Filename	Location	Necessary for full stack app?	Contents	Description

run_sql.py	db/	yes	`import psycopg2`
			`import psycopg2.extras as ext`
			
			`def run_sql(sql, values = None):`
			`  conn = None`
			`  results = []`
			`  database_name = 'database_name'`
			`  try:`
			`  conn = psycopg2.connect(f"dbname = '{database_name}'")`
			`  cur = conn.cursor(cursor_factory=ext.DictCursor)`
			`  cur.execute(sql, values)`
			`  conn.commit()`
			`  results = cur.fetchall()`
			`  cur.close()`
			`  except (Exception, psycopg2.DatabaseError) as error:`
			`  print(error)`
			`  finally:`
			`  if conn is not None:`
			`  conn.close()`
			`  return results`
				this is what has the connection from your repositories to your database back end. Shouldn't need to change much apart from ensuring you have the database named.
				
				The **database_name** is the one you created when you ran command `createdb <database_name>`

<dbname>.sql
 	
(Doesn't need to match the db name, but it's best practice)
	db/	for set up - not for running the application
		
		Use by running `psql -d <dbname> -f <filename.sql>`
			`DROP TABLE IF EXISTS <table>;`
			
			`CREATE TABLE <table> (`
			
			`);`
			
			`INSERT INTO <table> (col1, col2) VALUES ('value1', 'value2')`
				this is how to create/set up/seed your tables - useful for running tests in your console.py.
				
				Use single quotes for the string data in any insert statements.

console.py	Top Level	no		allows you to test the functionality of repos and models and that the tables are set up correctly
.gitignore	Top Level	If it's a git rep.	`__pycache__`	Contains what you want git to ignore tracking
<image>.png	/static/images	Yes		What? No pictures?
<class>.py	Models/	Yes		The definition of your class objects. You may also have methods here that your repo and controller can use (ie decrease_wallet(amt) or something) to ensure your classes are encapsulated.

Filename	Location	Necessary for full stack app?	Contents	Description

app.py	Top Level	yes	`from flask import Flask, render_template`
			`from controllers.<class>_controller import <class>_blueprint`
			
			`app = Flask(__name__)`
			
			`app.register_blueprint(<class>_blueprint)`
			
			`@app.route('/')`
			`def home():`
			`  return render_template('index.html')`
			`if __name__ == "__main__":`
			`app.run(debug=True)`
				Contains the blueprints (app.register_blueprint(task_blueprint) and the homepage route for the whole site.
				
				Should contain the route for the homepage as an app.route(), but should be the only app.route, as the rest in the controllers will be blueprint.route()

<class>_controller.py	controllers/	Yes. Should have 1 per concern - i.e. books_controller.py and authors_controller.py	`from flask import render_template, redirect, request`
			`from repositories import <class>_repository as <class>_repo`
			`from models.<class> import <Class>`
			
			`from flask import Blueprint`
			`<class>_blueprint = Blueprint("<class>",__name__)`
			
			`@<class>_blueprint.route('/<class>')`
				contains the actual blueprint and the routes for the <class> specific artifacts, as well as the repos.
				
				Connects the repositories to the routes.

<class>_repository.py	repositories/	Yes. Should have one per class or table.	`from db.run_sql import run_sql`
			`from models.<class> import <Class>`
			`import repositories.<otherclass>_repository as <otherclass>_repo`
			
			`def select_all(): ....` etc
				Connects the front end to the database - this is where we create the functions that run SQL. Will need the models, etc. These functions are what we use to retrieve data or to persist it (ie insert or update) on the database tables.

.flaskenv	Top Level	Yes	`FLASK_APP=app.py`
			`FLASK_RUN_PORT=4999`
			`FLASK_DEBUG=1`
			`TEMPLATES_AUTO_RELOAD=1`
				Contains the flask environment variables for running the application. Should contain

run_tests.py	Top Level	Best practice	`import unittest`
			`from models.<class>_test import <TestClass>`
			
			`if __name__ == "__main__":`
			`unittest.main()`
				To allow you to create model test files in order to unit test your classes. If you have no methods you can get away without this. Same as testing just straight python. Run by:
				`python3 run_tests.py`

<class>_test.py	/tests	Best practice	`import unittest`
			`from models.<class> import <Class>`
			
			`class Test<Class>(unittest.TestCase):`
			`  def setUp(self):`
				Run by the run_tests.py if you have methods in your classes. Same as for just testing pure python

Filename	Location	Necessary for full stack app?	Contents	Description
style.css	/static	Yes		You want flair, no? Remember to include other versions for other sizes - mobile.css for instance
Index.html	/templates	Yes	{% extends 'base.html' %} {% block content%} In here is what you want the central part of your splash page {% endblock %}	This is the front page - the landing page of your website. It should extend base.html and be entirely block content
Base.html	/templates	Yes	<!--base.html --> <!DOCTYPE html> <html lang="en"> <head> <meta charset="UTF-8"> <meta http-equiv="X-UA-Compatible" content="IE=edge"> <meta name="viewport" content="width=device-width, initial-scale=1.0"> <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/font-awesome/4.7.0/css/font-awesome.min.css" crossorigin="anonymous"> <!-- NEW --> <link rel="stylesheet" href="{{ url_for('static', filename='style.css') }}"> <title>Homepage</title> </head> <body> <header> Website Header Text <nav> Home - other menu item - other menu item </nav> </header> <hr /> <article> {% block content %} {% endblock %} </article> <hr /> </body> <footer> Footer Content </footer> </html>	This is the consistent look and feel of your website - the header and footer with the nav for instance. Has a block content section that all the other pages consist of

Filename	Location	Necessary for full stack app?	Contents	Description
Index.html	/templates/<class>	Yes	{% extends 'base.html' %} {% block content%} In here is what you want the central part of your class home page - ie. show all the books, for example. {% endblock %}	This is the class specific stuff. Ie the main books page. There will be others, based on what you're allowing the user to do, but keep them neatly organised.