontier of the web. This introduction of a 3D universe in a digital space for work and social activities, filled with NFTs, smart-contracts and Artificial Intelligence plays into the introduction of Web3.0.

Web3.0 is the perfect example of what is expected of the world wide web and how it is changing. It is an all-encompassing prediction of how we will interact with digital properties in the future. We will explore the possibilities and some driving research behind Web3.0, investigating it’s Semantic and AI properties and it’s influences on this project.

I see software as a solution to many problems of two separate parties, academic staff, and students. In a perfect world people would learn for free, ai would correct assignments and be available as a q&a chat bot 24/7, lecturers and PHD students would spend more time lecturing and researching, being recommended current top topics in their areas of expertise which can be procedurally generated into learning-formatted material based on information they’ve deemed worthy topics or explored.

I wanted to explore how to create an API and graphical interface, in doing this we can see how easy it is to create a portal to which people can visit and learn from or use interactive tools. I created a full stack web application plus various supporting scripts which mine data from different sources, and transforms it into different formats for applications.

Through collecting this various data, I realised how much textual data I had, I decided not to let the opportunity go to waste in the chance to process this data to create a recommendation model in which students can describe the career they want, and get recommended courses. This recommendation model is only an application of the pre-processing of unsorted textual data which is the true bulk of this part of the project.

Pre-processing of textual data is the foundation of any application of Natural Language Processing application, once text data is categorised and a model is trained, you can create continuous pipelines of document categorisation, chatbots, topic generalisation and recommendation models.

Through text analysis, we can deduct topics and categorise textual data so it can be used to train any type of artificial intelligence model you like, with categorised textual data we can create a predictive text model, informational chatbot, and automate or complete textual documents in a similar sense to openai’s GPT-3.

This project wanted to see how the applications of these types of foundational software stacks of API GUI and AI can assist students in creating a one stop shop for the start of the semester, and in finding a course which suits their description of themselves. Along the way I will be exploring topics such as Clustering, Data Analytics, alternatives to Frameworks or software, as well as performing investigations into various aspects of Third Level Education.

Personal motivation for this project is the fact many people cannot afford Third Level Education. I wish to make learning resources which can be customized and consumed by any person of any level. By using aggregated content from the internet based on peoples interests, we can create summarised learning material automatically through topic parsing, using inputs supplied by people, students and lecturers.

I hope a platform like this one day could assist someone in learning a subject that is not a clear cut path on how to evolve it’s topics or foundation. It could also be used as a platform for people to test out subjects before committing to a 4 year course, curving dropouts and helping people make informed decisions on their career.

This led me to create Unical, the full stack application where students could come at the start of a semester. I wanted to make an everything tool that streamlined students entry to start of semesters. The first few weeks of every semester are always stressful and will set the tone for the rest of the semester, it’s important to make this as easy for students as possible.

The current state of Unical will be explained throughout this project, briefly, it currently supplies downloadable calendars with integrated alarms based on the iCalendar (.ics) protocol used by many devices. These downloadable calendars are filled with events from their university timetable.

Using the information supplied by the university, Unical can recommend other content like recommended books by module professors. I will talk about how I built Unical and what I would of liked to evolve it to, and how I would build a Webapp like this if I was to recreate it.

The second motivation for this project as mentioned above, is helping people make informed, guided decisions on their transition from any level into Third Level, either as a mature student, leaving certificate student, transfer or exchange students. I’ve seen a lot of people close to me being let down by expectations of a course, this can de-rail your enthusiasm for at least a year, until the next CAO application.

I figured the solution to this is to make deciding on a course more personal. I wanted to have a tool which would curate courses based on the persons interests, or based on previous courses they read about and liked. Nowadays the best way of finding courses is by sifting through cao.ie or qualifax.ie, searching courses by keywords of what you have heard about elsewhere.

I will explain how I processed the text data, some better algorithms and some findings using Natural Language Processing techniques, and the best method of categorising unsorted textual data.

Frameworks and Methods of Interest

In this chapter we will define this projects current scope and the foundations behind it. Then we will explore the scope of where I hope this project could go and what would be needed for that. We will explore frameworks I used and how I could use them in this project.

***Scope, what we need our frameworks to support:***

1. Multiple web scrapers to gather large amounts of data.
2. An API with GUI for a portal to serve and host data/tools for students.
3. Methods of text processing, cleaning and analytics for a recommendation model as a tool to add to our portal in 2.

Python was chosen as the foundational language for each part of our scope. I have been writing Python for four years now, and have worked professionally in Python for a total of 2 years. I chose it due to my experience in it, specifically in web scraping, which will be the most important part of this project.

Multiple Python Libraries were used, which will be credited to throughout this project. Some honourable mentions are;

* Pandas; A data science package, used to create and manipulate Data Frames.
* Scrapy; A web scraping package, used to request and parse static HTML documents.
* Selenium; A browser automation tool which can programmatically navigate and parse web pages, specifically DOM (Document Object Model) enabled sites, which are HTML documents manipulated typically by JavaScript, of which cannot be navigated by Scrapy.
* Gensim; A Machine Learning document parsing package, to represent large corpus of text data as just a few summarised topics. Pre-filled with algorithms for parsing textual data, which will be used to find similarities among documents.

***Scope 1: Multiple Web Scrapers to Gather Large Amounts of Data.\***

Web scrapers, also called miners and/or bots, are programs designed to extract information from the internet; The most common use for web scrapers is to collect data for marketing purposes. For example, a web scraper might be used to collect data about the prices of items on a particular website so that a company can create a price comparison tool. [source]

Bots/Scrapers are used to do repetitive actions, such as scrolling or navigating of sites to extract pieces of information. Google uses web scrapers to aggregate information and recommend information based on keyword and context similarity of a given input.

***Scrapy:***

The framework for Python, Scrapy was released in 2008 and was developed by a web-aggregation company. [https://en.wikipedia.org/wiki/Scrapy]

Scrapy offers tools which are composed of:

* Auto-throttle, which assists in reducing strain on servers you are sending requests to, which also helps in staying undetected to sites, and is just a form of respect to not congest servers.
* Rotating Proxies, we can rotate proxies to remain undetected and avoid IP banning by sending different requests and performing different routines from different IP addresses. Luckily we did not experience being banned from any sources, but it is nice to have this back-up in the instance we do get banned.
* User-Agents, we can set different user-agents to make it appear as though we are using any device or browser we want the target site to believe we are using.

Combining these tools, we can develop scrapers with varying levels of detectability, not congesting servers, making traffic look as though is multiple users from different parts of the globe, we can appear as normal traffic. We can even use custom user-agents and headers to identify ourselves, and even log-in as our profile by inputting a auth token which is granted by most sites upon successful log-in.

Scrapy also offers Scrapy Shell, which is a kernel based environment for testing the expected behaviour of the scrapers, this will assist in debugging during development. We can use the kernel to step by step navigate and inspect the website and figure out the best way to develop the scraper. We can call the shell from our terminal using;

> scrapy shell “https://ul.ie”

Scrapy works by allowing us to utilize “*selectors*”. Selectors use the HTML attributes href , id , class , or tag to select a specific element from a web page, and then returns the information we need from that element. The documentation of Scrapy gives us excerpts of how we are to implement Selectors:

[https://docs.scrapy.org/en/latest/topics/selectors.html]

>>> from scrapy.selector import Selector

>>> body = '<html><body><span>good</span></body></html>'

>>> Selector(text=body).xpath('//span/text()').get()

'good'

Where the body is the HTML document, and the Selector object is a class predefined by the Scrapy developers. The parameter text in line 3 “Selector(text=body)”, we see a parameter which will be passed to the Classes \_\_init\_\_() function, this is a compulsory function which will run on the initialisation of the class, performing set-ups and initialising class attributes.

From our Selector() object, we can call a function xpath() (Selector(text=body\_of\_html).xpath(target\_html) ) where we pass the string of the xpath to search within the HTML element.

The ability to specify *attributes* (denoted by ‘@’ in xpaths and ‘::’ in css) is useful in being able to extract specific attributes of HTML elements, such as;

* Extracting links: @href
* The class of the HTML element: @class
* The id of the HTML element: @id
* The txt of the element; @text

We can even use different selector functions to specify whereabouts on the HTML element to extract, xpath was mostly used in this project due to the simplicity, but in instances xpath does not work we can use css(). CSS is usually what the stylesheet.css files in HTML directories are composed of. CSS is what gives HTML documents their colour, shape and layout, and are usually inherited into HTML elements using the “class” and “id” attributes, and can even be written into the HTML using the “style” element.

Taking HTML document:

<HTML>

<body>

<a href=http://ul.ie >University of Limerick</a>

</body>

</HTML>

Demonstrating

With target being the text (“University of Limerick”):

xpath: /html/body/a/@text

css: html body a ::text

With target being the link (“http://ul.ie”):

xpath: /html/body/a/@href

css: html body a ::href

Using Scrapy:

With target being the text (“University of Limerick”):

xpath: Selector(text = html\_document).xpath(‘/html/body/a/@text’)

css: Selector(text = html\_document).css(‘html body a ::text’)

With target being the link (“http://ul.ie”):

xpath: Selector(text = html\_document).xpath(‘/html/body/a/@href’)

css: Selector(text= html\_document).css(‘html body a ::href’)

Both the xpath and css functions of Scrapy rely on parseL, another open-source library for parsing HTML. parseL uses regex and the native Python lxml parsing package to search the HTML for the specified element or attribute.

Often we need to figure out the xpath to the link we wish to navigate to, in this case it is the timetable URL. Xpaths are a tricky thing to compose by yourself when a webpage has a lot of HTML tags and elements. Luckily we can use the inspection tool in any browser to find the html element and extract it’s xpath, css and others. [https://www.scrapestorm.com/tutorial/how-to-find-xpath-in-chrome/] Before doing so, it is important to go into your browser settings and turn off Javascript, since scrapy doesn’t support JavaScript, you need to look at the webpage like how scrapy does.

You can turn off JavaScript by changing settings in your browser, or use the Scrapy Shell and it’s native function *open()* passing in the response, which will show you what scrapy sees in browser. If you wish to do this from a script you need to import the open\_in\_browser() function and pass in the response to this function.

We use the selectors so often Scrapy comes with a shorter version to which we can use. Instead of importing the Selector object, we can directly call xpath() and css() from the html *response* object.

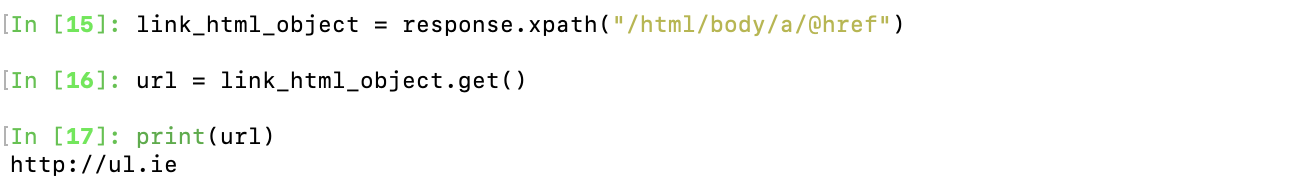
Using Selector Class;

Graphical user interface, text

Description automatically generated

* Here we import the Selector class, and initialize the class using the *body* of our *response*.
* We then call it’s “xpath” function, passing in the path to the link we wish to extract.
* *link\_html\_object* is now an instance of this selector class, there’s the data containing any href attributes of the xpath we provided. We can treat this variable as another Selector() class, just localised to only be able to access elements within the provided path. This function will always return a list, as we can point to multiple elements using our xpaths.
* Using the function get(), we extract the first element of the selector list. In the instance we collect a list of urls, and want to extract all of them too, we can use getall(), which returns a list of the data present.

Using Response Class;



* Here we simply call the response object (created by our script, from the initial request to the target site), and call the xpath function directly from it and display our data similarly to above.

We see using the response object is a lot more convenient for us. This was selectors and how we can use them to find different parts of a html document, and extract information like links and text.

**Response Class:**

The response class is the first thing returned from our scraper when it makes a *request.* The response class holds all the information about what the target site thinks of our request. It contains the html document to be returned among other data like;

* status codes (200 = Success, 404 = Page not Found, 401 = Unauthorized Access)
* headers (user-agent, cookies)
* body (body of the html document)
* meta (meta is a field we can store data in, if we needed to pass data between requests and responses, as the function that sends the request can input meta data that the function which interprets the response can use.)
* text (all the text of the html document, including the HTML and text attributes)

The response class also has a function called “follow()”, we can use follow by calling our response object and then calling follow; response.follow(url), url can be a relative url (/academic-registry/) or an absolute url (https://www.ul.ie/academic-registry/). *Follow* also comes with a parameter called “callback”, to which we can pass in a function, when the response from the *follow* is returned from our target url, the response is immediately passed to this function and is performed.

Here we see an excerpt from a UL scraper I built. This scraper was a prototype and was not used in production. We will talk more about why it wasn’t used in Chapter 5, which includes a call from UL cyber security.

def LoggedIn(self, response):

self.timetable['misc']['pwd'] = '' # clear password from memory.

element\_for\_student\_timetable = '//\*[@id="MainContent\_StudentTile"]/a/@href' # here we implement a shortened xpath type using regex.

student\_timetable\_link = response.xpath(element\_for\_student\_timetable).get()

return response.follow(student\_timetable\_link, callback=self.GetTimetable)

For context; by the time we get to this function we have just logged in and want to tell the scraper to go to the timetable URL. I could’ve hard coded the timetable URL having the scraper go from log in redirected straight to the timetable URL, but something like that can be pin pointed by IT administrators if they ever wanted to lock down on bots, so I try to make the actions it took look as human as possible, such as following the correct user flow.

In the parameters for the function we pass in an instance of the scrapers class itself (*self*), similar to how we can call “response.xpath()”, we can call “self.function()”, which is the same as calling the scraper itself “UL\_scraper.function()”. This is mandatory when creating a function within a class.

Firstly we wipe the password as I do store the data submitted by users, and of course we do not want to store passwords, so as soon as we are logged in we wipe this from our memory.

Secondly, we define our xpath to the link. Here you see a shortened version of xpath, it is a relative xpath, and utilises regex. The \* in regex represents anything, it’s a wild card, we tell it to find “*anything*”, \*, that has an attribute of id which is equal to “MainContent\_StudentTile”, go into this elements “a” tag, and look at the href attribute.

We then pass this xpath into our *xpath()* selector function, and extract the link using *get().* We pass this link into the response’s *follow* function, along with the function to call upon *callback*, which is the GetTimetable function where we parse the timetable page of timetable.ul.ie.

This is a brief overview of how Scrapy will play a big part in this project, from utilities to implementations. There is so much more which Scrapy can do from requests all the way down to storing the data, but a top-level view is sufficient enough for it’s current use in this project.

Scrapy is very useful with HTML documents not heavily reliant on JavaScript. But we often see the case in which when we turn off Javascript for web pages, and the page completely breaks, or our desired information is not displayed. Often we are met with a web page saying “Please turn on JavaScript and reload”.

For DOM (Document Object Model) webpages, we cannot use Scrapy, and require an alternative. We wish to still use scrapy, as it can be run in a Linux server environment and requires no arbitrary software, so it runs quite happily by itself and requires little resources since there is no Graphical Interface. Perfect to be hosted somewhere with little ram to run on demand.

**Selenium:**

Selenium is a web automation tool created in 2008 by an employee Jason Huggins at a software company called ThoughtWorks. The tool was created and evolved as a software automation test tool with a focus on web browser automation for navigation and testing of web-apps and websites. [https://en.wikipedia.org/wiki/Selenium\_(software)]

Selenium, unlike Scrapy, requires companion software such as the browser itself you wish to automate, and the suitable driver for the browser. From experience, this limits our hardware capabilities as if we wish to use a single board computer like a Raspberry Pi, it may not perform as well when having multiple browsers open performing operations, compared to running something like scrapy.

Selenium also limits you to choosing one browser, as the html of a webpage can change browser to browser. So if you are working with Google Chrome for Selenium, any xpaths exracted in Firefox will not work in your Selenium Script.

So why do we use it?; Selenium, using the power of the browser, can render JavaScript, and therefore widens our grasp at what websites we can scrape, as most up-to-date sites will not work in JavaScript.

Timeline

Description automatically generated with medium confidence Graphical user interface, text, website

Description automatically generated

Left: timetable.ul.ie with JavaScript, Right: timetable.ul.ie without JavaScript.

***Scope 2: An API with GUI for a portal to serve and host data/tools for students.***

In this scope the main package which I will be using is FastAPI. FastAPI is a python Framework in which we can create APIs and is composed of;

* WSGI (Web Server Gateway Interface, specification of a common interface between web servers and web applications)
* Werkzeug is a WSGI toolkit that implements requests, response objects, and utility functions.

[https://pythonbasics.org/what-is-flask-python/]

FastAPI also supports rendering of dynamic webpages, meaning we can use python variables within the HTML. This means Flask can serve as a full-stack solution, handling frontend and backend. I did not rely my entire stack on FastAPI, I often find separating out projects makes it easier to work on the separated components, in the event we wish to switch the frontend for a Framework such as Vue or Bootstrap, instead of the “*vanilla*” JavaScript, CSS and HTML I used.

Write about api hosting, creating and how we call it from javascript and how we change html using javascript

***Scope 3:*** A recommendation model as a tool to add to our portal in 2. (Which would later become the text processing, cleaning and analytics.)

Basically I want to be able to extract sub-data from the data itself, I want to identify structure within a body of text, and I want to be able to find similar courses. I already am thinking to go institution to institution, as we would expect to find similar structure among the fields, and similar language, but we cannot use the same method of extracting things like LC requirements or course contents in NUIG as UL, so we need a single function that can identify these things when shown a sample of documents.

I also want to apply the function to the approach of going similar course to similar course, if I can identify similar titles and content within popular fields, weighing emphasis on the rarity of words like “Engineering”, “Sciences”, “Fourier Series”.

So our scope is:

 - A single function to identify structure and important strings and terms, especially key pairs ("English: H2", "O3/H4 in Biology", “Graphic Design: Year 1: Students will….”) when given a sample of similar terminology (course title approach) or similarly structured (course provider approach) documents.

I believe by being able to reduce the text data into hyper organised classified data, we can use this pipeline on any academic course document and have it ready to be applied in training machine learning models such as recommendation models, information system models, and even in text analysis, chatbots, and the list goes on.

The idea is to build a pre-cursor that reformats existing un-categorised data, into a common format with predictions of what the course document could be about, either stating a course name or a course discipline.

One method we will use we spoke about already in the scraper section as Regular Expression (“Regex”).

A recommendation system was the original goal, but I found the textual data was too “dirty”, causing the similarity analysis’ done with Gensim and clustering done with skikit

Recognition of Pattern Identified Entities: Features such as telephone numbers, e-mail addresses, quantities (with units) can be discerned via regular expression or other pattern matches.

Document clustering: identification of sets of similar text documents

Pre-processing usually involves tasks such as tokenization, filtering and stemming.

Architecture (Tech Stack)

Execution (If perfection perfected itself)

Chapter to start on odd number page.

Problems and Solutions (Where there’s a will there’s a way)

Java python tabular package

Information consistency issue

Cybersecurity with UL

Background Interests, Hypothesis’ and Analytics

Chapter to start on odd number page.

Conclusions and future work

Replace this text with the conclusions and future work. Chapter to start on odd number page.

References

Replace this text with the references. References to start on odd number page. Use the IEEE citation style for references.

Appendices

Replace this text with a list of the appendices. Appendices to start on odd number page.

Appendix A: Updated project Gantt chart

Replace this text with the project Gantt chart. This will be an update of the Gantt chart presented in the interim report, and reflect changes made during the project implementation. Appendix to start on odd number page.

Appendix B: Final presentation slides

Replace this text with the project final presentation slides (2 slides per page format). Appendix to start on odd number page.

Appendix C: Project poster

Replace this text with a copy of the project poster.

Appendix D: Project progress reports

Replace this text with the project progress reports, weekly or monthly, if created. If progress reporting was not part of the project, this appendix can be removed.