Structure

GCE A Level Computer Science Project

Name: Jeremiah Boby
Candidate Number: 8302
School: Loreto Sixth Form College

Analysis

Problem Definition and Stakeholders

The stakeholders of my application will be fellow students of Loreto Sixth Form College. Being A Level students, they need to organize revision sessions in order to maintain the grades required for their chosen university.

Many Loreto college students have issues with organising a revision timetable. For example, creating study sessions that do not clash with classes, or setting up reminders for each of these study sessions. The needs of a College student also include regularly checking up on their performance (grades and attendance) to ensure that they are making the best of their study periods. My solution will help students with these tasks, by providing a simple, intuitive interface for them to create, update and set reminders for their study sessions.

I have chosen a fellow Computer Science student, **Troy Sherlock**, to be my stakeholder for this project. Troy is an ideal candidate for my software, as he is currently in his second year of A Levels with Maths, Physics and Computer Science. This means that efficient organisation of free study periods is vital, so he will benefit greatly from my solution.

Currently, a student must do the following in order to carry this ou t:

- Check their current timetable on the college website
- Check their grades and predicted grades on the college website
- Use this information to create a manual revision timetable
- Regularly update this timetable manually based on the above criteria

The issue with this system is that it is entirely manual, which means that students may neglect to carry out certain steps. Also, the current system does not allow a student to add custom lessons to their online timetable, or be notified of their upcoming lessons. My solution is to have a web-based dashboard that will integrate all these features, in a single web application.

My application will solve the problem by integrating revision management, homework reminders and performance analysis in one simple dashboard. Students will be able to access this dashboard via a Django web application.

In order to solve this problem with my chosen solution, I will need to learn the following:

- Usage of the Django web framework
- Usage of the PostgreSQL Database Management System
- Integration of this DBMS with Python
- Usage of JavaScript graphing libraries such as D3.js

Justification of Computational Methods

This problem lends itself to utilization of computational methods for several reasons:

- **Student performance data will be abstracted**
  Rather than being displayed as raw text data, it will be displayed in a series of tables and charts. This will make it much easier for the end-user to comprehend. In order to display the data in this format, a computational device will be needed. This is because the client will need to be able to execute client-side code (JavaScript) and render HTML and CSS to display

- **A Server-side will need to be implemented**
  As my solution will be a web application, I will need to write code to run on a server, which will serve HTML templates to the client computers.

Application of Computational Methods

There are several ways that computational methods can be applied to this project:

Method	Explanation of relevant features
Abstract	My web application will be as user-friendly as possible. this will require me to hide certain details from the user. For example, the raw data that is scraped by my webscraper. Instead, the user will only see data that is relevant to their current page: If they are on the summary page, they will only see simple summarized data and graphs for their performance.
Heuristic	The summary page of my dashboard will include rounded, approximate values to represent the correlation between various properties of the student's study time, free time, grades, and punctuality.
Procedural	My application stack will be comprised of several Python modules, each separated into classes, methods and functions. I will be able to remove modules, test modules and edit modules in a way that does not affect any irrelevant components of the application.
Concurrent	The web application must be able to handle multiple HTTP requests simultaneously. I will be applying concurrent thinking to plan and design the processing and response for each request.

The statements above have shown that my problem can be solved with a computational method. This makes it suited for a computer to solve with a program.

Research

Stakeholder Interview Questions

- Have you ever used a homework management system before?
- If so, what are your favourite features of this system?
- Are there any features you want to see from a system like this?
- Are there any features you want removed?

Interview with Troy Sherlock

Have you ever used a homework management system before?

- Yes, I have used Show my Homework in high school to manage my timetabled lessons and homework. However, it did not offer a system to manage revision sessions.

What are your favourite features of this system?

- My favourite feature is the ability to colour-code homework slots depending on whether they are completed or not. It allows me to easily check

Are there any features you want to see from a system like this?

- I would like to see a reminder system, such as browser push notifications for due homework. I would also like the ability to save and load custom revision sessions.

Are there any features you want removed?

- I do not like Show My Homework's "my calendar" system, and would like it to display more accurate time data for each homework slot.

From my stakeholder interview, I have concluded that **if the problem is successfully solved, the stakeholders will be able to access a web dashboard where they will be able to view their school timetable, add and remove events from this timetable, and view statistics of their study progress.**

Checking for a Successful solution

In order to ensure that my solution was successful in meeting the requirements described above, I will need to consider the following areas:

- **Meets user needs**
  I will need to ensure that the solution works well for my stakeholder's purposes. To measure this success criteria, I will use a grading system where my stakeholders will rate my application on a scale of 1 to 10 in usefulness.

- **Reliable**
  I will need to ensure that the solution is robust and continues to work optimally while it is in use. In order to measure this criteria, I will ensure that my stakeholders have access to the solution over a long period of time, and they will rate the reliability of the service on a scale of 1 to 10.

Similar solutions

I have found a similar solution to my own, **Show My Homework**, by Satchel.
Show my Homework is a service designed for schools, that aims to unify student timetable and homework tasks.

Features worth including	Possible limitations
• Simple interface • Colour – coded homework tasks • Homework dates sync with lessons • Homework reminders before lesson • Mobile client application and API	• Time limit I am given a relatively short time frame to complete a project of this scale. Therefore, sacrifices may be made to functionality in order to complete the project on time. • Cost Given that my project is one that requires a server to run my code, there may be costs incurred by the hosting providers that will be used. Therefore, I have decided to use the free tier of Heroku, a well known free hosting provider, to host my web application. This may result in performance losses, as Heroku Free Tier machines are put on “standby” when data is not requested after a set amount of time, requiring a couple of minutes to return to an active state.

Functionality Requirements

The solution will be required to:

- **Display any Loreto student’s timetable in an intuitive manner**
  This means that I will need to communicate with MyLoreto to retrieve this information, then parse it in such a way that my program will be able to extract the necessary information from it, and finally render this information in an abstracted format. As intuitiveness of the solution is subjective, I will be using the opinion of my stakeholders to judge this.

- **Allow students to create their own timetable entries**
  This means that I will need to store the student's timetable in a persistent way, suggesting that I will need to use a database. This database will store student timetables, as well as authentication for MyLoreto to verify the lessons. I will need to insert and remove data from the database upon the user's request, so usage of SQL may be required.

Execution Requirements

The machines running my solution will be required to:

- **Serve content at any time**
  The server-side code must be run on hardware that can maximise uptime, i.e. stay online and running a python web app indefinitely. This kind of hardware is provided by a number of services, and I will be using a Heroku Free Tier virtualised private server. As mentioned above, running this application on a free service is not ideal. However, Heroku provides build integration with GitHub, allowing me to easily

push updates to my live web app. As for software requirements, the VPS will need to support the python runtime and the PostgreSQL database, which Heroku does by default.

- **Execute JavaScript code and render HTML**
  The client-side code must be executed on hardware that supports a display system and web browser, as my service Is a web application. This minor task can be completed by most ARM/x64 devices, such as mobile phones, laptops, desktops, or refrigerators, therefore I do not have any hardware requirements. The only software requirement is to be able to support a web browser.

Design

Key Objects

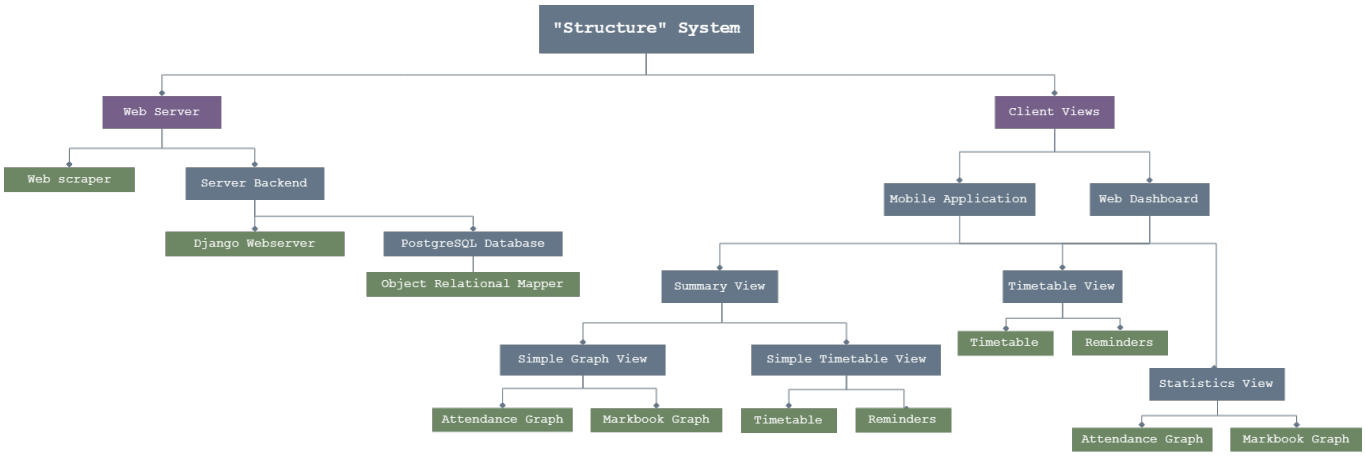
There are several key objects that I will need to use in my project:

Student	I will use this class to represent each student. It will have attributes that represent the student's ID, name, email, Base64-encoded avatar, reference number, tutor and timetable.
User	This class represents the least amount of data required to authenticate with MyLoreto. It will have only two attributes: username and password.
LandingPageParser	This class will be used to parse a given Loreto URL. It will take an instance of a User object in its contructor, and contains many methods that will be used to parse HTML.

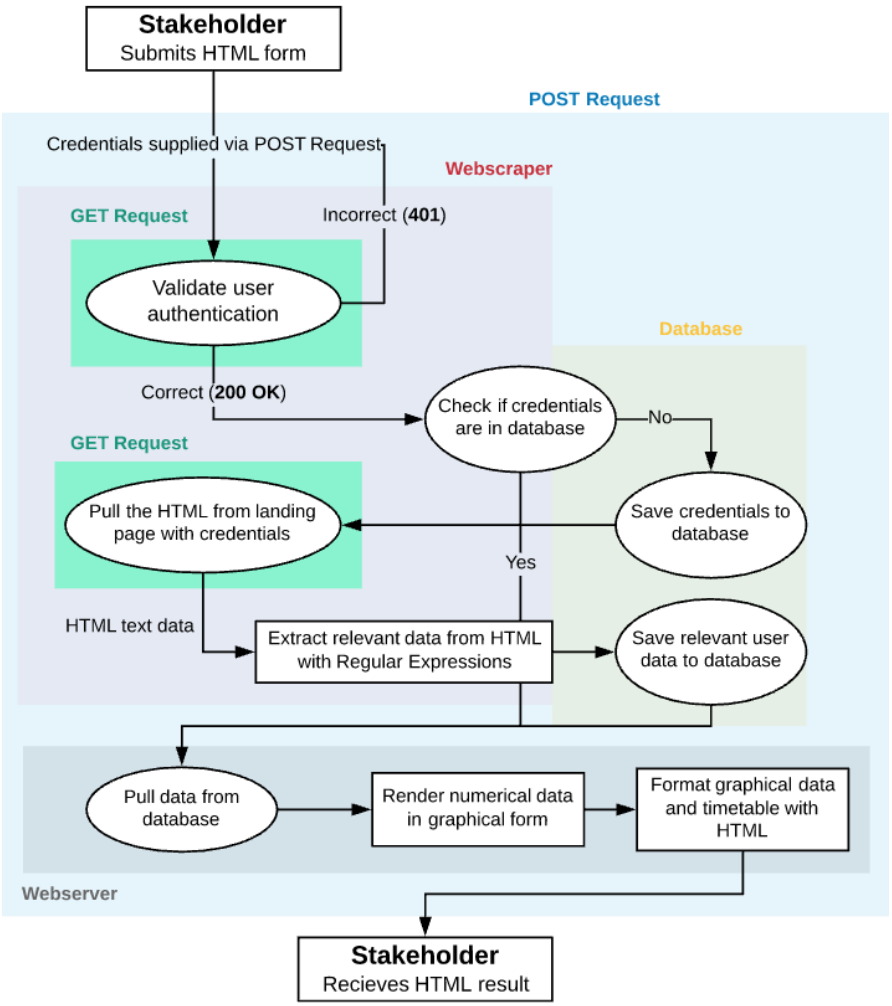
Design

Problem Decomposition

I have broken down my problem into a series of smaller problems, which are defined below:



I have also created a Data Flow Diagram which represents the smaller parts of my projects, and how they relate to the overall solution:



I have also written some heavily abstracted pseudocode for each aspect of the solution:

```
import requests

from models import Student


def generate_html_response(student):
    return '''
    <html>
        <p>Hello {0}.</p>
        <p>Your grades are: {1}</p>
    </html>
    '''.format(student.name, student.grades)


def parse_with_regex(response):
    pattern = 'Grades: (.*)'
    return regex.extract(pattern, response)


@call_on_post_form
def post(form):
    authentication = form.username, form.password
    response = requests.get(loreto_url, authentication)
    if response.status_code != 200:
        return 'Try again.' + form
    else:
        student = Student(authentication)
        if not student.in_database():
            data = parse_with_regex(response)
            student.grades = data.grades
            student.timetable = data.timetable
            student.attendance = data.attendance
            student.save_to_database()
        result = generate_html_response(student)
        return result
```

Development

Before starting development, I have decided on the naming convention and code style I will use throughought the project. This will be based on PEP8, a document stating the standard conventions for Python style guide.

In summary, I will be conforming to the following rules:

All functions and methods I define must contain a docstring explaining its use:

```
def add_two_numbers(a, b):
    """Adds two numbers, a and b."""
    return a + b
```

All variables, methods and functions will use snake_case**, rather than** mixedCase.
Any classes which I define will use PascalCase.

```
some_variable = 'foo'

class SomeClass:
    def __init__(self):
        self.bar = 'baz'

    def what_is_bar(self):
        """Returns the value of bar."""
        return self.bar

some_instance = SomeClass()
```

As is usual with any code (but particularly Python), I will be naming my variables and methods with thought, using names that enhance readability.

I started by creating a simple script that will allow me to request and parse data from MyLoreto. I have decided to use Kenneth Reitz's **requests** library to handle HTTP requests and responses, and the **re** builtin module to parse the content with regular expressions.

I will be using Regular Expression patterns to catch certain substrings of a HTTP response. For example, from this snippet of HTML:

```
<dl class="dl-horizontal" style="font-size: 0.9em">
        <dt>Reference: </dt>
        <dd>S20170000420</dd>
</dl>
```

I am able to extract the Reference number with the following regex pattern:
`Reference: </dt>\s+<dd>([A-Z0-9]+)`

```
>>> import re
>>> html = """
...     <dl class="dl-horizontal" style="font-size: 0.9em">
...         <dt>Reference: </dt>
...         <dd>S20170000420</dd>
...     </dl>"""

>>> match = re.search(r'Reference: </dt>\s+<dd>([A-Z0-9]+)', html)
>>> match.group(1)
'S20170000420'
```

I have written up an example of my original webscraper prototype below:

```
import re

import requests


endpoint = "https://my.loreto.ac.uk/"

patterns = {
    'name': b'fullName: "([A-Za-z ]+)"',
    'username': b'username: "([A-Za-z0-9]+)"',
    'avatar': b'base64,(.*?)">',
    'reference_number': br'Reference: </dt>\s+<dd>([A-Z0-9]+)',
    'tutor': b'Tutor: </dt> <dd> (.*?) </dd>'
}


def get_student_data(*credentials: str) -> dict:
    """Get data about a student, given their username and password."""
    student_data = {}
    landing_page = requests.get(endpoint, auth=credentials).content
    for key, pattern in patterns.items():
        student_data[key] = re.search(pattern, landing_page).group(1)
    return student_data
```

An example of this module in use can be seen below:

```
>>> get_student_data('jerbob42', 'password')
{
    'name': b'Jeremiah Boby',
    'username': b'JerBob42',
    'avatar': b'/9j/4AAQSkZJRgABAQEASABIAAD...',
    'reference_number': 'S20170000420',
    'tutor': 'Mr Bloggs'
}
```

Potential Improvement

I could improve this function's performance by utilising Python's builtin **threading** module, starting a new **Thread** for each regex search that I start. This means that each regex search will not block the execution of the next, enhancing my concurrent throughput.

I then decided to create a function for validating user credentials. I decided to add this validation to prevent users encountering an unexpected error or crash immediately after providing incorrect credentials.

In the case of incorrect credentials, this function will return **False**, and an alert will be provided to the user.

```
def valid_user(*credentials: str) -> bool:
    """Take a student's credentials and validate them."""
    print('Validating user...')
    return requests.get(
        endpoint, auth=credentials
    ).status_code == 200
```

This function checks that the HTTP response code for a request with the provided credentials is successful. If the credentials are incorrect, the server will respond with a **401 response code (Unauthorized)**, instead of a **200 (OK)**.

With some simple validation and webscraping done, I then decided to start work on the main webserver. To begin with, I used the `django-admin` command-line tool to generate the skeleton project required for the webserver, with `django-admin startproject structure` .

This created a directory with the following structure:

```
structure/
    manage.py
    structure/
        __init__.py
        settings.py
        urls.py
        wsgi.py
```

In order to comply with **separation of concerns**, I have decided to implement my student dashboard as a separate module in the project, to be named `dashboard`. To do this, I used the following command: `python manage.py startapp dashboard`

I then removed all redundant files which were not required for the project. This resulted in the following directory structure:

```
structure/
    manage.py
    structure/
```

```
        __init__.py
        settings.py
        urls.py
        wsgi.py
    dashboard/
        __init__.py
        models.py
        views.py
```

In order to be able to modify and update my database, I did some research into the Django Documentation on its builtin ORM. Though I initially decided to create my own ORM for this project, I have made the decision to use Django's builtin ORM to save time on the project.

If I had more time to enhance this project, I would have written a custom ORM for this purpose.

I described the following database models below in the **`structure/dashboard/models.py`** file like so:

```python
"""Define database models for use in the PostgreSQL server."""

from django.contrib.postgres import fields
from django.db import models


class Student(models.Model):
    """Student model: contains information used by the dashboard."""

    student_id = models.AutoField(primary_key=True)
    name = models.CharField(max_length=20)
    email = models.CharField(max_length=29)
    avatar = models.CharField(max_length=20000)
    reference_number = models.CharField(max_length=12)
    tutor = models.CharField(max_length=50)
    timetable = fields.JSONField()
    attendance = fields.JSONField()
    markbook = fields.JSONField()

    class Meta:
        # Give the database table a specific name.
        db_table = "student"


class User(models.Model):
    """User model: contains information specific to user authorization."""

    user_id = models.AutoField(primary_key=True)
    username = models.CharField(max_length=8, unique=True)
    password = models.CharField(max_length=20)

    class Meta:
        # Give the database table a specific name.
        db_table = "user"
```

As you can see, these

models do not match the ones that I created during the Design phase. This is because I needed to create a simplified version of the product for the initial version, so that I was able to test it as soon as possible.

Next, I added some simple views to run when a user requests pages. These were defined in the **`structure/dashboard/views.py`** file:

```python
from django.shortcuts import redirect, render
from django.views.generic import TemplateView

from .models import User
from .scraper import valid_user


# Dictionary containing data about a user's session
current_session: dict = {}


class HomePageView(TemplateView):
    """HomePage view: default view of the landing page."""

    def get(self, request, *args, **kwargs):
        """Runs on a HTTP GET request."""
        if current_session.get('authenticated'):
            # If the current user is authenticated, render the dashboard
            return render(request, "dashboard.html", context=current_session)
        else:
            # If the current user is not authenticated, redirect to the signin page
            return redirect("/sign-in")


class SignInView(TemplateView):
    """SignInView: view for the sign in page."""

    def get(self, request, *args, **kwargs):
        """Runs on a HTTP GET request."""
        return render(request, "sign-in.html", context=None)

    def post(self, request):
        """Runs on a HTTP POST request."""
        # Get the username and password from the POST request
        username, password = (
            request.POST.get('username'), request.POST.get('password')
        )
        # Create a User object with the username and password
        user = User(username=username, password=password)
        # Validate the user object
        if valid_user(user.username, user.password):
            # If the current user is valid, add the user to the session
            current_session['user'] = user
            # Set the "authenticated" flag to True
            current_session['authenticated'] = True
            # Redirect to the home page
            return redirect('/')
        else:
            # If the current user is not valid, set the "authenticated" flag to False:
            current_session['authenticated'] = False
        # If there has been no previous returns, render the signin page
        return render(request, "sign-in.html", context=None)
```

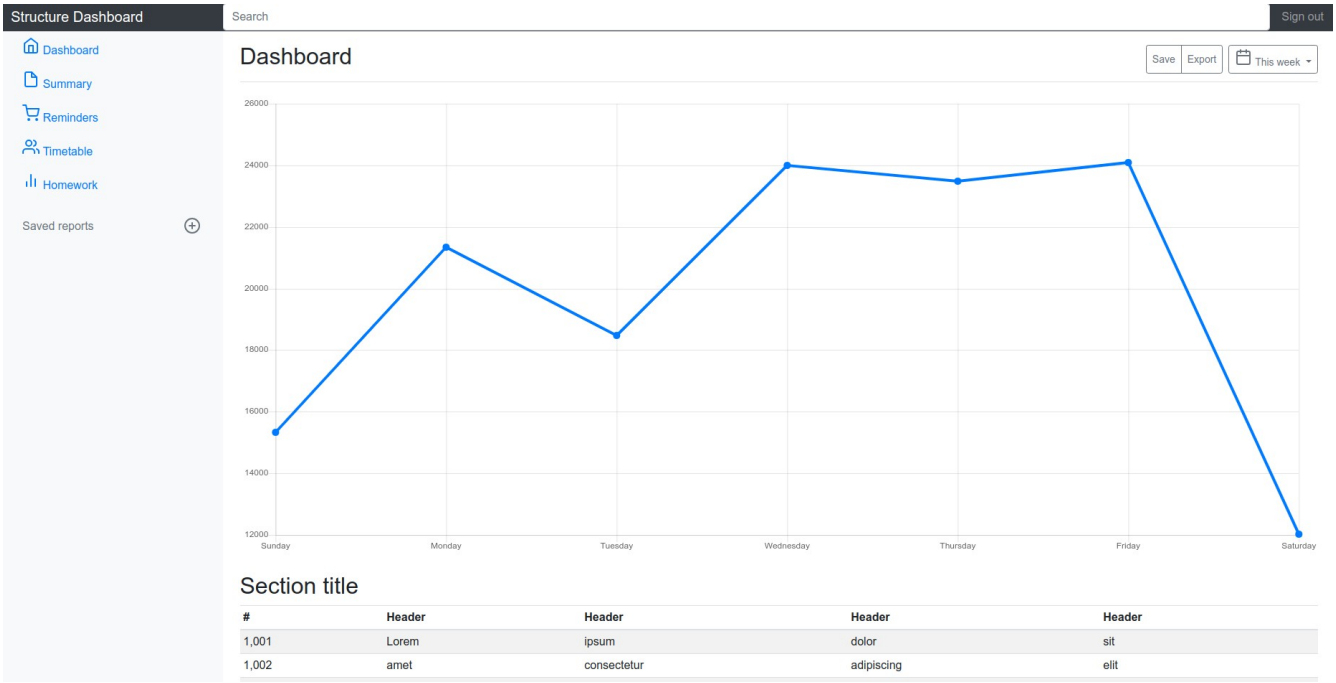
As shown above, when a user's browser requests the home page, the following code is executed:

```
def get(self, request, *args, **kwargs):
    """Runs on a HTTP GET request."""
    if current_session.get('authenticated'):
        # If the current user is authenticated, render the dashboard
        return render(request, "dashboard.html", context=current_session)
    else:
        # If the current user is not authenticated, redirect to the signin page
        return redirect("/sign-in")
```

In **`dashboard/templates`**, I also added **`dashboard.html`** and **`sign-in.html`**.
`dashboard.html` is a simple example page for the CSS framework that I am using, Bootstrap:

```
<html lang="en">
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">
    <meta name="description" content="">
    <meta name="author" content="">
    <link rel="icon" href="https://getbootstrap.com/favicon.ico">

    <title>Structure - Dashboard</title>

    <!-- Bootstrap core CSS -->
    <link href="../static/css/bootstrap.min.css" rel="stylesheet">

    <!-- Custom styles for this template -->
    <link href="../static/css/dashboard.css" rel="stylesheet">
  </head>

  <body>
    <nav class="navbar navbar-dark sticky-top bg-dark flex-md-nowrap p-0">
      <a class="navbar-brand col-sm-3 col-md-2 mr-0" href="dashboard.html#">Structure Dashboard</a>
      <input class="form-control form-control-dark w-100" type="text" placeholder="Search" aria-label="Search">
      <ul class="navbar-nav px-3">
        <li class="nav-item text-nowrap">
          <a class="nav-link" href="dashboard.html#">Sign out</a>
        </li>
      </ul>
    </nav>

    <div class="container-fluid">
      <div class="row">
        <nav class="col-md-2 d-none d-md-block bg-light sidebar">
          <div class="sidebar-sticky">
            <ul class="nav flex-column">
              <li class="nav-item">
                <a class="nav-link active" href="dashboard.html#">
                  <span data-feather="home"></span>
                  Dashboard <span class="sr-only">(current)</span>
                </a>
              </li>
              <li class="nav-item">
                <a class="nav-link" href="dashboard.html#">
                  <span data-feather="file"></span>
                  Summary
                </a>
              </li>
              <li class="nav-item">
                <a class="nav-link" href="dashboard.html#">
                  <span data-feather="shopping-cart"></span>
                  Reminders
                </a>
              </li>
              <li class="nav-item">
                <a class="nav-link" href="dashboard.html#">
                  <span data-feather="users"></span>
                  Timetable
                </a>
              </li>
              <li class="nav-item">
                <a class="nav-link" href="dashboard.html#">
                  <span data-feather="bar-chart-2"></span>
                  Homework
                </a>
              </li>
            </ul>
  </body>
</html>
```

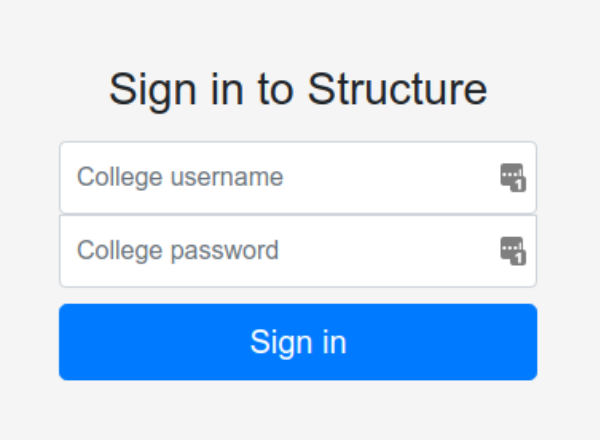
This resulted in the following web page:



`sign-in.html` contains just a login form:

```
<form action="", method="post", class="form-signin">
  <h1 class="h3 mb-3 font-weight-normal">Sign in to Structure</h1>

  <input id="inputUsername" name="username" placeholder="College username" required type="text">
  <input id="inputPassword" name="password" placeholder="College password" required type="password">
  <input class="btn" id="submit" name="submit" type="submit" value="Sign in">

</form>
```

This resulted in the following web page:



When the user enters their username and password, these parameters are sent as part of the POST request to the same endpoint, this is the default action done by HTML forms.

As shown in the backend code for this view, a POST request triggers the following code:

```
def post(self, request):
    """Runs on a HTTP POST request."""
    # Get the username and password from the POST request
    username, password = (
        request.POST.get('username'), request.POST.get('password')
    )
    # Create a User object with the username and password
    user = User(username=username, password=password)
    # Validate the user object
    if valid_user(user.username, user.password):
        # If the current user is valid, add the user to the session
        current_session['user'] = user
        # Set the "authenticated" flag to True
        current_session['authenticated'] = True
        # Redirect to the home page
        return redirect('/')
    else:
        # If the current user is not valid, set the "authenticated" flag to False:
        current_session['authenticated'] = False
    # If there has been no previous returns, render the signin page
    return render(request, "sign-in.html", context=None)
```

Potential Improvement

Currently, I am relying on the mutability of a global variable. This is unreliable, as it will only work for one user at a time. It is not scalable, nor lend itself to concurrent thinking. Instead, I could use Django's builtin `request.session` object, which is unique for each request, as it is an attribute of one parameter of the request method.

The code shown above does not store user details. Instead, it only stores a global `authenticated` flag.

In order to display student information, I will need to add more information to the scraper. In order to achieve this, I decided to create a custom class, which I will use solely to parse the MyLoreto landing page.

There are several reasons why I have decided to use a class for parsing the landing page:

- It allows me to create a combination of variables, methods and statements that I can reproduce whenever I need to.
- It allows me to import this class into any module I need, lending to **modular thinking**.
- It helps with separation of concerns, as all variables and methods will only have the scope of the instance it applies to.
- It creates more readable, maintainable code, as I write less and document more.

```
class LandingPageParser:
    """Class for parsing the Loreto landing page."""

    def _set_regex_group(self, page: bytes, name: str, pattern: bytes):
        """Set the Student data to the first group of the given match."""
        match = re.search(pattern, page)
        if match is not None:
            # If the Regex match was successful
            self.student.__setattr__(name, match.group(1).decode())

    def _get_short_timetable(self):
        """Get the timetable for the current day."""
        # List of lessons in the current day.
        timetable = []
        # Regular Expression pattern for a lesson.
        pattern = re.compile(
            b'TimetableEntry .*?>(?P<time>[0-9 -:]+)'
            b'.*?(?P<room>[()A-Z0-9]+)'
            b'.*?(?P<teacher>[()A-Za-z0-9 -]+) ',
            re.DOTALL
        )
        # Iterate each attribute of every lesson in the timetable
        for time, room, teacher in pattern.findall(self.page):
            # Remove brackets from the lesson
            room = room.strip(b'()')
            subject, teacher = teacher.split(b' - ')
            # Append this lesson dictionary to the list
            timetable.append(
                {
                    'time': time.decode(),
                    'room': room.decode(),
                    'subject': subject.decode(),
                    'teacher': teacher.decode()
                }
            )
        # Set the student's timetable to this list
        self.student.short_timetable = timetable
```

```python
    def _get_timetable(self):
        """Get the timetable for the current week."""
        # Create a dictionary of day -> lesson list
        timetable = {
            'Monday': [],
            'Tuesday': [],
            'Wednesday': [],
            'Thursday': [],
            'Friday': []
        }
        # Extract the HTML snippets for each day
        day_meta = '<th>{0}(.*?)(<th>|</script>)'
        # Pattern that will be applied to the previous day snippet
        lesson_meta = (
            'Times">(?P<time>[0-9: -]*)'
            '<.+?Code">(?P<subject>[A-Za-z() -]*)'
            '<.+?Staff"> *?(?P<teacher>[A-Za-z ]*) *?'
            '<.+?right">(?P<room>[A-Z0-9]*)'
        )
        # Get the Monday of the current week
        now = datetime.now()
        start = now - timedelta(days=now.weekday())
        start = start.strftime('%Y-%m-%d')
        # Create a HTTP request to MyLoreto with the student's data
        response: bytes = requests.post(
            f'{endpoint}attendance/timetable/studentWeek',
            data={'week': start, 'student_user_id': self.user_id},
            auth=(self.user.username, self.user.password),
            headers={'X-Requested-With': 'XMLHttpRequest'}
        ).content
        # Iterate each day and lesson list in the timetable
        for day, lesson in timetable.items():
            # Extract the content
            content = re.search(
                day_meta.format(day).encode(), response, re.DOTALL
            ).group(1).decode()
            for match in re.finditer(lesson_meta, content, re.DOTALL):
                # Append each lesson to the timetable
                lesson.append(match.groupdict())
        # Set the parser instance's "timetable" attribute to this timetable
        self.student.timetable = timetable

    def __init__(self, user: User):
        # Get the student's landing page with their credentials
        self.page = requests.get(
            endpoint, auth=(user.username, user.password)
        ).content
        # Set the student and user attributes persistently
        self.user = user
        self.student: Student = Student()
        # Pull the user's UserId with a regular expression
        user_id = re.search(
            br'UserId = "(\d+)"', self.page
        )
        if user_id is not None:
            # if the UserId was found:
            self.user_id: int = int(user_id.group(1))
        # Define regex patterns for use by core components of the student object.
        patterns = {
            'name': b'fullName: "([A-Za-z ]+)"',
            'username': b'username: "([A-Za-z0-9]+)"',
            'avatar': b'base64,(.*?)">',
            'reference_number': br'Reference: </dt>\s+<dd>([A-Z0-9]+)',
            'tutor': b'Tutor: </dt> <dd> (.*?) </dd>'
        }

    def parse(self) -> Student:
        """Parse the webpage content and return the resulting Student."""
        self._get_short_timetable()
        self._get_timetable()
        for key, pattern in patterns.items():
            self._set_regex_group(self.page, key, pattern)
        self.student.email = '{0}@student.loreto.ac.uk'.format(
            self.student.username
        )
        return self.student
```

I am using regular expressions to extract parts of the HTML page. I use regular expressions for the following reasons:

- Regular expressions enable me to reduce my code base drastically. If I were to use simple string manipulation to extract the data I need, I would have significantly more unmaintainable code.
- Regular expressions are much easier to read than many different string manipulations.
- Regular expression patterns are reusable once compiled.

However, I soon realised that each of these regular expression searches are blocking the thread of execution. Each extraction prevents the next one from being carried out. In order to apply concurrent thinking to this problem, I decided to use the **`threading`** module, part of Python's standard library, to run each regex search in a separate thread:

```
    def __init__(self, user: User):
        self.page = requests.get(
            endpoint, auth=(user.username, user.password)
        ).content
        self.user = user
    # Set of threads used in the parser.
    # I am using a set to ensure that threads are not duplicated.
        self.threads: set = set()
        self.student: Student = Student()
        user_id = re.search(
            br'UserId = "(\d+)"', self.page
        )
        if user_id is not None:
            self.user_id: int = int(user_id.group(1))
        patterns = {
            'name': b'fullName: "([A-Za-z ]+)"',
            'username': b'username: "([A-Za-z0-9]+)"',
            'avatar': b'base64,(.*?)">',
            'reference_number': br'Reference: </dt>\s+<dd>([A-Z0-9]+)',
            'tutor': b'Tutor: </dt> <dd> (.*?) </dd>'
        }
    # Create a thread for getting the student's daily timetable, add it to threads
        self.threads.add(Thread(target=self._get_short_timetable))
    # Create a thread for getting the student's full timetable, add it to threads
        self.threads.add(Thread(target=self._get_timetable))
    # The threads created so far have not been started yet
        for key, pattern in patterns.items():
        # For every regex pattern in the patterns I need to extract:
            self.threads.add(
            # Create a thread to extract that pattern and add it to threads
                Thread(
                    target=self._set_regex_group,
                    args=(self.page, key, pattern)
                )
            )

    def parse(self) -> Student:
        """Parse the webpage content and return the resulting Student."""
    # The __init__ method does not start the threads.
    # Instead, this parse method will start each thread.
        for thread in self.threads:
        # Start each thread concurrently
            thread.start()
        for thread in self.threads:
        # Close each thread once they have completed
            thread.join()
        self.student.email = '{0}@student.loreto.ac.uk'.format(
            self.student.username
        )
        return self.student
```

There are several reasons why using a threaded solution is an improvement:

- Decreases the time needed for initialising the student object
- Increases maximum code throughput, as multiple methods of my code will be running at one time
- This solution therefore lends itself to concurrent thinking well.