



FIT9136 Algorithm and programming foundation in Python

Assignment 3

Lecturer in Charge: Shirin Ghaffarian Maghool

May 2022

Table of Contents

1. Key Information	3
1.1. Do and Do NOT	4
1.2. Documentation	4
1.3. Submission	4
2. Getting help	6
2.1. English language skills	6
2.2. Study skills	6
2.3. Things are tough right now	6
2.4. Things in the unit don't make sense	6
2.5. I don't know what I need	6
3. Key tasks	7
3.1. Overview	7
3.2. Website business logic description	8
3.2 Task 1 - design classes	16
3.2.1 User class	16
3.2.2 Admin class	18
3.2.3 Instructor class	19
3.2.4 Student class	21
3.2.5 Course class	21
3.3. Task 2 - Index, user register, login and logout	25
3.4. Task 3 - Students page	26
3.5. Task 4 - Courses page	27
3.6. Task 5 - Instructors page	28
Important Notes:	29

1. Key Information

Purpose	<p>This assessment is related to the following learning objectives (LO):</p> <ul style="list-style-type: none">● L04: Investigate useful Python packages for scientific computing and data analysis● L05: Experiment with basic data manipulation, analysis, and visualisation technique to formulate business insight● L07: Build a basic web application based on Python web frameworks
Your task	<p>It is an Individual assignment, where you will write a code for a simple emulation of a data analysis website.</p>
Value	<p>35% of your total marks for the unit</p>
Due Date	<p>[Friday 10th Jun 2022, week 14] 4:30 pm</p>
Submission	<ul style="list-style-type: none">● Via Moodle Assignment Submission.● Turnitin will be used for similarity checking of all submissions.
Assessment Criteria	<p>See Moodle Assessment page</p>
Late Penalties	<ul style="list-style-type: none">● 10% deduction per calendar day or part thereof for up to one week● Submissions more than 7 calendar days after the due date will receive a mark of zero (0) and no assessment feedback will be provided.
Support Resources	<p>See Moodle Assessment page</p>
Feedback	<p>Feedback will be provided on student work via: general cohort performance specific student feedback ten working days post submission</p>

1.1. Do and Do NOT

Do	Do NOT
<ul style="list-style-type: none">• Maintain academic integrity¹• Get support early from this unit and other services in the university• Apply for special consideration for extensions²	<ul style="list-style-type: none">• Leave your assignment in draft mode• Submit late (10% daily penalty applies)³• Submission is not accepted after 7 days of the due date, unless you have special consideration.

1.2. Documentation

Commenting your code is essential as part of the assessment criteria (refer to Marking Rubrics).

You should also include comments at the beginning of your program file, which specify your name, your Student ID, the start date and the last modified date of the program, as well as with a high-level description of the program. In-line comments within the program are also part of the required documentation.

1.3. Submission

You have to submit your assignment via the assignment submission link (i.e., "Assignment 2 Submission") on the Moodle site by the deadline specified in Section 1, i.e. **10th Jun 2022 (Friday) by 4:30 pm**:

- There will be NO hard copy submission required for this assignment.
- You are required to submit your assignment as a .zip file named with your Student ID. For example, if your Student ID is 12345678, you would submit a zipped file named 12345678.zip
- Do not include any unnecessary file in this folder
- Note that marks will be deducted if this requirement is not strictly complied with.
- **No submission accepted via email.**

1

<https://www.monash.edu/rlo/research-writing-assignments/referencing-and-academic-integrity/academic-integrity>

² <https://www.monash.edu/exams/changes/special-consideration> (All the Special Consideration should be applied no later than two University working days after the due date of the affected assessment).

³ eg: original mark was 70/100, submitting 2 days late results in 50/100 (10 marks off). This includes weekends

1.4. Deliverables

Your submission should contain the following documents:

- All files in the “model” folder include course.py, user.py, user_admin.py, user_instructor.py and user_student.py.
- All files in the “controller” folder include course_controller.py, index_controller.py, instructor_controller.py and user_controller.py.
- Several files in the “templates” folder include 00login.html, 00register.html, 10student_list.html and 11student_info.html.
- All files need to be compressed into a zip file. The final submission file name is {your_student_id}.zip.
- **Marks will be deducted for any of these requirements that are not strictly complied with.**

2. Getting help

2.1. English language skills

if you don't feel confident with your English.

- Talk to English Connect: <https://www.monash.edu/english-connect>

2.2. Study skills

If you feel like you just don't have enough time to do everything you need to, maybe you just need a new approach

- Talk to a learning skills advisor:
<https://www.monash.edu/library/skills/contacts>

2.3. Things are tough right now

Everyone needs to talk to someone at some point in their life, no judgement here.

- Talk to a counsellor:
<https://www.monash.edu/health/counselling/appointments>
(friendly, approachable, confidential, free)

2.4. Things in the unit don't make sense

Even if you're not quite sure what to ask about, if you're not sure you won't be alone, it's always better to ask.

- Ask in Ed: <https://edstem.org/au/courses/7429/discussion/>
- Attend a consultation:
<https://lms.monash.edu/course/view.php?id=135703§ion=21>

2.5. I don't know what I need

Everyone at Monash University is here to help you. If things are tough now they won't magically get better by themselves. Even if you don't exactly know, come and talk with us and we'll figure it out. We can either help you ourselves or at least point you in the right direction.

3. Key tasks

This assignment is a data analysis website. You are required to analyse the given data in folder `data/source_course_files`. This project is based on Python Flask Framework and you need to complete the files inside the **controller** folder named **course_controller.py**, **index_controller.py**, **instructor_controller.py** and **user_controller.py**, **model** folder named **course.py**, **user.py**, **user_admin.py**, **user_instructor.py** and **user_student.py** and several html pages (views) in the **templates** folder called **00login.html**, **00register.html**, **10student_list.html** and **11student_info.html**.

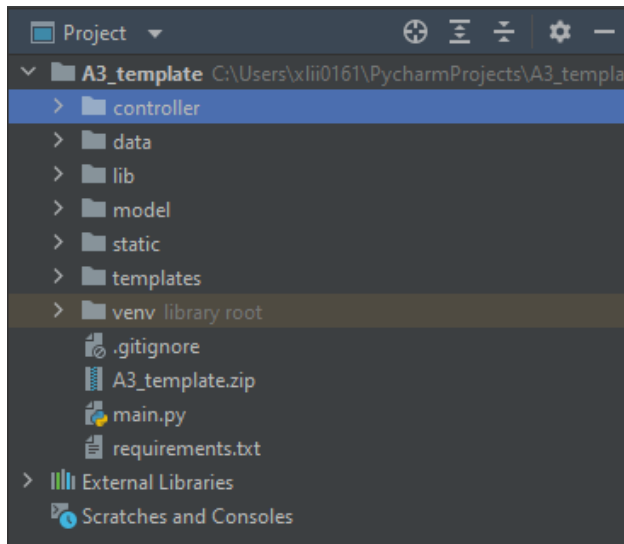
This is an individual assignment and must be your own work.

You can use any libraries except the time-related libraries or functions in this assignment.

In the **provided project-> lib folder**, there are a few methods and variables in the **helper.py** file that can be imported and used in your tasks. For example, if you need to access the `course.txt` data file, you can add `"from lib.helper import course_data_path"` in your file. All the path variables in the `helper.py` **should not** be changed. Any unexpected changes may cause the website to not run, which will lead to mark penalties.

3.1. Overview

1. **Data explanation** In the data folder, the **source_course_file** folder contains all the data files we need to use in this assignment. The `course.txt` and `user.txt` files are used to save all the course and user data. The example of course and user data can be found in `_demo_course.txt` and `_demo_user.txt` files. (course.txt and user.txt files are provided but empty)
2. **Folder structure**

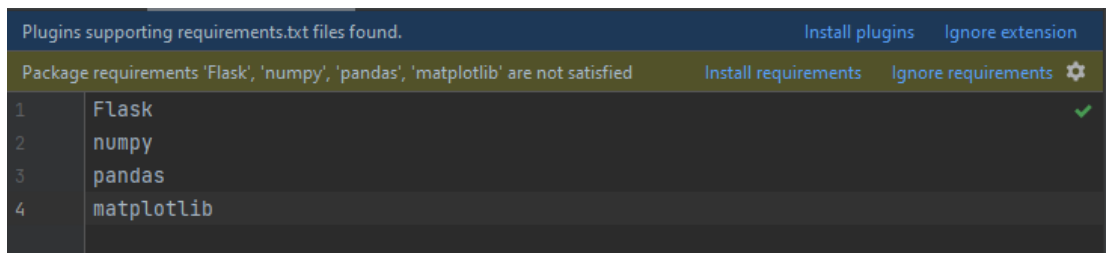


The image above shows a folder structure of this assignment. Running the *main.py* file can start this web application. The template file has some incomplete code which will generate errors. It is better to finish some tasks before starting to run the web application.

3. Start the assignment

Step1. Create a new project in Pycharm and copy all the files into this project.

Step2. Open requirements.txt file and you can see the image below.



Step3. Click install requirements.

After the instalment of all libraries, you can start to work on this assignment.

(Feel free to use the `pip install -r requirements.txt` if you can understand this command)

3.2. Website business logic description

1. Index page - Index page is the homepage of your web application. To launch the application you must run the *main.py* file, and use the link: <http://localhost:5000/> in the browser.

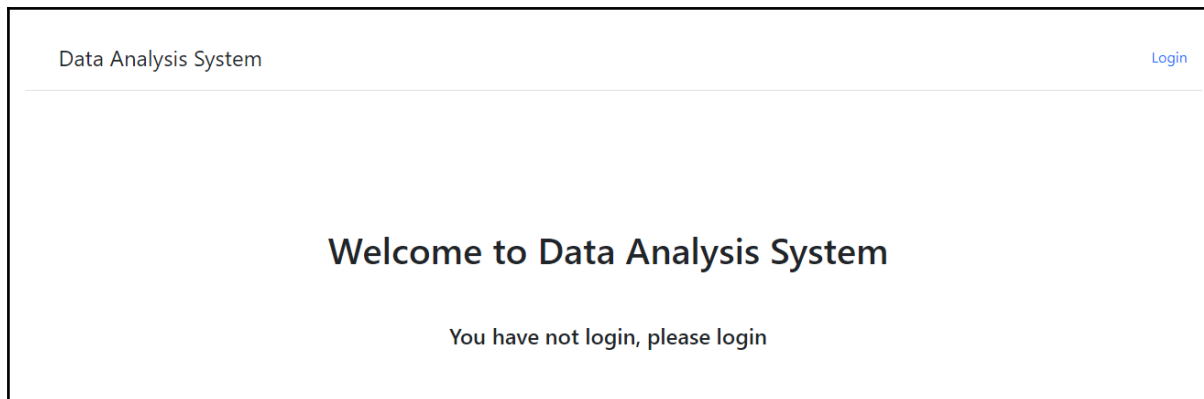


Figure 3.1 Index page no login

2. Login page - Login page allows the user to login into the web application which requires them to input username and password. There are three different kinds of users, i.e., Admin, Instructor, and student. Each kind of user will see different contents from the home (index) pages after login.

The image shows a login page with the heading 'Please sign in'. Below the heading are two input fields: 'Username' and 'Password'. Below these fields is a blue button labeled 'Sign in'. Under the button, there is a link that says 'No account? Please register [here](#)'. At the bottom of the page, the copyright notice '© 2017–2022' is displayed.

Figure 3.2 Login page

For example, when a student logs in, the index (home) page looks like the figure 3.3. Students can only check their own information and logout.

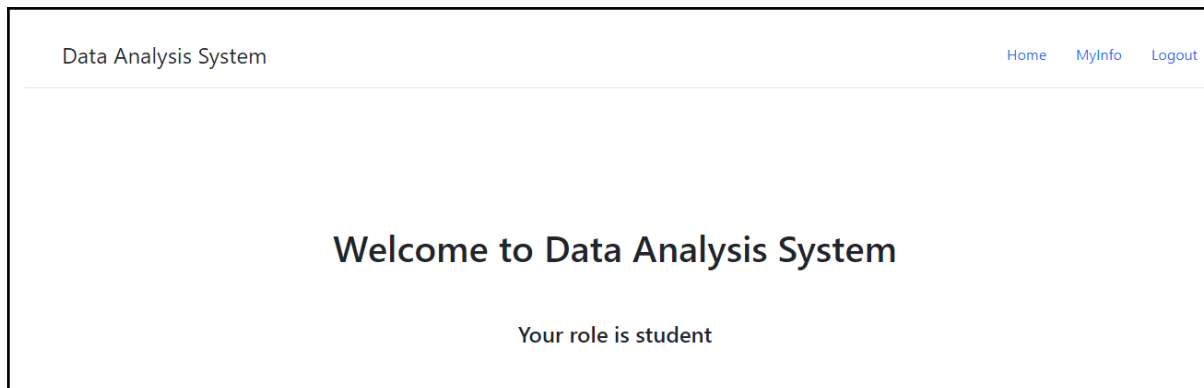


Figure 3.3 Student login page

For example, when the instructor logs in, the index (home) page looks like the figure 3.4. Instructors can only see the courses they teach and logout.

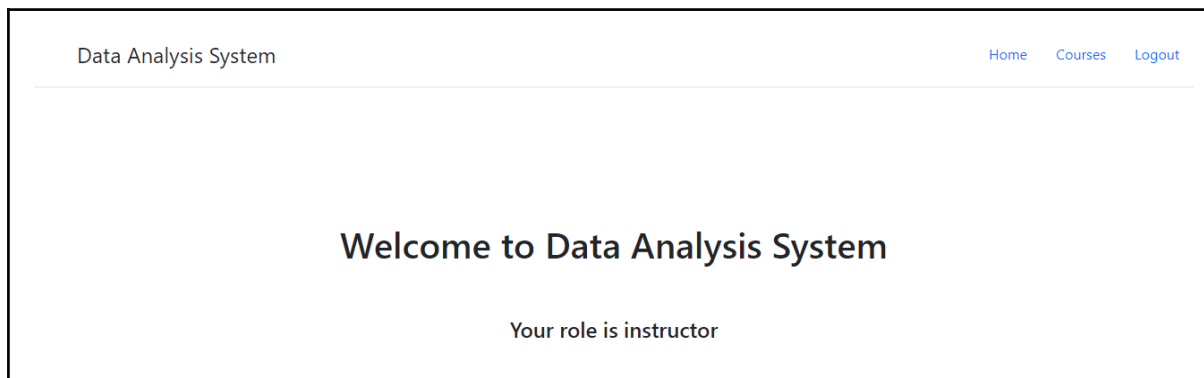


Figure 3.4 Instructor login page

For example, when admin logs in, the index (home) page looks like the figure 3.5. There is a reset database button which can remove all the content in the *course.txt* and *user.txt* files.

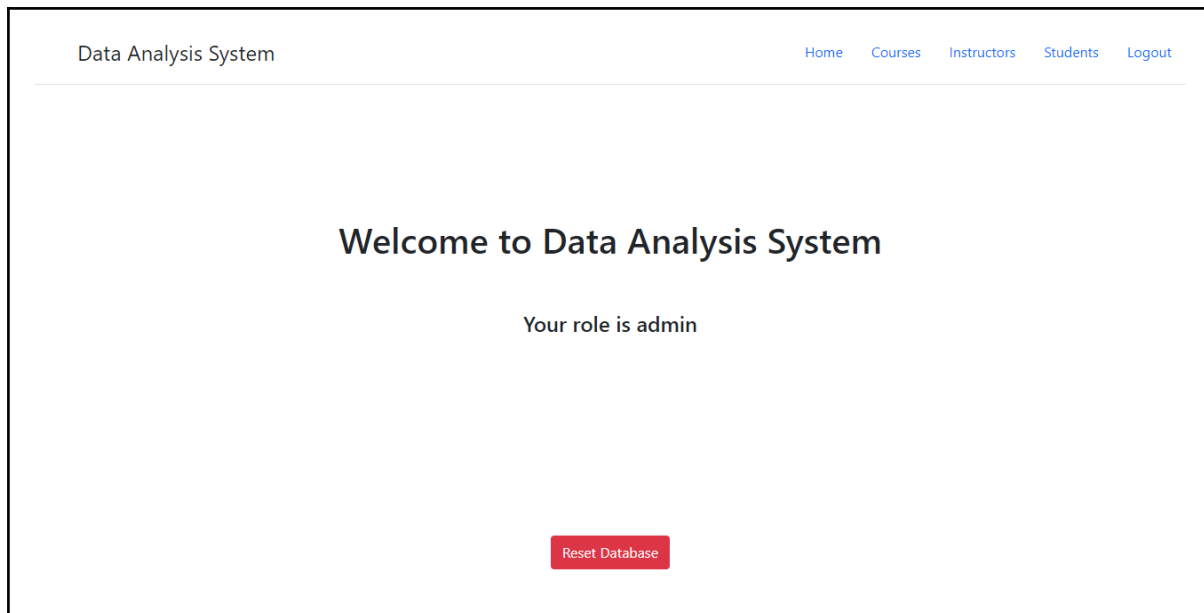
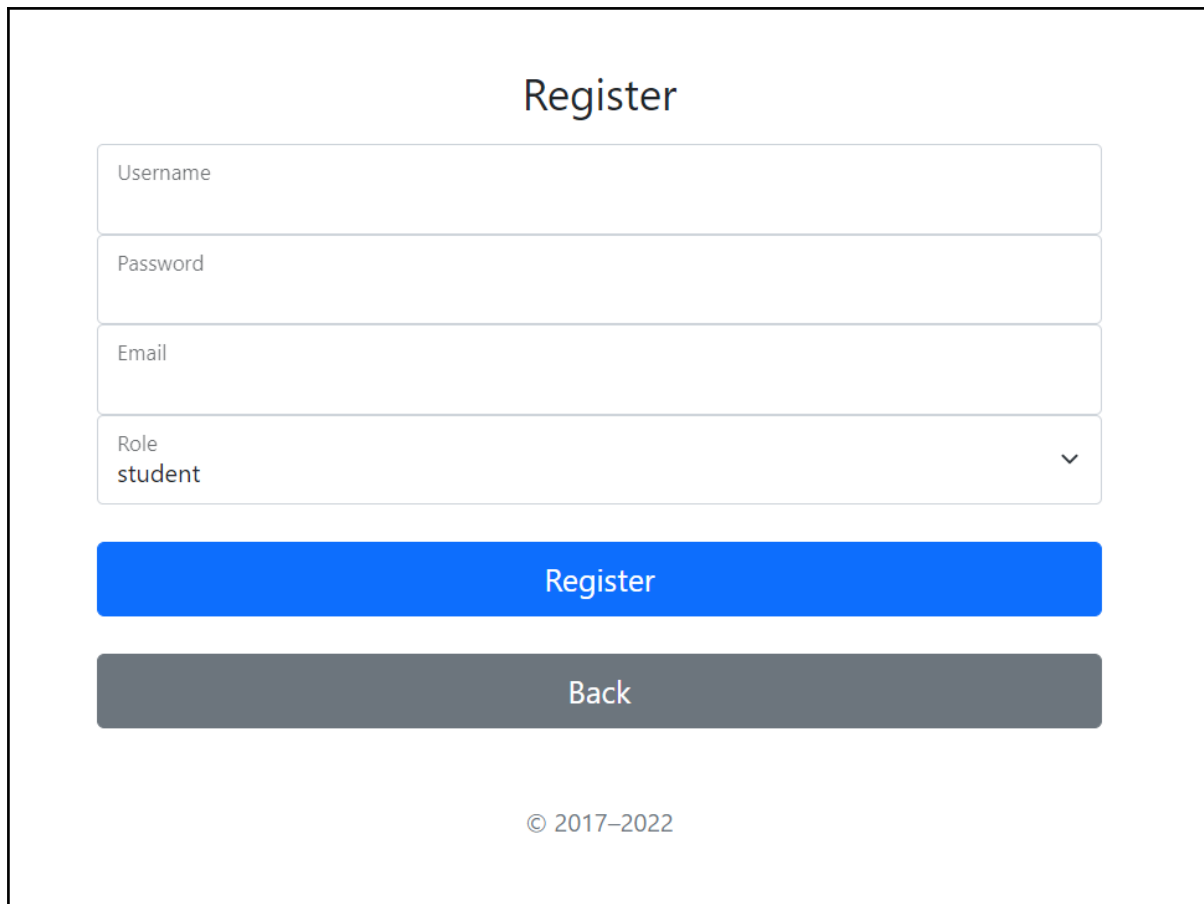


Figure 3.5 Admin login page

4. Register page - Register page allows users to register them into the web application, in the case they do not have an account. It is required to enter username, password, email address and role when registering a user. And a register timestamp(unix epoch time) will be generated automatically. All these five values will be sent to the application's backend to store the values in the user.txt file.



The image shows a web form titled "Register". It contains four input fields: "Username", "Password", "Email", and "Role". The "Role" field is a dropdown menu with "student" selected. Below the fields are two buttons: a blue "Register" button and a grey "Back" button. At the bottom, there is a copyright notice "© 2017-2022".

Register

Username

Password

Email

Role
student

Register

Back

© 2017-2022

Figure 3.6 Register page

5. Courses page - After a user logs in as admin, they can see the similar page as in figure3.7. In this page, the total number of courses and 20 course objects will be displayed. All the course objects can be returned based on the page number. By default, the page number is 1. Each page has a maximum of 20 courses. At the bottom of the webpage, a page number list is shown. By default, the page number list is always be [1,2,3,4,5,6,7,8,9].

Two buttons are placed below the total number of courses, which are the **Process Course Data** button and the **Course Analysis Figure** button. The **Process Course Data** button will retrieve all the data from given course data files(this process may take some time). The **Course Analysis Figure** button will generate 6 figures and the explanations of each figure. All the figures are saved into `lib.helper.figure_save_path` and all the explanations(i.e., the trend of the figure) about each figure will be displayed in a new webpage.

For each item of course, there will be a Details button and a Delete button. Details button will take you to the course details page. The Delete button will remove selected data from the course.txt file and user.txt file.

Data Analysis System				Home	Courses	Instructors	Students	Logout
<h2>Course Analysis</h2> <p>Number of Courses: 2220</p> <div><button>Process Course Data</button><button>Course Analysis Figure</button></div>								
#	Category Title	Subcategory Title	Course Title	Average Rating				
1	Development	Web Development	The Web Developer Bootcamp 2021	4.70	<button>Details</button>	<button>Delete</button>		
2	Development	Web Development	Angular - The Complete Guide (2021 Edition)	4.62	<button>Details</button>	<button>Delete</button>		
3	Development	Web Development	The Complete 2021 Web Development Bootcamp	4.68	<button>Details</button>	<button>Delete</button>		
4	Development	Web Development	The Complete JavaScript Course 2021: From Zero to Expert!	4.71	<button>Details</button>	<button>Delete</button>		
5	Development	Web Development	Modern React with Redux	4.68	<button>Details</button>	<button>Delete</button>		
6	Development	Web Development	The Complete Web Developer Course 2.0	4.55	<button>Details</button>	<button>Delete</button>		
7	Development	Web Development	Build Responsive Real-World Websites with HTML and CSS	4.66	<button>Details</button>	<button>Delete</button>		
8	Development	Web Development	The Complete Web Developer in 2021: Zero to Mastery	4.69	<button>Details</button>	<button>Delete</button>		

Figure 3.7 Course list page

Data Analysis System		Home	Courses	Instructors	Students	Logout
<h2>Course Title: The Web Developer Bootcamp 2021</h2> <p>Course ID: 156173119</p>						
Category:	Development					
Subcategory:	Web Development					
Subcategory Description:	Learn web development skills to build fully functioning websites.					
Subcategory Url:	null					
Course Url:	/course/the_web_developer_bootcamp_2021/					
Number of Subscribers:	713744					
Average Rating:	4.70					
Number of Reviews:	215075					
Overall Comment:	Top Courses					

Figure 3.8 Course details page

If you login as an instructor, the courses page will show the courses this instructor teaches.

Data Analysis System		Home	Courses	Logout
<h2>Instructor Teach Course List</h2>				
Number of Courses: 8				
#	Course ID	Course Title	Avg Rating	
1	39385047	Business Analysis Certification Program – The Concepts	3.85	
2	112007285	Land a Job in Innovation: The Complete Interview Guide	4.12	
3	582468331	Unlock HR Virtual Interview Techniques	0.00	
4	204107886	Blockchain Programming: Smart Contracts with Ardor	4.10	
5	970798161	Macroeconomía	4.83	
6	798836200	CFA® Level 1 (2021/2022) - Complete Fixed Income	4.64	
7	662704084	Burp Suite Bug Bounty Web Hacking from Scratch	3.95	
8	378659188	Beyond Arduino, Part 2: Analog Input Output	4.05	

Figure 3.9 Courses page when login as instructor

If you login as a student, there is no courses page available.

6. Instructors page - In this page, the total number of instructors will be displayed. Because each instructor can teach more than one course, the total number of instructors is less than the total number of courses. All the instructor objects can be returned based on the page number. By default, the page number is 1. Each page has a maximum of 20 instructors. At the bottom of the webpage, a page number list is shown. For each instructor, we can see all the courses this instructor teaches by clicking the Teach Courses button.

There are two buttons below the total number, that are **Process Instructor Data** button and **Instructor Analysis Figure** button. The Process Instructor will extract all the instructor information from the given data files and store instructors info into the user.txt file. This method may take a while to finish. The Instructor Analysis Figure button has similar functionality as the course analysis button.

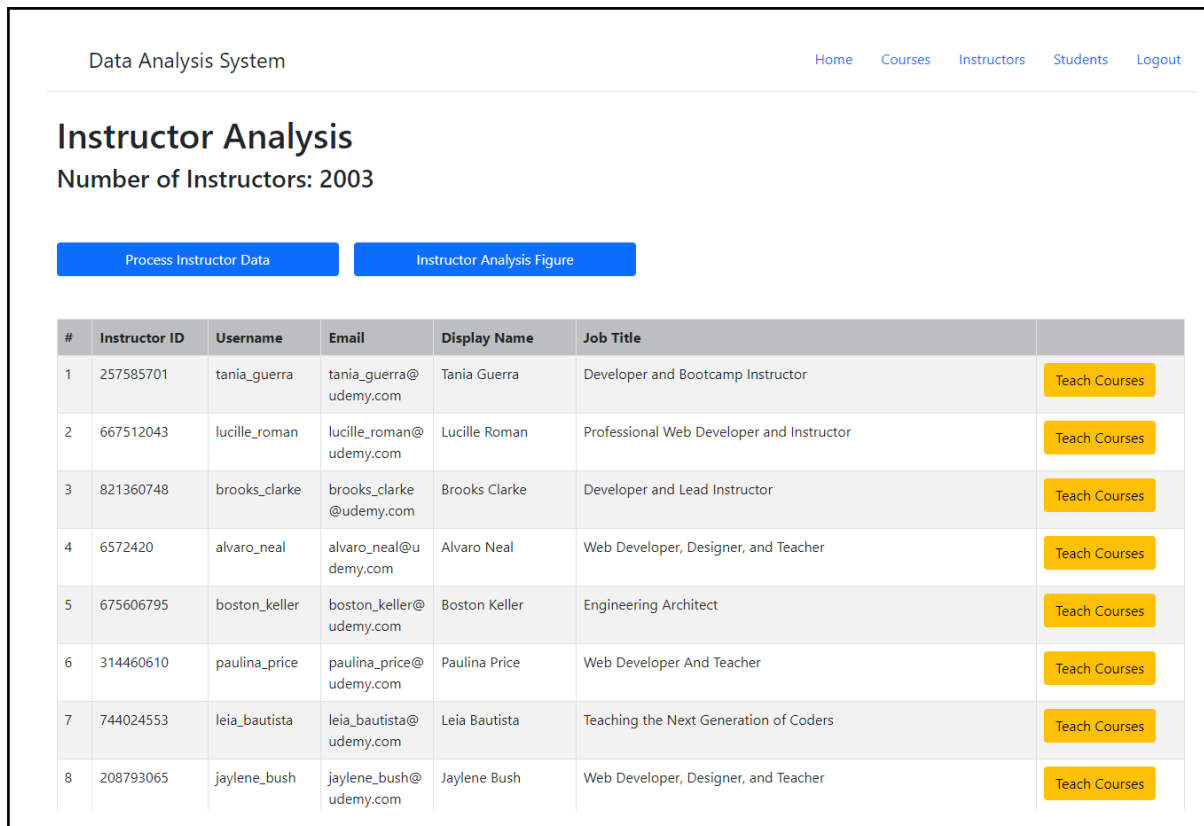


Figure 3.10 Instructors page

7. Students page. In this page, the total number of students and a list of students will be displayed. Students' info are not extracted from the files but registered manually in the register page.

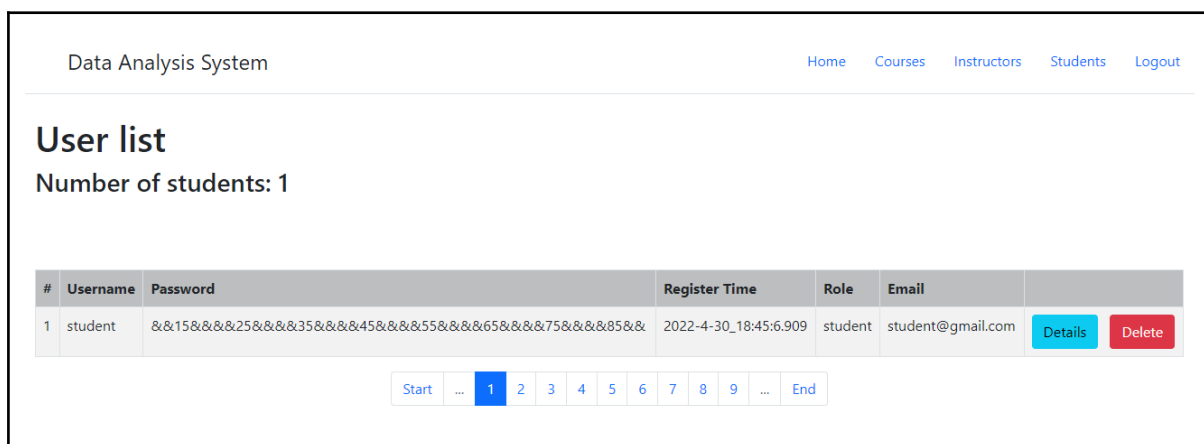


Figure 3.11 students page

Admin can click the details button to see students' details and click the delete button to remove this student (the student info will be deleted from the user.txt file).

If you login as a student, you can only see the MyInfo page.



Figure 3.12 Myinfo page when login as a student

3.2 Task 1 - design classes

You are required to implement the User class, Admin class, Instructor class, Student class, and Course class in the user.py, user_admin.py, user_instructor.py, user_student.py and course.py files respectively. **The Admin, Instructor and Student class should inherit from the User class.** The methods of each class are described below. The path in the helper.py file is valid for the Flask application. If you hope to perform a local test, you need to change the path. But, remember to change it back when you submit your files as your tutor will mark your work based on the submission files. The helper.py file is not submitted.

3.2.1 User class

1. constructor.

Five positional arguments: uid(int, default value is -1), username(str, default value is ""), password(str, default value is ""), register_time(str, default value is "yyyy-MM-dd_HH:mm:ss.SSS"), role(str, default value is ""). The role can only be value "admin", "instructor" and "student".

2. __str__() -> str.

Return string format example:

"uid;;;username;;;password;;;register_time;;;role"

3. authenticate_user() -> (bool, str).

Two positional arguments - username and password. This method is used to check whether username and password can be matched with users saved in user.txt data file. If matched, this method will retrieve the user information from user.txt file and **return a tuple (True, user_info_string), otherwise return (False, "").**

4. **check_username_exist()->bool.**

One positional argument - username. This method is to check whether the given username exists in the user.txt data file. If it exists, return True, otherwise return False.

5. **generate_unique_user_id()->str.**

This method is used to generate and return a 6 digit unique user id which is not in the user.txt file.

6. **encrypt_password()->str.**

One positional argument - password. For a given password, you are required to encrypt the string. You can reuse the encryption method in A1/A2 here.

7. **register_user()->bool.**

Five positional arguments - username, password, email, register_time, role. **The validation can happen in this function or before calling this function.**

- o If the username exists in the user.txt file, return False.
- o A unique user id is required when registering a new user.
- o If the user registers successfully, return True.
- o Register_time will be a unix epoch timestamp (milli seconds) which needs to be converted using **date_conversion()** method.
- o The new user needs to be written into the user.txt file. All the attributes are **separated by three semicolons - ";;;"**. The registration of different roles could generate different strings.

The format example 1:

```
675531;;;test;;;**t***e***s***t**;;;2022-1-24_12:47:18.244;;;
student;;;test@gmail.com
```

The format example 2:

(username: test_instructor, password: test_instructor)

The 9 semi-colons here indicate there are no values for display_name(str, default value is ""), job_title(str, default value is "") and course_id_list(list, default value is []).

```
196405;;;test_instructor;;;*t***e***s***t***_***i***  
n***s***t***r***u***c***t***o***r***;2022-4-30_19:  
8:29.509;;;instructor;;;test_instructor@gmail.com;;;;;;;;;
```

8. **date_conversion()**-> str.

One positional argument - register_time. The given register_time will be a unix epoch timestamp (milli seconds) and it needs to be converted to format "year-month-day_hour:minute:second.milliseconds". For example, a timestamp 1637549590753 will be converted to str "2021-11-22_13:53:10.753" and returned. The time should be GMT+11 Melbourne timezone.

Refer this link <https://www.unixtimestamp.com/index.php> to check how to convert unix epoch time to human readable format. A method called get_day_from_timestamp(timestamp) is provided in the lib.helper file. By using this method, you can convert the timestamp to the day of month. You can import and use this method in the user.py file. It is not allowed to use any time-related libraries or functions here. You are required to implement the conversion by yourself. Because we use some approximate values like 1 month (30.44 days) = 2629743 seconds, the boundary cases can be ignored like 1st Mar. Only make sure most of the results are correct. If you hope to try to generate a very accurate result, it is great (no extra mark allocated). The final readable time would be GMT +11.

9. **validate_username()**-> bool.

One positional argument - username. The username can only be letters or underscore. If not, return False.

10. **validate_password()**-> bool.

One positional argument - password. The length of password must be greater than or equal to 5. If not, return False.

11. **validate_email()**-> bool.

One positional argument - email. Use regex expressions to check whether the email address is valid or not. The email should end with ".com", contain "@", and have length greater than 8. If not, return False.

12. **clear_user_data()** no return.

This method will remove all the data in the user.txt file.

13. Class variable `current_login_user`.

Default value is None. This variable is used to save the user object(Could be Admin, Instructor or Student object) after login. If the user is not logged in, the web application will redirect the web page to the index page.

3.2.2 Admin class

1. **constructor.**

This method has five positional arguments: `uid(int, default value is -1)`, `username(str, default value is "")`, `password(str, default value is "")`, `register_time(str, default value is "yyyy-MM-dd_HH:mm:ss.SSS")` and `role(str, default value is "admin")`. Admin account does not have an email address attribute.

2. `__str__()`->str.

Return string format example:

```
285108;;;aaaaa;;;**a****a****a****a****a**;;;2021-11-29_32:32:28.590;;;admin
```

3. **register_admin()** no return.

This method will create a new admin account and write this account into the `user.txt` file. This method does not need to call the register method implemented in the User class. And, no validation required for the admin account. The default username and password can be any value predefined by yourself. For example, `username="admin"`, `password="admin"`. Admin account cannot be registered via frontend webpages.

3.2.3 Instructor class

1. **constructor.**

This method has 9 positional arguments: `uid(int, default value is -1)`, `username(str, default value is "")`, `password(str, default value is "")`, `register_time(str, default value is "yyyy-MM-dd_HH:mm:ss.SSS")`, `role(str, default value is "instructor")`, `email(str, default value is "")`, `display_name(str, default value is "")`, `job_title(str, default value is "")` and `course_id_list(list, default value is [])`.

2. `__str__()`->str.

Return string format example:

Example 1 (registered manually, so display_name="", job_title="", course_id_list=[]):

```
945546;;;test;;;&&15&&&&25&&&&35&&&&45&&&&55&&&&65&&&&75&&&&85&&;;;2022-5-3_11:39:11.731;;;instructor;;;12312312@gmail.com;;;;;;;;;;
```

Example 2 (extracted from files):

```
863133505;;;brooks_davila;;;&&85&&&&65&&&&35&&&&15&&&&35&&&&835&&&&55&&&&58&&&&55&&;;;xx-xx-xxxx;;;instructor;;;brooks_davila@udemy.com;;;Brooks Davila;;;Management trainer;;;39385047--112007285--582468331--204107886--970798161--798836200--662704084--378659188
```

3. **get_instructors()** no return.

This method will extract instructor information from the given course data files. Similar to the process of retrieving course data, but this method focuses on the instructor data of each course. In each course item, there could be multiple instructors. There is no need to perform registration validation for each instructor's info in this method. All the null value in json str should be saved as None or string "null". All the empty string value "" should be saved as same empty string "".

After retrieving the required data, you need to write the info into user.txt file to save all the instructor data. Each attribute needs to be separated by ";;".

The required attributes and data format is:

"{instructor_id};;;{username};;;{password};;;{register_time};;;{role};;;{email};;;{instructor_display_name};;;{instructor_job_title};;;{course_id_list}". The

username is generated by converting the display name to lowercase and replacing the whitespace to underscore. The password uses the instructor_id value directly. The email address is generated by combining the username and the "@gmail.com". All the course ids in the course_id_list will be connected to using two "-" marks. The format is "course_id--course_id--course_id". If an instructor is already in the user.txt file, only update the course_id_list which saves the course this instructor teaches. The register_time uses the default value.

The auto-generated instructor accounts are different from the manually registered instructor account. Manually registered instructor accounts do not have any course_id_list. And, the display name and job title are empty.

However, these accounts should have the same format in user.txt as the auto-generated accounts.

For example:

```
945546;;;test;;;&&15&&&&25&&&&35&&&&45&&&&55&&&&65&&&&
75&&&&85&&;;;2022-5-3_11:39:11.731;;;instructor;;;1231
2312@gmail.com;;;;;;;;;;
```

4. **get_instructors_by_page()**->tuple

One positional argument: page. This method reads the user.txt file to retrieve all the instructor information. With all the instructor information and the current page number, a list of Instructor objects and the total pages will be generated. Each page has at most 20 instructors. A tuple contains the list of instructors, total page number and the total number of instructors will be returned.

5. **generate_instructor_figure1()**->str

Generate a graph that shows the top 10 instructors who teach the most courses.(any chart)

In all the graphs, if the instructor display name is too long, you need to extract the first 3 words. The generate_instructor_figure1() method is required to return a string explanation about your understanding of this figure. **All the graphs use the instructor title, course title, category title or subcategory title as x-axis labels.**

3.2.4 Student class

1. **constructor.**

Six positional arguments: uid(int, default value is -1), username(str, default value is ""), password(str, default value is ""), register_time(str, default value is "yyyy-MM-dd_HH:mm:ss.SSS"), role(str, default value is "student"), email(str, default value is "")

2. **__str__()**->str.

Return string format example:

```
675531;;;test;;;**t****e****s****t**;;;2022-1-24_12:47:18.244;;;s
tudent;;;test@gmail.com
```

3. **get_students_by_page()**->tuple.

One positional argument: page. This method reads the user.txt file to retrieve all the student information. With all the student information and the current page number, a list of Student objects and the total pages will be generated. Each page has at most 20 students. A tuple contains the list of students, total page number and the total number of students will be returned.

4. **get_student_by_id()**->Student object

One positional argument id. This method returns a student object by retrieving the id from the user.txt file.

5. **delete_student_by_id()**->bool

One positional argument id. This method deletes a student item from the user.txt file based on the given id.

3.2.5 Course class

1. **constructor.**

Eleven positional arguments: category_title(str, default value is ""), subcategory_id(int, default value is -1), subcategory_title(str, default value is ""), subcategory_description(str, default value is ""), subcategory_url(str, default value is ""), course_id(int, default value is -1), course_title(str, default value is ""), course_url(str, default value is ""), num_of_subscribers(int, default value is 0), avg_rating(float, default value is 0.0) and num_of_reviews(int, default value is 0).

2. **__str__()**->str.

Return format:

```
{category_title};;;{subcategory_id};;;{subcategory_title};;;{subcategory_descrip
tion};;;{subcategory_url};;;{course_id};;;{course_title};;;{course_url};;;{num_of
_subscribers};;;{avg_rating};;;{num_of_reviews}
```

Return string format example:

```
Development;;;8;;;Web Development;;;Learn web development skills
to build fully functioning websites.;;;/courses/development/
web-development/;;;1565838;;;The Complete 2021 Web Development
Bootcamp;;;/course/the-complete-web-development-bootcamp/;;;47482
5;;;4.675107;;;150052
```

3. **get_courses()** no return.

This method will extract course information from the given course data files. In the **source_course_files** folder, there are 4 categories of courses. In each category folder, there are some subcategories. Inside each subcategory folder, you can find the course json files. You need to retrieve the category_title from the 4 category folder names and other course info from the json files. In each json file, there is another category name which is also acceptable for category_title value like the image below.



All the null value in json str should be saved as None or string "null". All the empty string value "" should be saved as same empty string "".

After retrieving the required data, you need to write the info into course.txt file to save all the course data. All the data need to be separated by ";;;". The required attributes and data format is: "{category_title;;;{subcategory_id;;;{subcategory_title;;;{subcategory_description;;;{subcategory_url;;;{course_id;;;{course_title;;;{course_url;;;{num_of_subscribers;;;{avg_rating;;;{num_of_reviews}}".

4. **clear_course_data()** no return.

This method will remove all the content in the course.txt file. After calling this method, the course.txt file will become an empty file.

5. **generate_page_num_list()**->list of int.

Two positional arguments: `page` and `total_pages`. This method uses the current page number and total pages to generate a list of integers as viewable page numbers. For example, the image below shows a default page number list [1,2,3,4,5,6,7,8,9] when the current page number is 1.

Start	...	1	2	3	4	5	6	7	8	9	...	End
-------	-----	---	---	---	---	---	---	---	---	---	-----	-----

If the current page number is less than or equal to 5, the generated page number list is always [1,2,3,4,5,6,7,8,9]. If the current page number is greater than 5 and less than total pages minus 4, the page number list will be integers from current page number minus 4 until current page number plus 4. For example, in the image below, the current page is 8 and the number list becomes [4,5,6,7,8,9,10,11,12].

Start	...	4	5	6	7	8	9	10	11	12	...	End
-------	-----	---	---	---	---	---	---	----	----	----	-----	-----

If the current page is greater than or equal to total pages minus 4, the list of numbers changes to range between total pages minus 8 until total pages.

6. **get_courses_by_page()**->tuple

One positional argument: `page`. The return value is a tuple that contains a list of Course objects, total pages of courses and the total number of courses. This method reads the `course.txt` file to retrieve all the course information. With all the course information and the current page number, a list of Course objects will be generated, the total pages and the total number of courses will be returned. Each page has at most 20 courses.

For example, if there are 100 courses info in the `course.txt` file and the current page number is 2, then the 21-40 lines course info will be converted to a list with 20 Course objects. The total page number is 5.

7. **delete_course_by_id()**->bool

One positional argument: `course_id`. The method reads course info from the `course.txt` file and deletes the course information belongs to that `course_id`. Meanwhile, if an instructor in the `user.txt` file teaches this course, the course id should also be removed from the instructor's `course_id_list`. Finally, this

method returns whether the deletion is successful or not. If the `course_id` cannot be found in the `course.txt` file, return `False`.

8. **`get_course_by_course_id()`**->tuple

One positional argument: `course_id`. You are required to find the course by given `course_id` and convert the info to a `Course` object. Then, using the retrieved course info to get the `num_of_subscribers`, `avg_rating` and `num_of_reviews`. Based on these three numbers, generate a comment for this course. If the `num_of_subscribers` greater than 100000 and `avg_rating` greater than 4.5 and `num_of_reviews` greater than 10000, the comment should be "Top Courses". If the `num_of_subscribers` greater than 50000 and `avg_rating` greater than 4.0 and `num_of_reviews` greater than 5000, the comment should be "Popular Courses". If the `num_of_subscribers` greater than 10000 and `avg_rating` greater than 3.5 and `num_of_reviews` greater than 1000, the comment should be "Good Courses". The other courses are "General Courses". The `Course` object and comment will be returned as a tuple.

9. **`get_courses_by_instructor_id()`**->tuple

One positional argument: `instructor_id`. This method reads the `user.txt` file and `course.txt` file to find all the course information the specified instructor teaches. If this instructor teaches more than 20 courses, only 20 courses will be returned with the total number of courses this instructor teaches (do not need to sort, just use the default order and get the first 20). Otherwise, all the courses and the total number will be returned. The return type is a tuple that contains a list of course objects and the total number of courses taught by this instructor.

10. **`generate_course_figure1()`**->str

Generate a graph to show the top 10 subcategories with the most subscribers. (any chart)

11. **`generate_course_figure2()`**->str

Generate a graph to show the top 10 courses that have lowest avg rating and over 50000 reviews.(any chart)

12. **`generate_course_figure3()`**->str

Generate a graph to show all the courses avg rating distribution and number of subscribers. The courses should have subscribers between 100000 and 10000.
(scatter chart)

13. **generate_course_figure4()**->str

Generate a graph to show the number of courses for all categories and sort in ascending order (pie chart, offsetting the second largest number of course with "explode")

14. **generate_course_figure5()**->str

Generate a graph to show how many courses have reviews and how many courses do not have reviews.(bar chart)

15. **generate_course_figure6()**->str

Generate a graph to show the top 10 subcategories with the least courses (any chart)

In all the graphs, if the course title is too long, you need to extract the first 3 words. If the Course title is "Welcome to introduction to Python", using "Welcome to introduction" is enough. If you have extra time, you can also extract the key word like "introduction to Python". But, this may require a complex process as your program may need to understand the sentence. All the generate_course_figure{1-6} methods are required to **return** a string explanation about your understanding of this figure. All the graphs use the instructor title, course title, category title or subcategory title as x-axis labels.

3.3. Task 2 - Index, user register, login and logout

This part will call the methods implemented in previously designed classes. In the lib/helper.py file, there are two functions called render_result() and render_err_result(), which can be used to return result success or failure for POST methods.

1. **index()** in index_controller. GET request, route is "/".

There is only one method in the index_controller which is the index page of this web application. In this method, it is required to check the class variable User.current_login_user to see if there is any logged user. If there exists a logged user, pass the current_login_user's role to context['current_user_role'].

Otherwise, do nothing. Next, create an admin account and register it manually. Finally, render the "01index.html". Admin account should be generated every time in the index_controller/index() method. But, you need to handle the duplicates. If an admin account already exists with the same username and password in the user.txt file, do not overwrite it.

2. **login()** in user_controller. GET request, route is "/login".
Return the "00login.html" page.
3. **login_post()** in user_controller. POST request, route is "/login".
Get "username", "password" values from the request.values. Use the user validation methods to check the username and password. If all valid, call the authentication method. If username and password belong to a valid user, return the string info of this user. Then, generate a corresponding user object using the generate_user() method and assign this user to the User.current_login_user class variable.
4. **templates/00login.html** page
Create two input boxes here for users to input username and password. The type of username input is text and the type of password input is password. Write your code within the student code comment area.
5. **logout()** in user_controller. GET request, route is "/logout".
Reset the User.current_login_user to None and return the "01index.html" page.
6. **generate_user(login_user_str)** in user_controller.
This method is defined and used only in user_controller for login_post() method. Because after login, it is required to generate a user object(could be Admin, Instructor or Student). Since using child class in parent class will cause exceptions, the User.authenticate_user() method cannot return an Admin/Instructor/Student object directly. So, you need to return a user string in User.authenticate_user() and convert the user string to an object in this generate_user() method. The return object could be Admin()/Instructor()/Student().
7. **register()** in user_controller. GET request, route is "/register".
Return the "00register.html" page.

8. **register_post()** in user_controller. POST request, route is “/register”.
Get “username”, “password”, “email”, “register_time”, “role” values from the request.values. Use the user validation methods to check the username, password and email. If all valid, register this user. Otherwise, return render_err_result(msg=“proper message for users”).
9. **templates/01register.html** page
Create three input boxes for the user to input username, password and email. The type of username and email input are text and the type of password input is password. Write your code within the student code comment area.

3.4. Task 3 - Students page

1. **student_list()** in user_controller. GET request, route is “/student-list”.
Try to write this method by referring to the instructor_list and course_list.
Make sure the context dict has “context['one_page_user_list]”, “context['total_pages]”, “context['page_num_list]”, “context['current_page]”, “context['total_num]”, “context['current_user_role]” has values as the web pages need to show these values and may cause errors if these values are not exist.
2. **student_info()** in user_controller. GET request, route is “/student-info”.
Find a student based on the “id” attribute in the request.values. If not exist, return a new student and set all instance variables with default values. Make sure the “context['current_user_role]” has values.
3. **student_delete()** in user_controller. GET request, route is “/student-delete”.
Delete student according to the “id” attribute in the request.values. This method will return “redirect(url_for(user_page.student_list))” if successful. Otherwise, return “redirect(url_for(index_page.index))”.
4. **templates/10student_list.html** page
Create a table to show the information of users. Each row of table need to show one student info and a details button/link and a delete button/link. The href of the details button is “/user/student-info?id={{user.uid}}” and the href of the delete button is “/user/student-delete?id={{user.uid}}”. Write your code within the given table tag. For example, <a href=“...{{user.uid}}” class=“btn

btn-primary">Details, Delete. When clicking Details link, you will go to student_info page. When clicking Delete link, you will delete one student item. If you do not use class attribute here in the HTML tag <a>, it is also correct.

5. **templates/11student_info.html** page

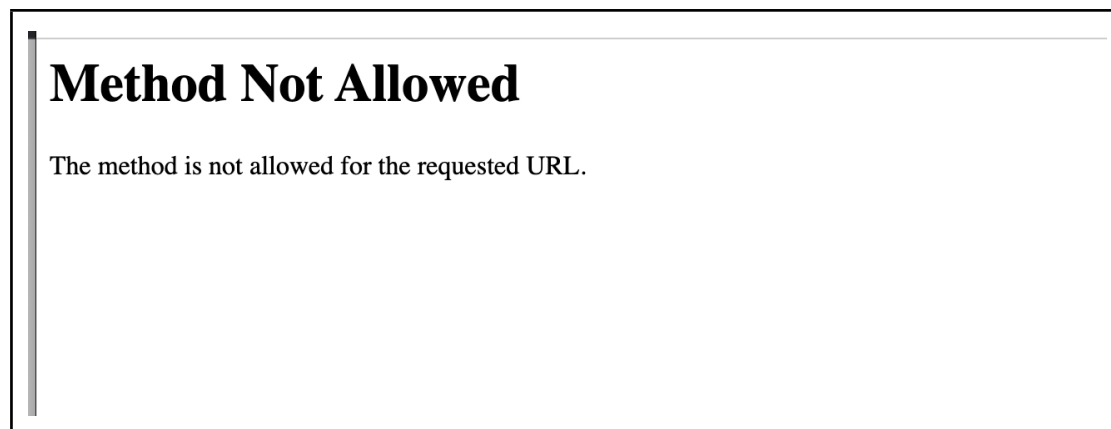
Create a page to have a header print out "Student Info" and all the information of a student below the header. Write your code within the given div tag.

3.5. Task 4 - Courses page

1. **course_list()** in **course_controller**. GET request, route is **"/course-list"**.

In this method, some code is already provided. Add more code to make the method run. If the user is not logged in, should not allow access to the courses and redirect the page to **"index_page.index"**. If there exists a logged user, get the expected course list and page number list. Make sure the **"context["current_user_role"]"** has values.

For Mac users, if you meet the "Method not allowed" error in pages, try to refresh the page. If the result can show properly, there should be no errors in your backend logic and you can ignore this error.



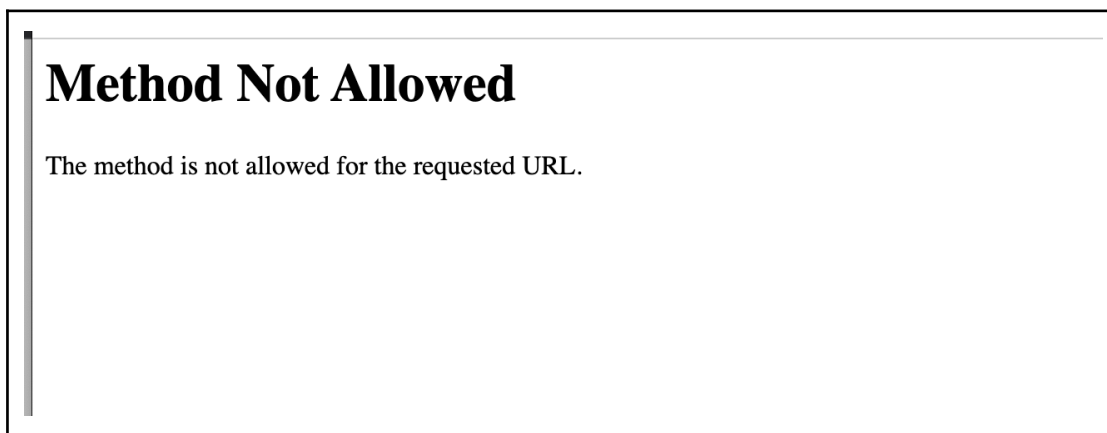
2. **reset_database()** in **course_controller**. POST request, route is **"/reset-database"**.

This method simply removes all the content in the **user.txt** and **course.txt** files by calling methods in **User** and **Course** class. After finish, return a success message. If an error happens, return the **render_err_result(msg="exception happened")** method result. Use try except block in this method to handle exceptions.

3.6. Task 5 - Instructors page

1. **Instructor_list()** in `instructor_controller`. Get request, route is `"/instructor-list"`. In this method, some code is already provided. Add more code to make the method run. If the user is not logged in, should not allow access to the instructors and redirect the page to `"index_page.index"`. If there exists a logged user, get the expected instructor list and page number list. Make sure the `"context["current_user_role"]"` has values.

For Mac users, if you meet the "Method not allowed" error in pages, try to refresh the page. If the result can show properly, there should be no errors in your backend logic and you can ignore this error.



2. **teach_courses()** in `instructor_controller`. GET request, route is `"/teach-courses"`.

In this method, some code is already provided. Add more code to make the method run. If the user is not logged in, should not allow access to the teach courses and redirect the page to `"index_page.index"`. If there exists an `"id"` attribute in the `request.values`, assign this value to the `instructor_id`. If there is no `"id"` attribute in the `request.values`, use the current logged user's id as the `instructor_id`. Then, you can get the teach courses list. Make sure the `"context["current_user_role"]"` has values.

Important Notes:

- If any exception/error happens when running your program, you will lose 50% marks.
- Please refer to the `_demo_{data}.txt` file to see the correct data format
- Your program should also work if you change all the data file paths to demo data file paths.

