

Flask – Part 2: Product Page. Forms.

Lab Objectives

Objectives of this lab are to carry on developing our **Online Bookstore**, by:

- adding an individual book page, which is accessed from the home page;
- implementing 'User Account' functionality using Flask forms.

NB: It is advisable to do all the work with your virtual environment **activated**.

General Comments

As before, this exercise is not assessed. If you do not manage to finish all the tasks in the lab, please attempt to finish them in your own time.

Use the the suggested resources and links, provided throughout this document and on the last page, to help you understand the code. A snapshot of the state of the project at the end of this lab is available on Learning Central. You can download this to help you if you are stuck, to check your progress (and, perhaps, at times to save your typing *from scratch*). However, please make sure you understand each line of the code!

Remember, the lab tutors are here to help. If you get stuck – raise your hand and ask for advice. It is also okay to discuss the solutions with your peers (these labs are not assessed!), however, make sure you understand everything by yourself.

IMPORTANT!! The labs will give you some basic understanding of how to develop a website in Flask. To get more comprehensive understanding of how this 'stuff' works, it is strongly advised that you read the recommended book, suggested documentation and 'quickstarts', as well as complete few tutorials. However, these are just suggestions and the list is non-exhaustive! There are lots of other tutorials on the Web.

Abbreviations used in this document

- **dir** – directory (folder)
- **venv** – virtual environment.
- **db** – database
- **NB** – Nota bene

INDIVIDUAL BOOK PAGE

In our online Book Store, each individual book is accessed by using 'dynamic' URLs in the form of `book/<book_id>`, e.g. for the first book in our database with the URL is `book/1`. To enable our website visitors to access each book's page, we need to:

- create a new `book.html` template,
- and then update `home.html`.

1. In `shop/templates` dir, create an empty `book.html`, make sure it inherits all the elements of our site's layout (i.e. navigation, etc.).
2. In `{% block content %}` section of the page, specify that we want the page to display each book's image, titles and author (similar to what we did previously in `home.html` page).
3. In `home.html`, update the template in such a way so that when the user clicks on a book's title that book's individual page is displayed. This is accomplished by using a `href`, e.g.:

```
<a href="{% url_for('book', book_id=book.id) %}">
```

4. Now, we need to update `routes.py` to tell the server where to redirect to `book.html` when the user clicks on the book's title:

```
@app.route("/book/<int:book_id>")
def book(book_id):
    book = Book.query.get_or_404(book_id)
    return render_template('book.html', title=book.title, book=book)
```

5. Test it works by that clicking on a book's title redirects to that book's page.
6. Update `home.html` to also enable the user to click on a book's image to redirect to that book page.

USER ACCOUNTS

We want our customer to create their user account, so that they can register, log in and log out. Later on, we will also be using the user accounts for 'placing an order'. Implementation of the user accounts is accomplished by using **Flask-Login**¹, which will be responsible for handling user authentication. We also need **Flask-WTF**², which is an integration of Flask and **WTForms**³ - for creating forms.

NB: Before you start working on tasks in this section, you need to check if you have these packages installed, and if not, install it using `pip` in your `venv`.⁴

NB: Most of the implementation in this section is based on **Flask-Login**⁵ documentation and Chapter 8 'User Authentication' of M. Grinberg's book "Flask web development", O'Reilly Media, 2014.

Initial Modifications

7. To start with, we need to initiate **Flask-Login** in our app:
 - (a) Open `__init__.py`. Add an import for `LoginManager` and initialise its object:

```
...
from flask_login import LoginManager
...
login_manager = LoginManager()
login_manager.init_app(app)
```

DB Update

8. We will **store the customer registration details** in our db. To enable this, we need to add a new table `User` to the db, by: (a) modifying `models.py`, and (b) writing the changes to the db:

```
(a) class User(UserMixin, db.Model):
    id = db.Column(db.Integer, primary_key=True)
    username = db.Column(db.String(15), unique=True, nullable=False)
    email = db.Column(db.String(120), unique=True, nullable=False)
    password_hash = db.Column(db.String(128))
    password = db.Column(db.String(60), nullable=False)


    def __repr__(self):
        return f"User('{self.username}', '{self.email}')"

    @property
    def password(self):
        raise AttributeError('password is not a readable attribute')

    @password.setter
```

¹<https://flask-login.readthedocs.io/en/latest/>

²<https://flask-wtf.readthedocs.io/en/stable/> 

³<https://wtforms.readthedocs.io/en/stable/> 

⁴**NB:** If you are getting an error message when you use `pip` to install a python package, you might need to use `--user` option, i.e. `pip install --user <PACKAGE>`.

⁵<https://flask-login.readthedocs.io/en/latest/>

```

def password(self, password):
    self.password_hash = generate_password_hash(password)

def verify_password(self, password):
    return check_password_hash(self.password_hash, password)

@login_manager.user_loader
def load_user(user_id):
    return User.query.get(int(user_id))

```

The above code requires two *imports*:

```

from werkzeug.security import generate_password_hash, check_password_hash
from flask_login import UserMixin

```

- (b) Update the db, using the python shell (see the instruction for the previous lab, i.e. "Flask Lab 1", "DATABASE (DB)" section). Confirm that you now have the `User` table created.

User Registration

To register a new user, we need to create a **user registration** form. It will have the following fields: `username`, `email`, `password`, `confirm_password`, and `submit` button.

9. Create a new file `forms.py` in the `shop` dir.

- (a) Start with importing `FlaskForm` from `flask_wtf`:

```

from flask_wtf import FlaskForm

```

- (b) Use class `RegistrationForm(FlaskForm)` to declare the first field, `username`, as follows:

```

class RegistrationForm(FlaskForm):
    username = StringField('Username',
                           validators=[DataRequired(), Length(min=3, max=15)])

```

NB: this statement requires several *imports*, such as: `StringField` from `wtforms`, and `DataRequired` and `Length` from `wtforms.validators`, i.e.:

```

from wtforms import StringField
from wtforms.validators import DataRequired, Length

```

- (c) Similarly, create the fields for: `email`, `password`, `confirm_password`, and `submit` button:

```

...
email = StringField('Email',
                    validators=[DataRequired(), Email()])
password = PasswordField('Password', validators=[DataRequired()])
confirm_password = PasswordField('Confirm Password',
                                  validators=[DataRequired(), EqualTo('password')])
submit = SubmitField('Register')

```

- (d) add the necessary imports from `wtforms` and `wtforms.validators`, i.e. `PasswordField`, etc.

10. The next step is to add the form to the registration page - `register.html`.

- (a) Create `register.html` in `shop/templates` dir.
- (b) In `{% block content %}` section, specify that we want to add the form and its field `username` ⁶:

```
...
<form method="POST" action="">
    {{ form.csrf_token }}
    {{ form.username.label }} {{ form.username}}
    <input type="submit" value="Register">
</form>
...
```

- (c) Using the same principle, add all other form fields (`email`, etc.).

11. To process the form, we need to modify `routes.py` to tell the server how to handle the form, by adding `@app.route` decorator for `register` :

```
(a) ...
@app.route("/register", methods=['GET', 'POST'])
def register():
    form = RegistrationForm()
    if request.method == 'POST':
        user = User(username=form.username.data,
                    email=form.email.data, password=form.password.data)
        db.session.add(user)
        db.session.commit()
        return redirect(url_for('home'))
    return render_template('register.html', title='Register', form=form)
...
```

12. Test the registration process works as intended, by going to `http://127.0.0.1:5000/register` and create few 'users'. Check these users' details are now listed in your db, in `user` table.

NB: For the moment, we are assuming the user only provides valid input. Handling user input's validity and errors is the subject of the next lab.

13. Modify the model by adding additional attributes, such as the user's first and last name and address, and then update your db. Add these fields to the form and display them on the `register` page.
14. Currently, when users registers successfully, they are taken to the home page. Change the redirection, so that after they have registered, instead, they are taken to a *'Thank you for registering'* page.

⁶If you are curious about what "form.csrf_token" is about, see here: <https://wtforms.readthedocs.io/en/stable/csrf.html>

User Login

User Login functionality is implemented, using similar principles we followed when we implemented **User Registration**, i.e. you need to create `LoginForm` class in `forms.py`, create `login.html` template, and add logic of how to handle login to `routes.py`.

15. Update `forms.py`, by creating `LoginForm`:

```
class LoginForm(FlaskForm):
    email = StringField('Email', validators=[DataRequired(), Email()])
    password = PasswordField('Password', validators=[DataRequired()])
    submit = SubmitField('Login')
```

16. Create new `login.html`, and add the form to display the user's email and password fields (similar to 10).

17. Modify `routes.py`:

(a) Add `@app.route("/login")`:

```
...
@app.route("/login", methods=['GET', 'POST'])
def login():
    form = LoginForm()
    if request.method == 'POST':
        user = User.query.filter_by(email=form.email.data).first()
        if user is not None and user.verify_password(form.password.data):
            login_user(user)
            return redirect(url_for('home'))
    return render_template('login.html', title='Login', form=form)
...
```

(b) Add all the necessary *imports*, including import of `LoginForm`.

18. Test the login process works as intended, by going to `http://127.0.0.1:5000/login` and check you can log in as one of the 'user' you added to the database in Task 12. On successful login the website will return to home page.

NB: As before, for the moment, we are assuming the user only provides valid input. Handling user input validity and errors is covered in the next lab.

User Logout

19. You might be pleased to know that '**user logout**' functionality is, perhaps, the easiest of all to implement. We don't need to have a special form for this, just modification of `routes.py`, to which we need to add the `@app.route("/logout")` decorator:

```
...
@app.route("/logout")
def logout():
    logout_user()
    return redirect(url_for('home'))
```

NB: make sure you to add all the necessary import(s) to '`from flask_login import ...`' statement at the top of the file.

20. Similar to our implementation of `login` functionality, on successful logout we are returning the user to the home page. To test whether the logout functionality works, we can: either create a 'successful logout' page and modify `routes.py` to redirect to it, or modify the website navigation to include the menu items for registration, logging in and logging out, and then deploy some logic so that the home page will reflect the current state of the user - see the next section.

NAVIGATION ENHANCEMENT

21. Modify `layout.html` to enhance the website navigation by providing your website visitors with the links to `register`, `login` and `logout`. The website should display the links appropriate to each user, e.g. a `'guest'` user should see `register` and `login`, while the logged-in user should see `logout`. You could also add a `Hello, <USER NAME>` greeting to the navigation bar to personalise the website for the logged-in user. Whilst, the default greeting would be `Hello, Guest!`.

Hint: `current_user` proxy allows you to access the logged-in user - see: <https://flask-login.readthedocs.io/en/latest/>^[1]. Also, check out the examples provided here: <http://flask.pocoo.org/docs/1.0/tutorial/templates/>^[2].

Useful resources	
Flask Website:	http://flask.pocoo.org/docs/1.0/ ^[1]
Flask Tutorial:	http://flask.pocoo.org/docs/1.0/tutorial/ ^[2]
Flask-WTF home page:	https://flask-wtf.readthedocs.io/en/stable/ ^[3]
Flask-WTF Quickstart:	https://flask-wtf.readthedocs.io/en/stable/quickstart.html ^[4]
Book on Flask:	M. Grinberg's (2014). "Flask web development", O'Reilly Media.
M. Grinberg's tutorial on 'User Logins' (<i>more or less the same as in the book above</i>):	https://blog.miguelgrinberg.com/post/the-flask-mega-tutorial-part-v-user-logins ^[5]
Plenty of video tutorials on YouTube (<i>as usual, in no particular order!</i>), e.g.:	[1], [2] (these links are to the first lessons in a series; other episodes cover the flask forms).
A number of cheat sheets could be found on the web.	
