Web Technologies Lab

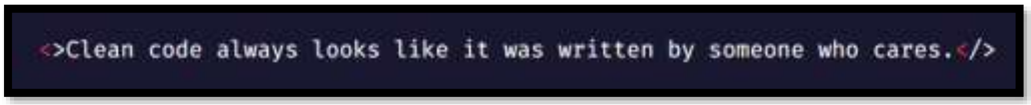
LAB-06 **BIT F20** May 24, 2023

The objective of this lab is to

- Practice Python's web framework Flask.
- Perform insert and select queries on data via flask.
- Use the MVC approach for designing the solution.
- Perform data validation.
- Handle exceptions efficiently for handling unexpected scenarios.

Instructions!

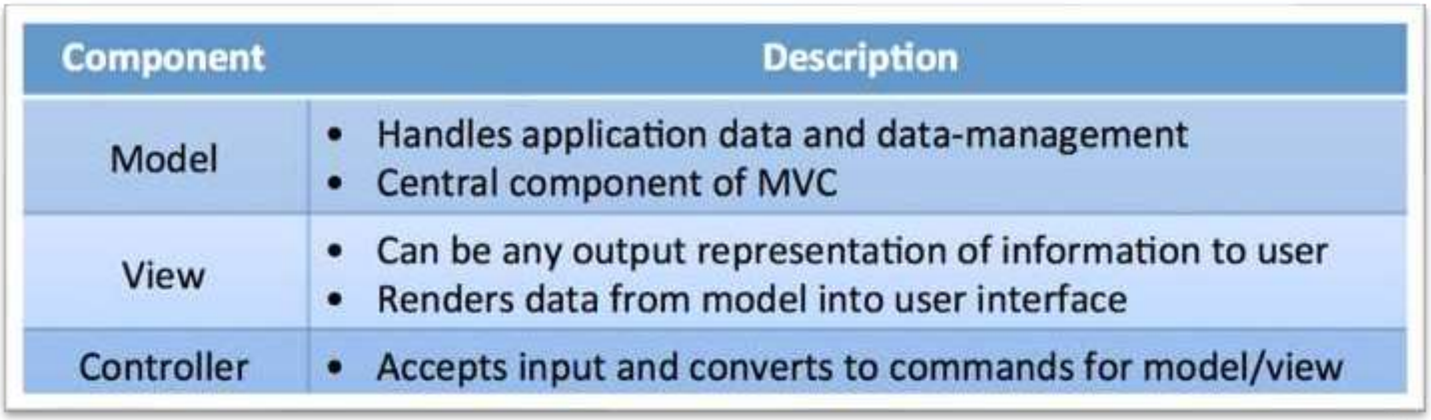
- Keep your student identity cards with you.
- This is an individual lab, you are strictly **NOT** allowed to discuss your solutions with your fellow colleagues, even not allowed to ask how he/she is doing, as it may result in a negative marking.
- You can **ONLY** discuss this with TAs or Ma'am.
- Save your work frequently. Make a habit of pressing CTRL+S after every line of code you write.
- This is a **GRADED** lab, so, at the end of the lab session, you should have your complete work ready for evaluation.
- Follow proper coding conventions and write comments.
- *Total Time for this Lab is 100 minutes.*



```
<>Clean code always looks like it was written by someone who cares.</>
```

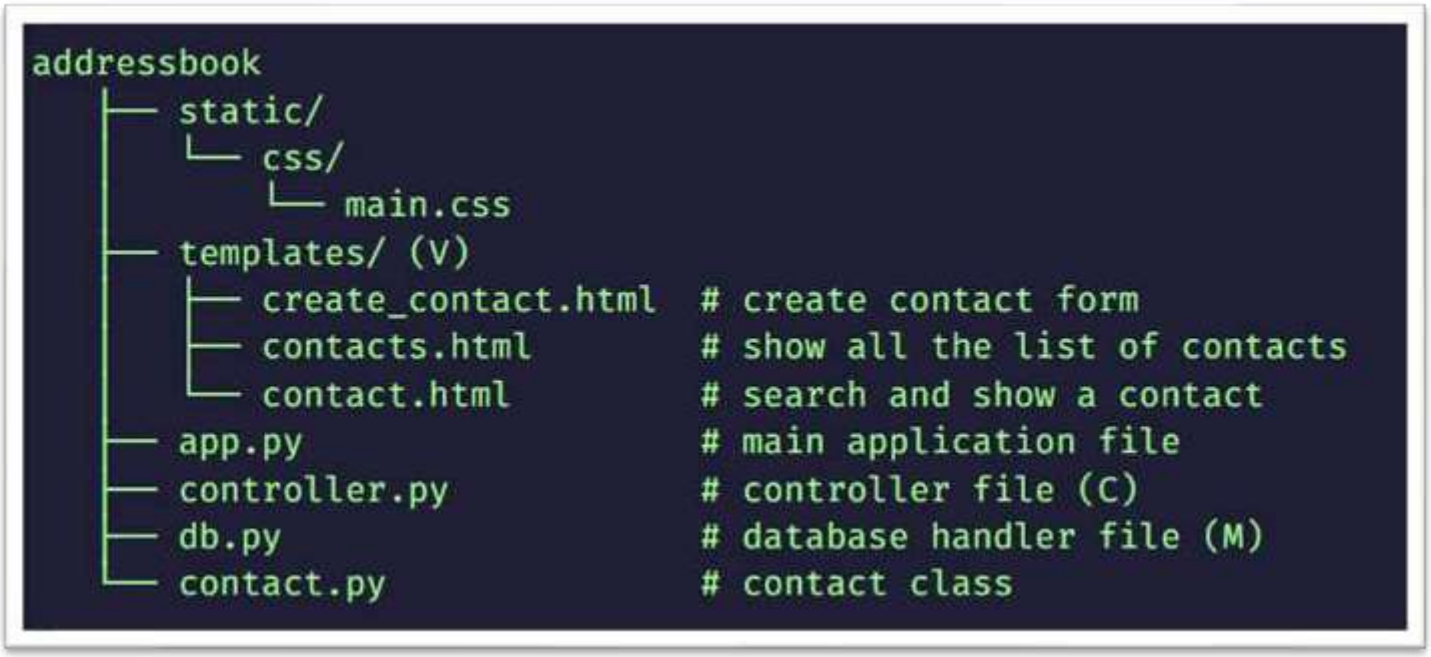
Lab manual: a conceptual overview Read time (10 minutes)

MVC



Component	Description
Model	• Handles application data and data-management • Central component of MVC
View	• Can be any output representation of information to user • Renders data from model into user interface
Controller	• Accepts input and converts to commands for model/view

File Structure of a Flask Application



```
addressbook
    ├── static/
    │   └── css/
    │       └── main.css
    ├── templates/ (V)
    │   ├── create_contact.html  # create contact form
    │   ├── contacts.html        # show all the list of contacts
    │   └── contact.html         # search and show a contact
    ├── app.py                   # main application file
    ├── controller.py            # controller file (C)
    ├── db.py                    # database handler file (M)
    └── contact.py               # contact class
```

1. The static folder contains assets used by the templates, including CSS files, JavaScript files, and images. In the example, we have only one asset file, main.css. Note that it's inside a CSS folder that's inside the static folder.

2. The templates folder contains only templates. These have a *.html* extension. As we will see, they contain more than just regular HTML. These files are named templates because the HTML files are nothing more than the templates unless we populate them with the data from the backend (the flask). {Just like a class in OOP, until an object is created}
3. In addition to the static and templates folders, this app also contains .py files. Note that these must be outside the two folders named static and templates. (ref. 1).

- **Suggests using separate HTML files for each operation for readability and accessibility purposes,but if you can handle things on a single HTML page, that'll be fine too.**

Task 01 **30 Marks (90 Minutes)**

A flask-based address book.

You all are aware of the phone or address book in your cell phones, each contact has different attributes. You'll develop a similar flask web-based address book, that'll let the user perform the following 3 functionalities:

1. **Create a new contact.**
    - *take data from the HTML form, validate it and store it in the database.*
2. **Show all contacts.**
    - *show all the contacts stored in the DB in a well-formatted HTML table.*
3. **Search contact by name/Number.**
    - *take a contact name or number from the HTML form, then search and display the contact on the HTMLpage. Note if user only enter name then it should display the contact that match with the name also for number as well.*

In the database, you'll have the following table:

- **Contacts**

  It represents all the contacts
    - **Id** <INT, pk, not null, unique, auto-increment>
    - **name** <VARCHAR, unique, not null>
        - *to represent the name of the user.*
    - **mobileno** <VARCHAR, not null>
        - *the mobile number of the user.*
    - **city** <VARCHAR, not null>
        - *the city of the user.*
    - **profession** <VARCHAR, not null>
        - *the profession one belongs to.*

Enough Database, now see the explanation for the file structure:

1. Create a flask project named **'addressbook'.**
2. A static folder, if you need to work with any assets like images, css, and js files.
3. A folder **templates**, keep all the HTML files in this folder.
    a. create_contact.html will have an HTML form, that'll post data, that'll be validated and stored in the database.

       You can use JS (if have prior knowledge) and Python (in templates or after request) for validation to make sure:
        - *No empty/null value gets into the database.*
        - *Maintain the uniqueness of 'contact_name'.*
        - *Number format should be like +92 3337675885/ 03337675885.user can enter number in any format. Ensure that user enter correct number.*
4. **app.py** that'll contain the main flask code.
5. **controller.py** will contain *ContactController* class that'll play an intermediary role between *db.py*

6. **app.py**
   the app.py will have the main code (containing configurations and routes), to run the flask application.
7. **db.py** should handle queries, data requests, and retrieval from the database, via a class named as *DBHandler*.
8. **contact.py** to represent the contact class. It'll have the same attributes as the contacts table you've already created.

Grading Criteria

There are some things you've been doing repeatedly, and following them is mandatory, due to their level of being basic, those'll carry no marks, but not following them will result in a negative marking 😊.

Component	**Requirement(s)**	**Marks**
exceptions	Exceptions are handled carefully.	-2
proper names	The proper naming convention is used for variables, functions, classes, etc.	-2
MVC approach	Following the MVC approach, only file names are not going to do any better unless they are performing the functionality, they are meant to be,	1x3 = 3
app.py	Make sure to set the proper: ▪ Configuration. ▪ Routes, rendering, and redirects.	5
templates	▪ Html file(s) are set properly, using the proper HTML elements, tags, and placeholders. (3) ▪ Data is displayed in the table with the proper header. (3)	6
Data validation	Data is validated and then stored in the database. ▪ Unique name for each contact maintained. (3) ▪ No blank values are being dumped into the database. (1)	4
DB.py	Database class is well implemented and used.	3
Controller.py	A controller class is created and well-managed.	3
Flow	For full marks on this section, make sure to: ▪ add proper redirects (if multiple HTML pages are used). ▪ give a better look to the HTML page. ▪ Overall flow is perfectly maintained across the components.	6
Total		30

"Those who are wise won't be busy, and those who are too busy can't be wise."