**Storing Python Objects in a MySQL data base**

Using MySQL within Flask applications

First we need to install a packagfe top link Python objects to a data base.

We are going to use sqlalchemy

From the MAC terminal prompt:

Pauls-MacBook-Pro-2:get-it-done paulcorrao$ source activate flask-env

(flask-env) Pauls-MacBook-Pro-2:get-it-done paulcorrao$ conda install -c conda-forgflask-sqlalchemy

Note – you must be in the flask virtual environment ( source activate flask-env)

(Our project folder is lc101/get-it-done)

The output is:

Fetching package metadata .............

Solving package specifications: .

Package plan for installation in environment /Users/paulcorrao/miniconda3/envs/flask-env:

The following NEW packages will be INSTALLED:

flask-sqlalchemy: 2.3.2-py\_0 conda-forge

sqlalchemy: 1.1.13-py36\_0 conda-forge

Proceed ([y]/n)? y

sqlalchemy-1.1 100% |#####################################| Time: 0:00:01 1.49 MB/s

flask-sqlalche 100% |#####################################| Time: 0:00:00 2.09 MB/s

(flask-env) Pauls-MacBook-Pro-2:get-it-done paulcorrao$

Next we have to install the python specific application to connect to a data base:

(flask-env) Pauls-MacBook-Pro-2:get-it-done paulcorrao$ conda install pymysql

Fetching package metadata ...........

Solving package specifications: .

Package plan for installation in environment /Users/paulcorrao/miniconda3/envs/flask-env:

The following NEW packages will be INSTALLED:

pymysql: 0.7.11-py36h75d80ff\_0

Proceed ([y]/n)? y

pymysql-0.7.11 100% |#####################################| Time: 0:00:00 1.41 MB/s

(flask-env) Pauls-MacBook-Pro-2:get-it-done paulcorrao$

IN MAMP – add user:

We added user get-it-done and pw beproductive

And checked the box to add the data base with the same name and give the user all permissions

In your python app you have to add these lines in order to connect the Python app to the data base:

from flask\_sqlalchemy import SQLALchemy

app.config["SQLALCHEMY\_DATABASE\_URI"] = "mysql+pymysql://get-it-done:get-it-done@localhost:8889/get-it-done"

app.config["SQLALCHEMY\_ECHO"] = True

db = SQLALchemy(app)

class Task(db.Model):

id = db.Column(db.Integer, primary\_key = True)

name = db.Column(db.String(120))

def \_\_init\_\_(self,name):

self.name = name

in the app.config statement:

the first “get-it-done” is the user name you set up in MAMP

the second “get-it-done” is the password you set up in MAMP

@localhost specifies the server wherfe the dara base resides

and 8889 is the port. MAMP uses 8889 as its default port for mysql

and the final get-it-done is the name of the data base.

The SQLALCHEMY\_ECHO config is good for learning how apps connect to data bases and it’s good for debugging. This will echo the SQL commands that SQLALCHEMY generates during processing

The db variable creates a data base object that will be used to interface with the data base via Python code

We create a class called Task – this is to allow each row or record in our data base table be a Python object.

class Task(db.Model):

This extends the functionality of db.Model within our SQLALchemy class (This is not a variable – you must do \*.Model where \* is the variable you set up. In our example the variable name we set up is db

We set up two columns in the row:

id and name.

id is set up as an integer and as the primary key in the table

name is set up as a string, with a max length of 120 characters

def \_\_init\_\_(self,name):

self.name = name

is used to set up a constructor – this must be included in the class definition.

if \_\_name\_\_ =="\_\_main\_\_":

app.run()

we put the app.run() inside this if statement so that it only runs when the main.py program is running. This allows us to import this code to other Python programs without starting up a full blown flask application.

Now at the terminal prompt: To create the table Task, enter data into the table and see what was created:

(flask-env) Pauls-MacBook-Pro-2:get-it-done paulcorrao$ python

>>>from main import db, Task

/Users/paulcorrao/miniconda3/envs/flask-env/lib/python3.6/site-packages/flask\_sqlalchemy/\_\_init\_\_.py:794: FSADeprecationWarning: SQLALCHEMY\_TRACK\_MODIFICATIONS adds significant overhead and will be disabled by default in the future. Set it to True or False to suppress this warning.

'SQLALCHEMY\_TRACK\_MODIFICATIONS adds significant overhead and '

>>> db

<SQLAlchemy engine=mysql+pymysql://get-it-done:\*\*\*@localhost:8889/get-it-done?charset=utf8>

>>> db.create\_all()

2018-01-08 21:30:50,537 INFO sqlalchemy.engine.base.Engine SHOW VARIABLES LIKE 'sql\_mode'

2018-01-08 21:30:50,537 INFO sqlalchemy.engine.base.Engine {}

2018-01-08 21:30:50,538 INFO sqlalchemy.engine.base.Engine SELECT DATABASE()

2018-01-08 21:30:50,539 INFO sqlalchemy.engine.base.Engine {}

2018-01-08 21:30:50,539 INFO sqlalchemy.engine.base.Engine show collation where `Charset` = 'utf8' and `Collation` = 'utf8\_bin'

2018-01-08 21:30:50,540 INFO sqlalchemy.engine.base.Engine {}

2018-01-08 21:30:50,541 INFO sqlalchemy.engine.base.Engine SELECT CAST('test plain returns' AS CHAR(60)) AS anon\_1

2018-01-08 21:30:50,542 INFO sqlalchemy.engine.base.Engine {}

2018-01-08 21:30:50,542 INFO sqlalchemy.engine.base.Engine SELECT CAST('test unicodereturns' AS CHAR(60)) AS anon\_1

2018-01-08 21:30:50,542 INFO sqlalchemy.engine.base.Engine {}

2018-01-08 21:30:50,543 INFO sqlalchemy.engine.base.Engine SELECT CAST('test collated returns' AS CHAR CHARACTER SET utf8) COLLATE utf8\_bin AS anon\_1

2018-01-08 21:30:50,543 INFO sqlalchemy.engine.base.Engine {}

2018-01-08 21:30:50,544 INFO sqlalchemy.engine.base.Engine DESCRIBE `task`

2018-01-08 21:30:50,544 INFO sqlalchemy.engine.base.Engine {}

2018-01-08 21:30:50,545 INFO sqlalchemy.engine.base.Engine ROLLBACK

2018-01-08 21:30:50,546 INFO sqlalchemy.engine.base.Engine

CREATE TABLE task (

id INTEGER NOT NULL AUTO\_INCREMENT,

name VARCHAR(120),

PRIMARY KEY (id)

)

2018-01-08 21:30:50,546 INFO sqlalchemy.engine.base.Engine {}

2018-01-08 21:30:50,572 INFO sqlalchemy.engine.base.Engine COMMIT

# These next commands queue up data to be added to the table

>>> new\_task = Task("finish ORM Lesson 2")

>>> db.session.add(new\_task)

>>> another\_task = Task("post lesson video")

>>> db.session.add(another\_task)

This command actually processes the queue and enters the data into the table

>>> db.session.commit()

2018-01-08 21:38:19,659 INFO sqlalchemy.engine.base.Engine BEGIN (implicit)

2018-01-08 21:38:19,660 INFO sqlalchemy.engine.base.Engine INSERT INTO task (name) VALUES (%(name)s)

2018-01-08 21:38:19,660 INFO sqlalchemy.engine.base.Engine {'name': 'finish ORM Lesson 2'}

2018-01-08 21:38:19,662 INFO sqlalchemy.engine.base.Engine INSERT INTO task (name) VALUES (%(name)s)

2018-01-08 21:38:19,662 INFO sqlalchemy.engine.base.Engine {'name': 'post lesson video'}

2018-01-08 21:38:19,663 INFO sqlalchemy.engine.base.Engine COMMIT

# This command displays the data in the table

>>> Task.query.all()

2018-01-08 21:40:59,527 INFO sqlalchemy.engine.base.Engine BEGIN (implicit)

2018-01-08 21:40:59,528 INFO sqlalchemy.engine.base.Engine SELECT task.id AS task\_id, task.name AS task\_name

FROM task

2018-01-08 21:40:59,528 INFO sqlalchemy.engine.base.Engine {}

[<Task 1>, <Task 2>] # NOTE – there are two tasks in the table – we added these earlier

>>> tasks = Task.query.all() # here we are putting the results of the query into a variable called tasks

>>> tasks # here we output the variable tasks and see that there are two tasks/entries in the table

[<Task 1>, <Task 2>]

# the following commands display the contents of the name field in both entries in the data base/table

>>> tasks[0].name

'finish ORM Lesson 2'

>>> tasks[1].name

'post lesson video'

>>>

Now that we have created objects, when we want to display them in our HTML form we need to change this:

<ul>

{% for task in tasks %}

<li>{{task}}</li>

{% endfor %}

</ul>

To this:

<ul>

{% for task in tasks %}

<li>{{task.name}}</li>

{% endfor %}

</ul>

Notice we now have to specify the “name field to actually get the name displayed.

If we were outputting a string or a list we wouldn’t need to specify .name

In order to add an item to our data base:

if request.method == "POST":

task\_name = request.form["task"]

task\_name = request.form["task"]

db.session.add(new\_task)

db.session.commit()

First we get the input from the user:

task\_name = request.form["task"]

then we create an object from that info as we prep it for entry into the data base

task\_name = request.form["task"]

Then we add that to the queue to be entered into our data base:

db.session.add(new\_task)

and finally, we process the queue to add the objects to the dara base:

db.session.commit()

Now to add a button that says “Done!” next to each item on the list :

<li>{{task.name}}

<form method = "POST" action = "/delete-taks" style="display:inline-block">

<input type = "hidden" name="task\_id" value="{{task.id}}"/>

<input type="submit" value="Done!"/>

</form>

</li>

to make the button appear next to t4eh name of the task in our data base:

<form method = "POST" action = "/delete-taks" style="display:inline-block">

To tell ourselves the ID of any task the user pushes the “Done!” button without displaying the ID to the user:

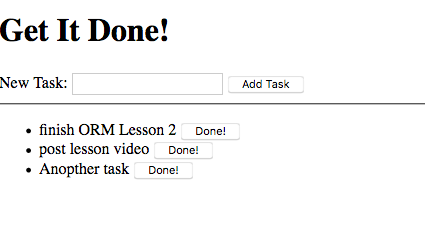
<input type = "hidden" name="task\_id" value="{{task.id}}"/>

and finally to create and display the “Done!” button and close the form

<input type="submit" value="Done!"/>

</form>

This results in this:



Now we have to add the delete-task handler to actually remove the item/row/object from the data base:

@app.route("/delete-task", methods = ["POST"])

def delete\_task():

task\_id = int( request.form["task-id"] )

task = Task.query.get(task\_id)

db.session.delete(task)

db.session.commit()

return redirect("/")

so, first we get the task\_id. Remember we told ourselves what that is with the input type = “hidden” statement in the todos.html form

task\_id = int( request.form["task-id"] )

We had to cast that as integer since it is returned to us as type string and the ID in the data base is of type integer

We then put that ID into a variabvle called rask

task = Task.query.get(task\_id)

We then tell the data base via out SQLAlchemy toqueue the item up for deletion and then to actually delete it:

db.session.delete(task)

db.session.commit()

finally we display the main page back to the user via a redirect statement

return redirect("/")

Next up – instead of actually deleting the record from the data base, we will flag it to indicate the user deleted the row/object and only display those records/rows/objects that the user did not want deleted. In this manner, we preserve the original “user deleted” row/object in the data base.

So, first we add another variable to our class:

class Task(db.Model):

id = db.Column(db.Integer, primary\_key = True)

name = db.Column(db.String(120))

completed = db.Column(db.Boolean, default = False)

def \_\_init\_\_(self, name):

self.name = name

self.completed = False

completed is its name, and it is a Boolean type, with default value False. Note we could also have used the

self.completed = False

statement in the constructor.

Now we go back to Python shell:

>>> from main import db, Task

/Users/paulcorrao/miniconda3/envs/flask-env/lib/python3.6/site-packages/flask\_sqlalchemy/\_\_init\_\_.py:794: FSADeprecationWarning: SQLALCHEMY\_TRACK\_MODIFICATIONS adds significant overhead and will be disabled by default in the future. Set it to True or False to suppress this warning.

'SQLALCHEMY\_TRACK\_MODIFICATIONS adds significant overhead and '

Now we delete the table in its entirety

>>> db.drop\_all()

2018-01-09 12:25:50,852 INFO sqlalchemy.engine.base.Engine SHOW VARIABLES LIKE 'sql\_mode'

2018-01-09 12:25:50,852 INFO sqlalchemy.engine.base.Engine {}

2018-01-09 12:25:50,853 INFO sqlalchemy.engine.base.Engine SELECT DATABASE()

2018-01-09 12:25:50,853 INFO sqlalchemy.engine.base.Engine {}

2018-01-09 12:25:50,853 INFO sqlalchemy.engine.base.Engine show collation where `Charset` = 'utf8' and `Collation` = 'utf8\_bin'

2018-01-09 12:25:50,854 INFO sqlalchemy.engine.base.Engine {}

2018-01-09 12:25:50,855 INFO sqlalchemy.engine.base.Engine SELECT CAST('test plain returns' AS CHAR(60)) AS anon\_1

2018-01-09 12:25:50,855 INFO sqlalchemy.engine.base.Engine {}

2018-01-09 12:25:50,856 INFO sqlalchemy.engine.base.Engine SELECT CAST('test unicode returns' AS CHAR(60)) AS anon\_1

2018-01-09 12:25:50,856 INFO sqlalchemy.engine.base.Engine {}

2018-01-09 12:25:50,856 INFO sqlalchemy.engine.base.Engine SELECT CAST('test collated returns' AS CHAR CHARACTER SET utf8) COLLATE utf8\_bin AS anon\_1

2018-01-09 12:25:50,857 INFO sqlalchemy.engine.base.Engine {}

2018-01-09 12:25:50,857 INFO sqlalchemy.engine.base.Engine DESCRIBE `task`

2018-01-09 12:25:50,857 INFO sqlalchemy.engine.base.Engine {}

2018-01-09 12:25:50,859 INFO sqlalchemy.engine.base.Engine

DROP TABLE task

2018-01-09 12:25:50,859 INFO sqlalchemy.engine.base.Engine {}

2018-01-09 12:25:50,861 INFO sqlalchemy.engine.base.Engine COMMIT

>>> db.drop\_all()

2018-01-09 12:27:14,639 INFO sqlalchemy.engine.base.Engine DESCRIBE `task`

2018-01-09 12:27:14,639 INFO sqlalchemy.engine.base.Engine {}

2018-01-09 12:27:14,641 INFO sqlalchemy.engine.base.Engine ROLLBACK

This completely deletes the table Task from our data base get-it-done

Next we create the table again

>>> db.create\_all()

2018-01-09 12:36:06,501 INFO sqlalchemy.engine.base.Engine DESCRIBE `task`

2018-01-09 12:36:06,501 INFO sqlalchemy.engine.base.Engine {}

2018-01-09 12:36:06,503 INFO sqlalchemy.engine.base.Engine ROLLBACK

2018-01-09 12:36:06,504 INFO sqlalchemy.engine.base.Engine

CREATE TABLE task (db.drop.all()

id INTEGER NOT NULL AUTO\_INCREMENT,

name VARCHAR(120),

completed BOOL,

PRIMARY KEY (id),

CHECK (completed IN (0, 1))

)

2018-01-09 12:36:06,504 INFO sqlalchemy.engine.base.Engine {}

2018-01-09 12:36:06,516 INFO sqlalchemy.engine.base.Engine COMMIT

>>>

Of course this is brute force – what we really want is to change the model of the data base without losing all of the data:

For this we would use a tool such as flash-migrate (this is not covered in this document)

Now, if we want to show the to do list with the items yet to be done and a list of items already completed:

tasks = Task.query.filter\_by(completed = False).all()

completed\_tasks = Task.query.filter\_by(completed = True).all()

return render\_template("todos.html", title = "Get It Done!",

tasks = tasks, completed\_tasks = completed\_tasks)

We update our query to only retrieve those records where completed = False

So it goes from this:

tasks = Task.query..all()

To this:

tasks = Task.query.filter\_by(completed = False).all()

we then do another query to retrieve only those records where completed = True

completed\_tasks = Task.query.filter\_by(completed = True).all()

and we have to pass the completed items to our htm,l form so we add this:

eturn render\_template("todos.html", title = "Get It Done!",

tasks = tasks, completed\_tasks = completed\_tasks)

and in our todos.html file we add this:

<hr />

<h2>Completed</h2>

<ul>

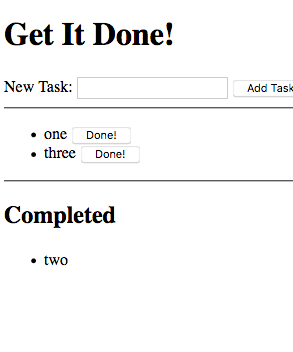
{% for task in completed\_tasks %}

<li>{{task.name}}</li>

{% endfor %}

</ul>

and our output looks like this:



Here are the complete files:

main.py

from flask import Flask, request, redirect, render\_template

from flask\_sqlalchemy import SQLAlchemy

app = Flask(\_\_name\_\_)

app.config["DEBUG"] = True

app.config["SQLALCHEMY\_DATABASE\_URI"] = "mysql+pymysql://get-it-done:beproductive@localhost:8889/get-it-done"

app.config["SQLALCHEMY\_ECHO"] = True

db = SQLAlchemy(app)

class Task(db.Model):

id = db.Column(db.Integer, primary\_key = True)

name = db.Column(db.String(120))

completed = db.Column(db.Boolean, default = False)

def \_\_init\_\_(self, name):

self.name = name

self.completed = False

@app.route("/delete-task", methods = ["POST"])

def delete\_task():

task\_id = int( request.form["task-id"] )

task = Task.query.get(task\_id)

task.completed = True

db.session.add(task)

db.session.commit()

return redirect("/")

@app.route("/", methods = ["POST", "GET"])

def index():

if request.method == "POST":

task\_name = request.form["task"]

new\_task = Task(task\_name)

db.session.add(new\_task)

db.session.commit()

tasks = Task.query.filter\_by(completed = False).all()

completed\_tasks = Task.query.filter\_by(completed = True).all()

return render\_template("todos.html", title = "Get It Done!",

tasks = tasks, completed\_tasks = completed\_tasks)

if \_\_name\_\_ == "\_\_main\_\_":

app.run()

base.html

<!doctype html>

<html>

<head>

<title>{{title}}</title>

</head>

<body>

{% block content %}

{% endblock %}

</body>

</html>

todos.html

{% extends "base.html" %}

{% block content %}

<h1>Get It Done!</h1>

<form method='post'>

<label>

New Task:

<input type="text" name='task' />

</label>

<input type="submit" value='Add Task' />

</form>

<hr />

{% if tasks|length == 0 %}

<p>No tasks yet</p>

{% else %}

<ul>

{% for task in tasks %}

<li>{{task.name}}

<form method = "POST" action = "/delete-task" style="display:inline-block">

<input type = "hidden" name="task-id" value="{{task.id}}"/>

<input type="submit" value="Done!"/>

</form>

</li>

{% endfor %}

</ul>

{% endif %}

<hr />

<h2>Completed</h2>

<ul>

{% for task in completed\_tasks %}

<li>{{task.name}}</li>

{% endfor %}

</ul>

{% endblock %}

**object-relational mapping (ORM)**.

**Working with Users**

**USER Model Class**

Need to have:

* + Base.html file

<!doctype html>

<html>

<head>

<title>{{title}}</title>

</head>

<body>

{% block content %}

{% endblock %}

</body>

</html>

* + login.html file

{% extends "base.html" %}

{% block content %}

<h1>Log in to Get It Done!</h1>

<p>If you don't have an account, you may <a href="/register">register</a> for one</p>

<form action="/login" method="post">

<table>

<tr>

<td><label for="email">Email</label></td>

<td>

<input name="email" type="email" value="{{ email }}">

</td>

</tr>

<tr>

<td><label for="password">Password</label></td>

<td>

<input name="password" type="password">

</td>

</tr>

</table>

<input type="submit" value="Login">

</form>

{% endblock %}

* + register.html file

{% extends "base.html" %}

{% block content %}

<h1>Register to Get It Done!</h1>

<form action="/register" method="post">

<table>

<tr>

<td><label for="email">Email</label></td>

<td>

<input name="email" type="email" value="{{ email }}">

</td>

</tr>

<tr>

<td><label for="password">Password</label></td>

<td>

<input name="password" type="password">

</td>

</tr>

<tr>

<td><label for="verify">Verify Password</label></td>

<td>

<input name="verify" type="password">

</td>

</tr>

</table>

<input type="submit" value="Register">

</form>

{% endblock %}

* + User class in Python program (main.py)

class User(db.Model):

id = db.Column(db.Integer, primary\_key = True)

email = db.Column(db.String(120), unique = True)

password = db.Column(db.String(120))

def \_\_init\_\_(self, email, password):

self.email = email

self.password = password

Then you need to create the user table in your data base using python shell

>>>python

>>>from main import db, User

/Users/paulcorrao/miniconda3/envs/flask-env/lib/python3.6/site-packages/flask\_sqlalchemy/\_\_init\_\_.py:794: FSADeprecationWarning: SQLALCHEMY\_TRACK\_MODIFICATIONS adds significant overhead and will be disabled by default in the future. Set it to True or False to suppress this warning.

'SQLALCHEMY\_TRACK\_MODIFICATIONS adds significant overhead and '

>>> db.create\_all()

Then you create a new user from the Python shell:

>>>new\_user = User([chris@launchcode.orgm](mailto:chris@launchcode.orgm) “cheese”)

>>>db.session.add(new\_user)

>>>db.session.commit()

018-01-10 13:08:54,137 INFO sqlalchemy.engine.base.Engine BEGIN (implicit)

2018-01-10 13:08:54,138 INFO sqlalchemy.engine.base.Engine INSERT INTO user (email, password) VALUES (%(email)s, %(password)s)

2018-01-10 13:08:54,138 INFO sqlalchemy.engine.base.Engine {'email': 'chris@launchcode.org', 'password': 'cheese'}

2018-01-10 13:08:54,140 INFO sqlalchemy.engine.base.Engine COMMIT

>>>

**Working with Users**

# Login and Register Handlers

We need to display the form for login and then have a login handler in our Python Code:

@app.route("/login",methods = ["POST", "GET"])

def login():

if request.method == "POST":

email = request.form["email"]

password = request.form["password"]

user=User.query.filter\_by(email=email).first()

if user and user.password == password:

#TODO remember that the user has logged in

return redirect("/")

else:

#TODO - explain why login failed

return "<h1>Error!</h1>"

return render\_template("login.html")

First, we need to make sure we specify POST and GET methods since this request will come to us as a POST and when you specify POST, if you want GET (which we do) you need to specify that as well:

@app.route("/login",methods = ["POST", "GET"])

We then retrieve the input from the user:

email = request.form["email"]

password = request.form["password"]

Then we check the data base to see if this user exists:

user=User.query.filter\_by(email=email).first()

Note the use of first – we expect only one entry so our query specifies we only want the first occurrence where we find the user inputted email in our data base

If we find the email in the data base and the password entered is correct we return the user to our main page showing the tasks, etc. If we don’t find the email then we return an error

if user and user.password == password:

#TODO remember that the user has logged in

return redirect("/")

else:

#TODO - explain why login failed

return "<h1>Error!</h1>"

finally, if the request method is not POST we render the login form again

return render\_template("login.html")

**Register**

@app.route("/register", methods = ["POST", "GET"])

def register():

if request.method == "POST":

email = request.form["email"]

password = request.form["password"]

verify = request.form["verify"]

#TODO - validte user data (User-signup project has this code)

existing\_user=User.query.filter\_by(email=email).first()

if not existing\_user:

new\_user=User(email,password)

db.session.add(new\_user)

db.session.commit()

return redirect("/")

else:

#TODO use better response messaging

return "<h1>Duplicate user</h1>"

First, we need to make sure we specify POST and GET methods since this request will come to us as a POST and when you specify POST, if you want GET (which we do) you need to specify that as well:

@app.route("/register", methods = ["POST", "GET"])

Then we check the data base to see if this user exists:

existing\_user=User.query.filter\_by(email=email).first()

if this query returns no record, meaning the email does not exist in the data base, then we can go ahead and add this user to the data base and return the user to the main page where all the info is.

if not existing\_user:

new\_user=User(email,password)

db.session.add(new\_user)

db.session.commit()

return redirect("/")

if the email is already in the data base, then we return an error

else:

#TODO use better response messaging

return "<h1>Duplicate user</h1>"

**Managing login using the session**

1. remember that the user has logged in or already registered via session capability

**Remember the user has** **registered:**

def register():

if request.method == "POST":

email = request.form["email"]

password = request.form["password"]

verify = request.form["verify"]

#TODO - validte user data (User-signup project has this code)

existing\_user=User.query.filter\_by(email=email).first()

if not existing\_user:

new\_user=User(email,password)

db.session.add(new\_user)

db.session.commit()

session["email"] = email

return redirect("/")

else:

#TODO use better response messaging

return "<h1>Duplicate user</h1>"

in the register function in our python code we insert the email of the user into a dictionary called session, with the key “email”

session["email"] = email

1. create logout capability.
2. Put a link in for log out in base html for all pages to see/have

@app.route("/logout")

def logout():

del session["email"]

return redirect("/")

Here we simply have a function that removes the user’s email from the session dictionary. We check for this entry for every request, so if it is not there we know the user has logged out

In our base.html form we put a link so that the user can logout from any page visited:

<div>

<a href = "/logout">log out</a>

</div>

1. Check if the user is logged in on a request by request basis.
   * If they are logged in, good. If not, make them log in

@app.before\_request

def require\_login():

allowed\_routes = ["login", "register"]

if request.endpoint not in allowed\_routes and "email" not in session:

return redirect("/login")

a before\_request is executed first, regardless of the rtequest from the user. So we use it to check to see if the user has logged in.

if request.endpoint not in allowed\_routes and "email" not in session:

If the user has not logged in we send the user back to the login page.

return redirect("/login")

We can specify exceptions to the before\_request rule/function. As an example, we want the user to be able to access the login and register pages without being logged in. So we create a LIST, here called allowed\_routes, of those routes that are excluded from this before\_request rule/function. Then we check the route destination (request.endpoint) of the request. If it is not in the allowed\_routes LIST and the email is not in the session dictionary (meaning the user has not logged in), we send the user to the login page.

allowed\_routes = ["login", "register"]

if request.endpoint not in allowed\_routes and "email" not in session:

return redirect("/login")

1. Set the secret key for the session

In our python code we set a key for the session dictionary:

app.secret\_key = "y337kGcys&zP3B" where "y337kGcys&zP3B" is a key we provide/make up ourselves.

Here are the complete files:

Python.main

from flask import Flask, request, redirect, render\_template, session

from flask\_sqlalchemy import SQLAlchemy

app = Flask(\_\_name\_\_)

app.config["DEBUG"] = True

app.config["SQLALCHEMY\_DATABASE\_URI"] = "mysql+pymysql://get-it-done:beproductive@localhost:8889/get-it-done"

app.config["SQLALCHEMY\_ECHO"] = True

db = SQLAlchemy(app)

app.secret\_key = "y337kGcys&zP3B"

class Task(db.Model):

id = db.Column(db.Integer, primary\_key = True)

name = db.Column(db.String(120))

completed = db.Column(db.Boolean, default = False)

def \_\_init\_\_(self, name):

self.name = name

self.completed = False

class User(db.Model):

id = db.Column(db.Integer, primary\_key = True)

email = db.Column(db.String(120), unique = True)

password = db.Column(db.String(120))

def \_\_init\_\_(self, email, password):

self.email = email

self.password = password

@app.before\_request

def require\_login():

allowed\_routes = ["login", "register"]

if request.endpoint not in allowed\_routes and "email" not in session:

return redirect("/login")

@app.route("/login",methods = ["POST", "GET"])

def login():

if request.method == "POST":

email = request.form["email"]

password = request.form["password"]

user=User.query.filter\_by(email=email).first()

if user and user.password == password:

session["email"] = email

return redirect("/")

else:

#TODO - explain why login failed

return "<h1>Error!</h1>"

return render\_template("login.html")

@app.route("/logout")

def logout():

del session["email"]

return redirect("/")

@app.route("/register", methods = ["POST", "GET"])

def register():

if request.method == "POST":

email = request.form["email"]

password = request.form["password"]

verify = request.form["verify"]

#TODO - validte user data (User-signup project has this code)

existing\_user=User.query.filter\_by(email=email).first()

if not existing\_user:

new\_user=User(email,password)

db.session.add(new\_user)

db.session.commit()

session["email"] = email

return redirect("/")

else:

#TODO use better response messaging

return "<h1>Duplicate user</h1>"

return render\_template("register.html")

@app.route("/delete-task", methods = ["POST"])

def delete\_task():

task\_id = int( request.form["task-id"] )

task = Task.query.get(task\_id)

task.completed = True

db.session.add(task)

db.session.commit()

return redirect("/")

@app.route("/", methods = ["POST", "GET"])

def index():

if request.method == "POST":

task\_name = request.form["task"]

new\_task = Task(task\_name)

db.session.add(new\_task)

db.session.commit()

tasks = Task.query.filter\_by(completed = False).all()

completed\_tasks = Task.query.filter\_by(completed = True).all()

return render\_template("todos.html", title = "Get It Done!",

tasks = tasks, completed\_tasks = completed\_tasks)

if \_\_name\_\_ == "\_\_main\_\_":

app.run()

base.html

<!doctype html>

<html>

<head>

<title>{{title}}</title>

</head>

<body>

<div>

<a href = "/logout">log out</a>

</div>

{% block content %}

{% endblock %}

</body>

</html>

login.html

{% extends "base.html" %}

{% block content %}

<h1>Log in to Get It Done!</h1>

<p>If you don't have an account, you may <a href="/register">register</a> for one</p>

<form action="/login" method="post">

<table>

<tr>

<td><label for="email">Email</label></td>

<td>

<input name="email" type="email" value="{{ email }}">

</td>

</tr>

<tr>

<td><label for="password">Password</label></td>

<td>

<input name="password" type="password">

</td>

</tr>

</table>

<input type="submit" value="Login">

</form>

{% endblock %}

register.html

{% extends "base.html" %}

{% block content %}

<h1>Register to Get It Done!</h1>

<form action="/register" method="post">

<table>

<tr>

<td><label for="email">Email</label></td>

<td>

<input name="email" type="email" value="{{ email }}">

</td>

</tr>

<tr>

<td><label for="password">Password</label></td>

<td>

<input name="password" type="password">

</td>

</tr>

<tr>

<td><label for="verify">Verify Password</label></td>

<td>

<input name="verify" type="password">

</td>

</tr>

</table>

<input type="submit" value="Register">

</form>

{% endblock %}

todos.html

{% extends "base.html" %}

{% block content %}

<h1>Get It Done!</h1>

<form method='post'>

<label>

New Task:

<input type="text" name='task' />

</label>

<input type="submit" value='Add Task' />

</form>

<hr />

{% if tasks|length == 0 %}

<p>No tasks yet</p>

{% else %}

<ul>

{% for task in tasks %}

<li>{{task.name}}

<form method = "POST" action = "/delete-task" style="display:inline-block">

<input type = "hidden" name="task-id" value="{{task.id}}"/>

<input type="submit" value="Done!"/>

</form>

</li>

{% endfor %}

</ul>

{% endif %}

<hr />

<h2>Completed</h2>

<ul>

{% for task in completed\_tasks %}

<li>{{task.name}}</li>

{% endfor %}

</ul>

{% endblock %}

**ONE TO MANY RLATIONSHIPS IN A DATA BASE**

1. add owner to class Tasks in our project “get-it-done” and establish relationship between tasks and owners

class Task(db.Model):

id = db.Column(db.Integer, primary\_key = True)

name = db.Column(db.String(120))

completed = db.Column(db.Boolean, default = False)

owner\_id = db.Column(db.Integer, db.ForeignKey("user.id"))

def \_\_init\_\_(self, name, owner):

self.name = name

self.completed = False

self.owner = owner

so, first we added owner and established its relationship to the user id. is from the login handler. We established the foreign key relationship to user.id – note that user is from logon handler (that’s what we called the user) and we assign the relationship owner\_id for tasks to user.id field in users.

owner\_id = db.Column(db.Integer, db.ForeignKey("user.id"))

Finally, we modify the constructor to reflect the owner of a task

def \_\_init\_\_(self, name, owner):

self.owner = owner

1. update data base schema to reflect the changes in item 1

we need to import our functions and classes from our project

>>>from main import db, User, Task

>>>

from our python shell we execute these commands. First we drop or delete all the tables in our data base since we are changing the schema.

>>>db.drop\_all()

lots of output here – make sure there are no errors reported

Now we need to recreate the data base tables

>>> db.create\_all()

lots of output here – make sure there are no errors reported

CREATE TABLE user (

id INTEGER NOT NULL AUTO\_INCREMENT,

email VARCHAR(120),

password VARCHAR(120),

PRIMARY KEY (id),

UNIQUE (email)

)

CREATE TABLE task (

id INTEGER NOT NULL AUTO\_INCREMENT,

name VARCHAR(120),

completed BOOL,

owner\_id INTEGER,

PRIMARY KEY (id),

CHECK (completed IN (0, 1)),

FOREIGN KEY(owner\_id) REFERENCES user (id)

)

1. add some data to our data base since we lost all data when we updated the data base schema
   1. user
   2. tasks

so we’ll add a user and a task

>>> user = User('chris@launchcode.org', 'cheese')

>>> task = Task('finish lesson 10', user)>>> db.session.add(user)

>>> db.session.add(task)

>>> db.session.add(user)

>>> db.session.commit()

1. Update python code to deal with new owner to task relationship

In our project it’s the index function we have to modify since the other functions (login, register) don’t deal with the user to task relationship and it’s where we deal with adding a new task so it has to be associate3d with its user

@app.route("/", methods = ["POST", "GET"])

def index():

owner = User.query.filter\_by(email = session["email"]).first()

if request.method == "POST":

task\_name = request.form["task"]

owner = User.query.filter\_by(email = session["email"]).first()

new\_task = Task(task\_name, owner)

db.session.add(new\_task)

db.session.commit()

tasks = Task.query.filter\_by(completed = False,owner=owner).all()

completed\_tasks = Task.query.filter\_by(completed = True, owner=owner).all()

return render\_template("todos.html", title = "Get It Done!",

tasks = tasks, completed\_tasks = completed\_tasks)

first we get the user’s email – we will use this as the owner of the task we get it from the session dictionary,

owner = User.query.filter\_by(email = session["email"]).first()

then we assign that task to the current user

new\_task = Task(task\_name, owner)

1. Update python code to filter on owner when rendering tasks

now we have to display only those task belonging to the user – both completed and uncompleted

tasks = Task.query.filter\_by(completed = False,owner=owner).all()

completed\_tasks = Task.query.filter\_by(completed = True, owner=owner).all()

return render\_template("todos.html", title = "Get It Done!",

tasks = tasks, completed\_tasks = completed\_tasks)

Here are the complete files:

Python.main

from flask import Flask, request, redirect, render\_template, session, flash

from flask\_sqlalchemy import SQLAlchemy

app = Flask(\_\_name\_\_)

app.config["DEBUG"] = True

app.config["SQLALCHEMY\_DATABASE\_URI"] = "mysql+pymysql://get-it-done:beproductive@localhost:8889/get-it-done"

app.config["SQLALCHEMY\_ECHO"] = True

db = SQLAlchemy(app)

app.secret\_key = "y337kGcys&zP3B"

class Task(db.Model):

id = db.Column(db.Integer, primary\_key = True)

name = db.Column(db.String(120))

completed = db.Column(db.Boolean, default = False)

owner\_id = db.Column(db.Integer, db.ForeignKey("user.id"))

def \_\_init\_\_(self, name, owner):

self.name = name

self.completed = False

self.owner = owner

class User(db.Model):

id = db.Column(db.Integer, primary\_key = True)

email = db.Column(db.String(120), unique = True)

password = db.Column(db.String(120))

tasks = db.relationship("Task", backref="owner")

def \_\_init\_\_(self, email, password):

self.email = email

self.password = password

@app.before\_request

def require\_login():

allowed\_routes = ["login", "register"]

if request.endpoint not in allowed\_routes and "email" not in session:

return redirect("/login")

@app.route("/login",methods = ["POST", "GET"])

def login():

if request.method == "POST":

email = request.form["email"]

password = request.form["password"]

user=User.query.filter\_by(email=email).first()

if user and user.password == password:

session["email"] = email

flash("logged in")

print(session)

return redirect("/")

else:

flash("User password incorrect, or user does not exist", "error")

return render\_template("login.html")

@app.route("/logout")

def logout():

del session["email"]

print("WE ARE HERE IN LOGOUT")

print("we deleted email from session", session)

return redirect("/")

@app.route("/register", methods = ["POST", "GET"])

def register():

if request.method == "POST":

email = request.form["email"]

password = request.form["password"]

verify = request.form["verify"]

#TODO - validte user data (User-signup project has this code)

existing\_user=User.query.filter\_by(email=email).first()

if not existing\_user:

new\_user=User(email,password)

db.session.add(new\_user)

db.session.commit()

session["email"] = email

return redirect("/")

else:

#TODO use better response messaging

return "<h1>Duplicate user</h1>"

return render\_template("register.html")

@app.route("/delete-task", methods = ["POST"])

def delete\_task():

task\_id = int( request.form["task-id"] )

task = Task.query.get(task\_id)

task.completed = True

db.session.add(task)

db.session.commit()

return redirect("/")

@app.route("/", methods = ["POST", "GET"])

def index():

owner = User.query.filter\_by(email = session["email"]).first()

if request.method == "POST":

task\_name = request.form["task"]

owner = User.query.filter\_by(email = session["email"]).first()

new\_task = Task(task\_name, owner)

db.session.add(new\_task)

db.session.commit()

tasks = Task.query.filter\_by(completed = False,owner=owner).all()

completed\_tasks = Task.query.filter\_by(completed = True, owner=owner).all()

return render\_template("todos.html", title = "Get It Done!",

tasks = tasks, completed\_tasks = completed\_tasks)

if \_\_name\_\_ == "\_\_main\_\_":

app.run()

base.html

<!doctype html>

<html>

<head>

<title>{{title}}</title>

</head>

<body>

<div>

<a href = "/logout">log out</a>

</div>

{% block content %}

{% endblock %}

</body>

</html>

login.html

{% extends "base.html" %}

{% block content %}

<h1>Log in to Get It Done!</h1>

<p>If you don't have an account, you may <a href="/register">register</a> for one</p>

<form action="/login" method="post">

<table>

<tr>

<td><label for="email">Email</label></td>

<td>

<input name="email" type="email" value="{{ email }}">

</td>

</tr>

<tr>

<td><label for="password">Password</label></td>

<td>

<input name="password" type="password">

</td>

</tr>

</table>

<input type="submit" value="Login">

</form>

{% endblock %}

register.html

{% extends "base.html" %}

{% block content %}

<h1>Register to Get It Done!</h1>

<form action="/register" method="post">

<table>

<tr>

<td><label for="email">Email</label></td>

<td>

<input name="email" type="email" value="{{ email }}">

</td>

</tr>

<tr>

<td><label for="password">Password</label></td>

<td>

<input name="password" type="password">

</td>

</tr>

<tr>

<td><label for="verify">Verify Password</label></td>

<td>

<input name="verify" type="password">

</td>

</tr>

</table>

<input type="submit" value="Register">

</form>

{% endblock %}

todos.html

{% extends "base.html" %}

{% block content %}

<h1>Get It Done!</h1>

<form method='post'>

<label>

New Task:

<input type="text" name='task' />

</label>

<input type="submit" value='Add Task' />

</form>

<hr />

{% if tasks|length == 0 %}

<p>No tasks yet</p>

{% else %}

<ul>

{% for task in tasks %}

<li>{{task.name}}

<form method = "POST" action = "/delete-task" style="display:inline-block">

<input type = "hidden" name="task-id" value="{{task.id}}"/>

<input type="submit" value="Done!"/>

</form>

</li>

{% endfor %}

</ul>

{% endif %}

<hr />

<h2>Completed</h2>

<ul>

{% for task in completed\_tasks %}

<li>{{task.name}}</li>

{% endfor %}

</ul>

{% endblock %}