Check results
=============

Code
=============
```
import os
from flask import (
    Flask, flash, render_template,
    redirect, request, session, url_for)
from flask_pymongo import PyMongo
from bson.objectid import ObjectId
from werkzeug.security import generate_password_hash,
check_password_hash
if os.path.exists("env.py"):
    import env

app = Flask(__name__)

app.config["MONGO_DBNAME"] = os.environ.get("MONGO_DBNAME")
app.config["MONGO_URI"] = os.environ.get("MONGO_URI")
app.secret_key = os.environ.get("SECRET_KEY")

mongo = PyMongo(app)


@app.route("/")
@app.route("/get_recipes")
def get_recipes():
    categories =
list(mongo.db.categories.find().sort("recipe_category", 1))
    levels = list(mongo.db.level_of_difficulty.find())
    recipes = list(mongo.db.recipes.find())
    return render_template("recipes.html", recipes=recipes,
                           categories=categories, levels=levels)


@app.route("/search", methods=["GET", "POST"])
def search():
    query = request.form.get("query")
    categories =
list(mongo.db.categories.find().sort("recipe_category", 1))
    levels = list(mongo.db.level_of_difficulty.find())
    recipes = list(mongo.db.recipes.find({"$text": {"$search":
query}}))
    return render_template("recipes.html", recipes=recipes,
                           categories=categories, levels=levels,
                           query=query)


@app.route("/register", methods=["GET", "POST"])
def register():
    if request.method == "POST":
        # Check if username already exists in the Database
        existing_user = mongo.db.users.find_one(
            {"username": request.form.get("username").lower()})

        if existing_user:
            flash("Username already exists")
            return redirect(url_for("register"))

        register = {
            "first_name": request.form.get("first_name"),
            "last_name": request.form.get("last_name"),
            "username": request.form.get("username").lower(),
            "email": request.form.get("email").lower(),
            "password":
generate_password_hash(request.form.get("password"))
        }
        mongo.db.users.insert_one(register)

        # Put the new user into 'session' cookie
        session["user"] = request.form.get("username").lower()
        flash("Registration successful!")
        return redirect(url_for("profile",
username=session["user"]))

    return render_template("register.html")


@app.route("/login", methods=["GET", "POST"])
def login():
    if request.method == "POST":
        # Check if username exists in Database
        existing_user = mongo.db.users.find_one(
            {"username": request.form.get("username")})

        if existing_user:
            # Ensure hashed password matches user input
            if check_password_hash(
              existing_user["password"],
request.form.get("password")):
                session["user"] =
request.form.get("username").lower()
                flash("Welcome
{}" .format(request.form.get("username")))
                return redirect(url_for("profile",
username=session["user"]))
            else:
                # Invalid password match
                flash("Incorrect Username and/or Password")
                return redirect(url_for("login"))

        else:
            # Username doesn't exist
            flash("Incorrect Username and/or Password")
            return redirect(url_for("login"))
    return render_template("login.html")


@app.route("/profile", methods=["GET", "POST"])
def profile():
    if request.method == "POST":
        user = mongo.db.users.find_one({"username":
session["user"]})
        recipes = mongo.db.recipes.find()
        return render_template(
            "profile.html", user=user,
            recipes=recipes)

    user = mongo.db.users.find_one({"username": session["user"]})

    # Get username for displaying user's recipes and favourites
    username = mongo.db.users.find_one(
        {"username": session["user"]})["username"]
    recipes_created_by = list(mongo.db.recipes.find({"created_by":
username}))
    favourites_of = list(mongo.db.recipes.find({"favourite_of":
username}))

    recipes = list(mongo.db.recipes.find())

    if session["user"]:
        return render_template(
            "profile.html", user=user,
            recipes=recipes, username=username,
            recipes_created_by=recipes_created_by,
            favourites_of=favourites_of)

    return render_template("profile.html")


@app.route("/edit_profile", methods=["GET", "POST"])
def edit_profile():
    if request.method == "POST":
        data = {"$set": {
            "image_url": request.form.get("profile_image_url"),
            "first_name": request.form.get("first_name"),
            "last_name": request.form.get("last_name"),
            "email": request.form.get("email")}
            }

        mongo.db.users.update_one({"username": session["user"]},
data)

        user = mongo.db.users.find_one({"username":
session["user"]})
        return redirect(url_for('profile',
username=session["user"]))
```

```
    user = mongo.db.users.find_one({"username": session["user"]})

    # Get username for displaying user's recipes
    username = mongo.db.users.find_one(
        {"username": session["user"]})["username"]
    recipes_created_by = list(mongo.db.recipes.find({"created_by":
username}))

    recipes = mongo.db.recipes.find()

    # If session cookie exists
    if session["user"]:
        return render_template(
            "edit_profile.html", user=user,
            recipes=recipes, username=username,
            recipes_created_by=recipes_created_by)

    return render_template("edit_profile.html")


@app.route("/logout")
def logout():
    # Remove user from session cookies
    flash("You have been logged out")
    session.pop("user")
    return redirect(url_for("login"))


@app.route("/add_recipe", methods=["GET", "POST"])
def add_recipe():
    if request.method == "POST":
        user = mongo.db.users.find_one(
            {"username": session["user"]})["username"]
        favourite_of = [user] if request.form.get("favourite_of")
else [""]

        category = {
            "recipe_category": request.form.get("recipe_category")
        }

        # Sends EACH cat in category to categories collection
        categories = category["recipe_category"].split("\r\n")

        for cat in categories:
            new_cats = {"recipe_category": cat}
            categories_collection = mongo.db.categories
            # Check if category exists in Database
            existing_category = mongo.db.categories.find_one(
                {"recipe_category": cat})
            if not existing_category:
                categories_collection.insert_one(new_cats)

        recipe = {
            "recipe_name": request.form.get("recipe_name"),
            "recipe_ingredients":
request.form.get("recipe_ingredients"),
            "recipe_preparation":
request.form.get("recipe_preparation"),
            "recipe_notes": request.form.get("recipe_notes"),
            "recipe_prep_time":
request.form.get("recipe_prep_time"),
            "recipe_cooking_time":
request.form.get("recipe_cooking_time"),
            "recipe_total_time":
request.form.get("recipe_total_time"),
            "recipe_description":
request.form.get("recipe_description"),
            "recipe_category": categories,
            "recipe_level_of_difficulty": request.form.get(
                "recipe_level_of_difficulty"),
            "recipe_servings": request.form.get("recipe_servings"),
            "image_url": request.form.get("recipe_image_url"),
            "recipe_source": request.form.get("recipe_source"),
            "created_by": session["user"],
            "favourite_of": favourite_of
        }
        mongo.db.recipes.insert(recipe)
        flash("Recipe added to the Recipeas and Greens community!")
        return redirect(url_for("get_recipes"))

    level_of_difficulty = mongo.db.level_of_difficulty.find()
    servings = mongo.db.servings.find()
    return render_template(
        "add_recipe.html", level_of_difficulty=level_of_difficulty,
        servings=servings)


@app.route("/recipe/<recipe_id>")
def recipe(recipe_id):
    categories =
list(mongo.db.categories.find().sort("recipe_category", 1))
    levels = list(mongo.db.level_of_difficulty.find())
    recipe = mongo.db.recipes.find_one({"_id": ObjectId(recipe_id)})
    return render_template("recipe.html", recipe=recipe,
categories=categories,
                           levels=levels)


@app.route("/edit_recipe/<recipe_id>", methods=["GET", "POST"])
def edit_recipe(recipe_id):
    if request.method == "POST":
        user = mongo.db.users.find_one(
            {"username": session["user"]})["username"]

        category = {
            "recipe_category": request.form.get("recipe_category")
        }
        # Sends EACH cat in category to categories collection
        categories = category["recipe_category"].split("\r\n")

        for cat in categories:
            new_cats = {"recipe_category": cat}
            categories_collection = mongo.db.categories
            # Check if category exists in Database
            existing_category = mongo.db.categories.find_one(
                {"recipe_category": cat})
            if not existing_category:
                categories_collection.insert_one(new_cats)

        submit = {
            "recipe_name": request.form.get("recipe_name"),
            "recipe_ingredients":
request.form.get("recipe_ingredients"),
            "recipe_preparation":
request.form.get("recipe_preparation"),
            "recipe_notes": request.form.get("recipe_notes"),
            "recipe_prep_time":
request.form.get("recipe_prep_time"),
            "recipe_cooking_time":
request.form.get("recipe_cooking_time"),
            "recipe_total_time":
request.form.get("recipe_total_time"),
            "recipe_description":
request.form.get("recipe_description"),
            "recipe_category": categories,
            "recipe_level_of_difficulty": request.form.get(
                "recipe_level_of_difficulty"),
            "recipe_servings": request.form.get("recipe_servings"),
            "image_url": request.form.get("recipe_image_url"),
            "recipe_source": request.form.get("recipe_source"),
            "created_by": session["user"],
        }

        mongo.db.recipes.update({"_id": ObjectId(recipe_id)},
submit)
        flash("Recipe successfully edited")
        return redirect(url_for('get_recipes'))

    recipe = mongo.db.recipes.find_one({"_id": ObjectId(recipe_id)})

    """Redirect to home page if user who did not create
    recipe tries to force edit recipe """
    recipe_created_by = mongo.db.recipes.find_one(
        {"_id": ObjectId(recipe_id)})["created_by"]
    username = mongo.db.users.find_one(
        {"username": session["user"]})["username"]
    if username not in recipe_created_by:
        return redirect("/")

    level_of_difficulty = mongo.db.level_of_difficulty.find()
    servings = mongo.db.servings.find()
    return render_template(
        "edit_recipe.html", recipe=recipe,
        level_of_difficulty=level_of_difficulty,
        servings=servings)


@app.route("/delete_recipe/<recipe_id>")
def delete_recipe(recipe_id):
    mongo.db.recipes.remove({"_id": ObjectId(recipe_id)})
    flash("Recipe Successfully Deleted")
    return redirect(url_for("get_recipes"))


@app.route("/add_to_favourites/<recipe_id>")
def add_to_favourites(recipe_id):
    username = mongo.db.users.find_one(
        {"username": session["user"]})["username"]

    mongo.db.recipes.update_one(
        {"_id": ObjectId(recipe_id)},
        {"$addToSet": {"favourite_of": username}})
    flash("Recipea added to your list of favourites!")
    return redirect(url_for("get_recipes"))


@app.route("/remove_from_favourites/<recipe_id>")
def remove_from_favourites(recipe_id):
    username = mongo.db.users.find_one(
        {"username": session["user"]})["username"]

    mongo.db.recipes.update_one(
        {"_id": ObjectId(recipe_id)},
        {"$pull": {"favourite_of": username}})
    flash("Recipea removed from your list of favourites!")
    return redirect(url_for("get_recipes"))


@app.route("/remove_all_from_favourites_and_delete_account")
def remove_all_from_favourites_and_delete_account():
    # This deletes all recipes favourited_by the user
    username = mongo.db.users.find_one(
        {"username": session["user"]})["username"]

    recipes = mongo.db.recipes
    existing_favourite_of = {"favourite_of": username}
    remove_favourite_of = {"$pull": {"favourite_of": username}}
    recipes.update_many(existing_favourite_of, remove_favourite_of)

    """This re-credits all recipes created by the user as
    "created by Former Member" """
    existing_created_by = {"created_by": session["user"]}
    recredited_created_by = {"$set": {"created_by": "Former
Member"}}
    recipes.update_many(existing_created_by, recredited_created_by)
    # This deletes the user account
    user = mongo.db.users.find_one(
        {"username": session["user"]})["_id"]

    mongo.db.users.delete_one({"_id": ObjectId(user)})

    # This removes the cookie
    session.pop("user")

    return redirect(url_for("get_recipes"))


@app.route("/
remove_all_from_favourites_and_delete_recipes_and_delete_account")
def
remove_all_from_favourites_and_delete_recipes_and_delete_account():
    # This deletes all recipes favourited_by the user
    username = mongo.db.users.find_one(
        {"username": session["user"]})["username"]

    recipes = mongo.db.recipes
    existing_favourite_of = {"favourite_of": username}
    remove_favourite_of = {"$pull": {"favourite_of": username}}
    recipes.update_many(existing_favourite_of, remove_favourite_of)

    # This deletes all recipes created by the user
    created_by = {"created_by": session["user"]}
    recipes.delete_many(created_by)

    # This deletes the user account
    user = mongo.db.users.find_one(
        {"username": session["user"]})["_id"]

    mongo.db.users.delete_one({"_id": ObjectId(user)})

    # This removes the cookie
    session.pop("user")

    return redirect(url_for("get_recipes"))


@app.route("/get_categories")
def get_categories():
    categories = mongo.db.categories.find().sort("recipe_category",
1)
    return render_template("categories.html", categories=categories)


@app.route("/category/<category_id>")
def category(category_id):
    # This gets the category in the categories collection by its ID
    category = mongo.db.categories.find_one(
        {"_id": ObjectId(category_id)})["recipe_category"]
    # This gets the levels of difficulty for the search by level of
difficulty
    levels = mongo.db.level_of_difficulty.find()
    """This gets the categories for the search by category and
    sorts them alphabetically button """
    categories =
list(mongo.db.categories.find().sort("recipe_category", 1))
    recipes = list(mongo.db.recipes.find({"recipe_category":
category}))

    return render_template("category.html", category=category,
                           levels=levels, recipes=recipes,
                           categories=categories)


@app.route("/edit_category/<category_id>", methods=["GET", "POST"])
def edit_category(category_id):
    if request.method == "POST":
        submit = {
            "recipe_category": request.form.get("recipe_category")
        }
        mongo.db.categories.update({"_id": ObjectId(category_id)},
submit)
        flash("Category Successfully Updated")
        return redirect(url_for("get_categories"))

    category = mongo.db.categories.find_one({"_id":
ObjectId(category_id)})
    return render_template("edit_category.html", category=category)


@app.route("/delete_category/<category_id>")
def delete_category(category_id):
    mongo.db.categories.remove({"_id": ObjectId(category_id)})
    flash("Category Successfully Deleted")
    return redirect(url_for("get_categories"))


@app.route("/get_difficulty_levels")
def get_difficulty_levels():
    levels = list(mongo.db.level_of_difficulty.find())
    return render_template("difficulty_levels.html", levels=levels)


@app.route("/level/<level_id>")
def level(level_id):

    level = mongo.db.level_of_difficulty.find_one(
        {"_id": ObjectId(level_id)})["recipe_level_of_difficulty"]

    levels = mongo.db.level_of_difficulty.find()

    categories =
list(mongo.db.categories.find().sort("recipe_category", 1))
```

```
    recipes = list(mongo.db.recipes.find(
        {"recipe_level_of_difficulty": level}))

    return render_template("level.html", level=level,
                           recipes=recipes, levels=levels,
                           categories=categories)


@app.route("/edit_levels/<level_id>", methods=["GET", "POST"])
def edit_levels(level_id):
    if request.method == "POST":
        submit = {
            "recipe_level_of_difficulty": request.form.get(
                "recipe_level_of_difficulty")
        }
        mongo.db.level_of_difficulty.update(
            {"_id": ObjectId(level_id)}, submit)
        flash("Successfully Updated")
        return redirect(url_for("get_difficulty_levels"))

    level = mongo.db.level_of_difficulty.find_one({"_id":
ObjectId(level_id)})
    return render_template("edit_levels.html", level=level)


@app.route("/delete_level/<level_id>")
def delete_level(level_id):
    mongo.db.level_of_difficulty.remove({"_id": ObjectId(level_id)})
    flash("Level Successfully Deleted")
    return redirect(url_for("get_difficulty_levels"))


@app.errorhandler(404,)
def page_not_found(e):
    categories =
list(mongo.db.categories.find().sort("recipe_category", 1))
    levels = list(mongo.db.level_of_difficulty.find())
    return render_template("404.html",
                           categories=categories,
                           levels=levels), 404


@app.errorhandler(410)
def page_not_there(e):
    categories =
list(mongo.db.categories.find().sort("recipe_category", 1))
    levels = list(mongo.db.level_of_difficulty.find())
    return render_template("404.html", categories=categories,
                           levels=levels), 410


@app.errorhandler(500)
def special_exception_handler(error):
    categories =
list(mongo.db.categories.find().sort("recipe_category", 1))
    levels = list(mongo.db.level_of_difficulty.find())
    return render_template("500.html", categories=categories,
                           levels=levels), 500


if __name__ == "__main__":
    app.run(host=os.environ.get("IP"),
            port=int(os.environ.get("PORT")),
            debug=False)
```