YUMMLY 3.0 - README

This paper contains an executive README of the application **YUMMLY 3.0**, created by Group 1 of the course 'Applications in Object-Oriented Programming and Databases'.

Prerequisites

- **Python version**: python3
- **Python Packages**: check out the file `prerequisits.txt`!

Workings of Yummly 3.0

This section mainly serves as explanation of the program itself. For easier understanding (if needed) it is recommended to use the application overview diagram as reference. The Yummly 3.0 program is executed in a python environment, with which the user interacts via `input()` functions.

The program comes together in the `main.py` file which taps into the other necessary functions. Upon executing the latter file the user is required to sign in or sign up. Following this choice a new user is either saved into the database or the username and password are compared to a hashed record stored in the database. Hereby the `main.py` script directly communicates with the data access object [DAO].

Following this first interaction, the user is offered a choice of options which shall be discussed hereafter. The concept of each request is always a similar one. Thus the user chooses an option, if the latter requires further input the main file requests this. The obtained information are then used by one of the API controllers which requests the API and returns the information which is saved as object using one of the blueprints provided by the classes in the Models file. In a next optional step, the data is saved to the users profile in the MySQL database through the functions provided in one of the helper files which are accessed through the `master_helpers.py` file.

```
Please enter the number of one of the following options:

    1.  Search a new recipe using pictures
    2.  Search recipes by name
    3.  Get all your recipes
    'info' to get information on all options
    'exit' to end the program
```

1. Search a new recipe using pictures

This option uses pictures stored locally in a folder. The path is provided by the user where after the images are encoded and through use of the Google Vision API the ingredients shown on the pictures are recognized and used as input for the Spoonacular API which returns a list of possible recipes to the user. The user can select to see explicit instructions about the recipe and thereafter decide to store it in the database.

2. Search recipes by name

Here the user may search a recipe through an input string, which will be sent to the Spoonacular API and returns once again a list of recipes. Once again the user can select to see more details and save the inspected recipe.

3. Get all your recipes

This functionality taps into the database to obtain all recipes saved to the users profile. The recipes can again be inspected.

'info' to get information on all options

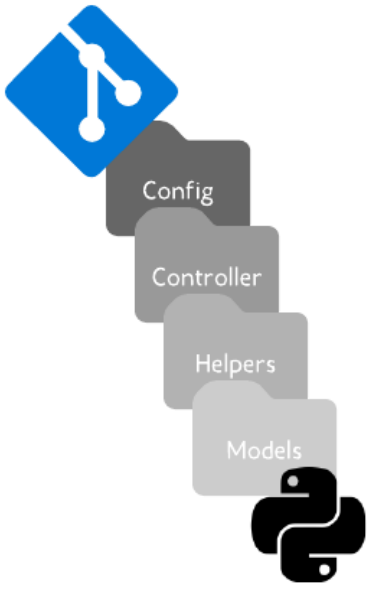
The `'info'` option allows you to obtain more information about the different functionalities of the program.

'exit' to end the program

`'exit'` allows you to stop the `main.py` file process by exiting the while loop.

File Structure

On the right the file structure of our project is displayed. Our git repository mainly consists of four folders and the `main.py` file:

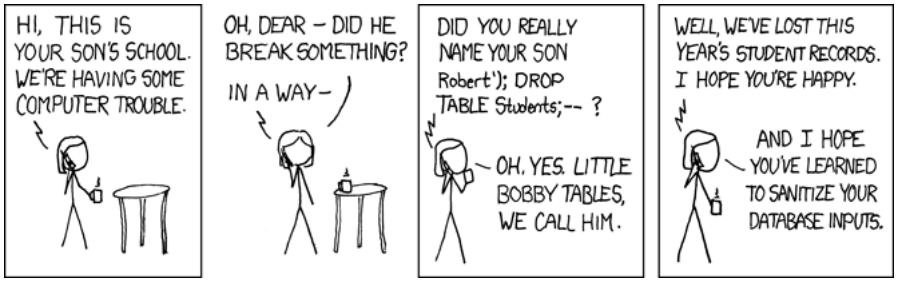


- **`Config`**: Contains the `config.ini` file which stores the database access parameters.
- **`Controller`**: Contains the controller files that communicate with the APIs.
- **`Helpers`**: Contains the DAO.
- **`Models`**: Contains the Classes.
- **`main.py`**: Contains the actual program.

Security

One issue that can never be overemphasized in our digital world is security. In the course we heard that in some databases whole tables can be deleted by simple user inputs through the front end.

Of course we have thought about how we can prevent this in our application to prevent embarrassment during the presentation.



We read ourselves into database injections and tried to penetrate our database and delete our tables. Fortunately without success. Our database query structure seemed to be save enough, so that no user input could ever harm our valuable data.

In order to gain some more insights we tried building an 'unsafe' function that extracts usernames out of the database. We then tried out the notorious commands found through different sources to delete or at least retrieve unsafe data from our database. Unfortunately, also these efforts weren't crowned with success. Thus we accepted the fact that our data is as secure as it can be and well prepared against hacker attacks.

Project

This section describes the project from an organizational perspective. The software development project is in itself not easy to handle, especially for business students who do not have a lot of programming knowledge, which justifies the existence of this section.

Motivation

If the data science project taught us one thing (apart from all the technical knowledge we gained), it probably is the fact that spending nights and nights working on an project whose topic one does not like can be a harsh thing to do. That's why we asked us what subject ignites our spark of enthusiasm. We quickly gained the insight, that this can only be food.

Students often encounter the problem that they have some ingredients in the fridge, which on their own may be delicious flavor carriers, but don't seem to match, however the sheet is turned. This is exactly the misery we want to get rid of with our application.

Organization

To collaborate successfully, we used the version control system [VCS] git. We created a repository on GitHub and thus were able to work remotely and simultaneously on our project (nearly) without any problem. It can be regarded as a nice side-take-away, that we all learned to handle the kinks and peculiarities of git and got introduced to the world of VCS.

Since we were three people working on the project we split the tasks as follows: One person focusing on the front-end and API calls, one person building the Database and one person coding the helper functions of the DAO.

Challenges

This section provides an overview over the challenges encountered during the project and summarizes our key takeaways.

Challenge 1 - On the Search

As our main challenge we regard finding a suitable project that fulfills the needs of the course and doesn't completely surpass the scope. After having changed our minds several times, we tried sticking to the things needed, but doing them as clean as possible, which wasn't always easy. Sometimes the need of writing quick and dirty code is just bigger than the urge to keep everything organized. In the end we found a use case that not only provided a lot of fun to us food-loving people. The application we built is usable, equipped with a nice front-end and enriched with some more functionality, it could well be used as a web application by a large community of users.

We have long lived in the belief that we are the only ones who offer such a solution. Unfortunately there's already a group of people that is working on a similar idea. They called their application 'Yummly 2.0'. However, we think that our version has much more potential and thus we named it 'Yummly 3.0'.

Challenge 2 - Dealing with Databases

MySQL workbench may seem easy and intuitive at first sight and invites the user to start creating tables and building databases before actually knowing what he does. This happened to us as well and reading theory another hour or two would have saved us quite some time.

Luckily we knew from another project that handling locally hosted databases is quite a struggle when working in a team. Therefore we decided to host our database on a server from the beginning on which took some time to set up, but proved to make many things a lot easier and turned out to be worth the struggles.

An additional difficulty we found, when accessing the database via a python package. The functions to be written are often nested and finding an error is quite tedious, especially when the error handling (i.e. try and except statements) have already been implemented and the only output is the the 'answer' of the except statement. What may have helped here is a lecture about error handling. Especially in a project of this scope where we are using many different files and nested scripts, a proper lecture about debugging using Pycharm's integrated debugging function would have provided a lot of additional security while fighting errors.

> Q: Why do you never ask SQL people to help you move your furniture?
> A: They sometimes drop the table.
>
> The Internet

Lessons Learned (and thoughts about the course)

Even tough following this class was not always a pleasure, not to say these words are being written down at 05:00 am, in retrospective there are quite some things we could take with us and would like to say to whom it may concern:

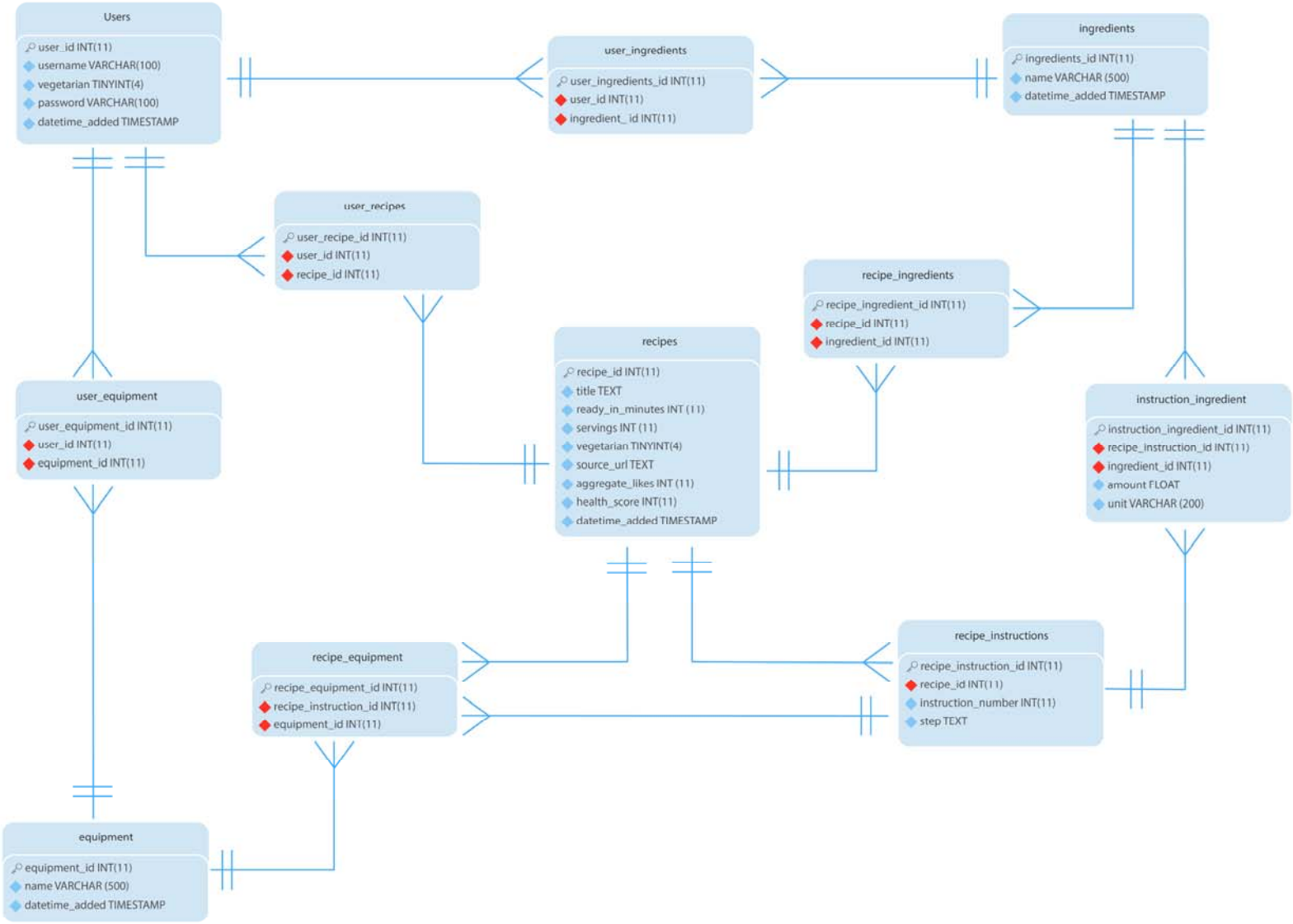
- The course was ideal to extend our knowledge on programming, moving a step closer to understanding 'full stack development'. It was definitely a step up from functional programming and will help us write more organized code in our next projects. The project showed impressively what needs to be done to create a functioning application.
- This course has the legitimacy to exist because it is more than just an introduction to programming. Many courses at the university offer some base knowledge, but don't go beyond that level. This course does exactly that.
- The course might not be suited to people that program for the first time. For someone who has never written a line of code before, we imagine the effort needed to succeed immense, way beyond the three credit points received. Nevertheless we don't think that the class should be stripped down. The level of difficulty it offers makes it attractive to follow to intermediate and advanced programmers.

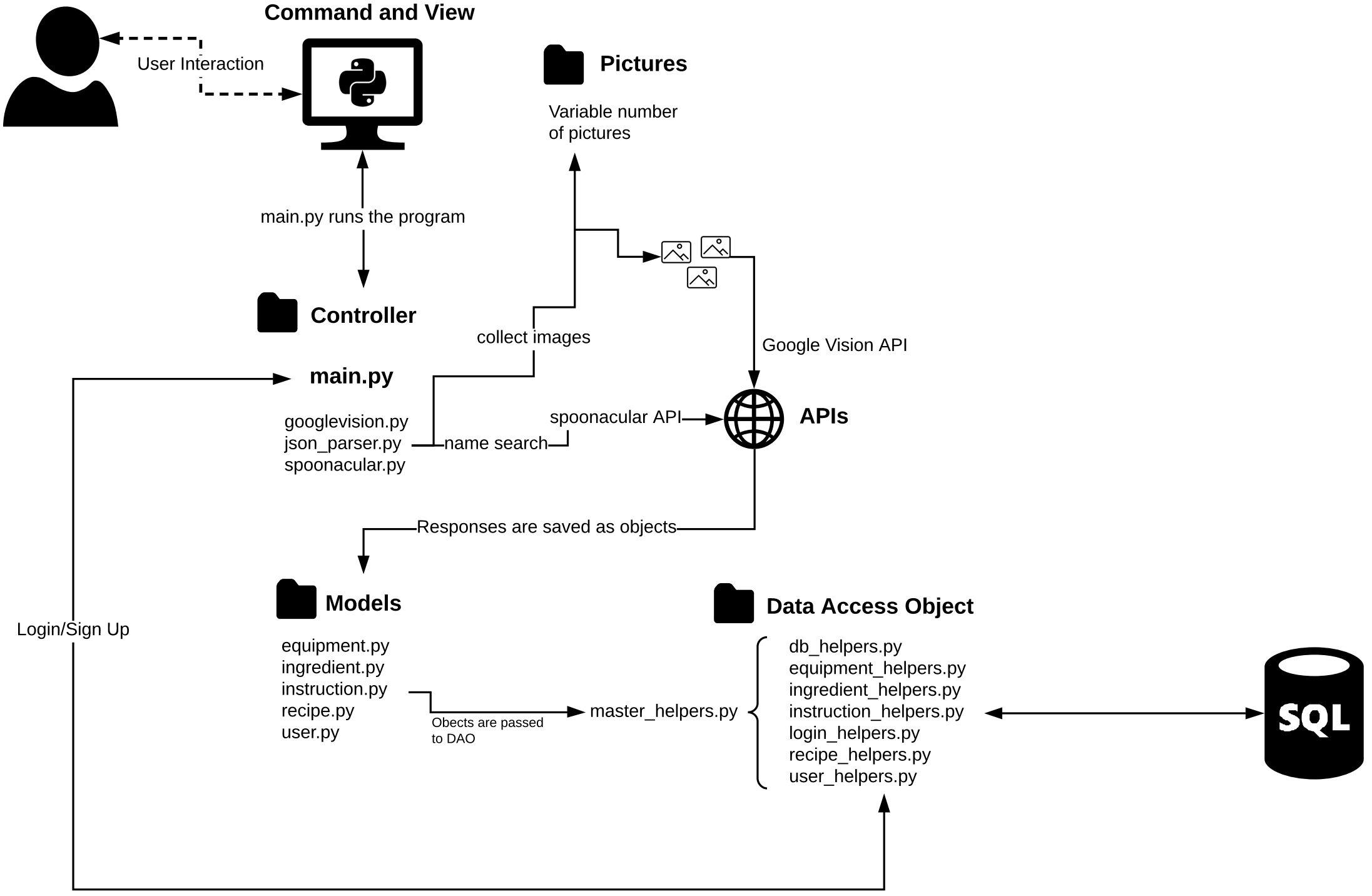
Appendix

The following content can be found in the appendix of this executive README:

- Class diagram
- Database ERD
- Application overview diagram
- Database dump files (not in appendix but in `DB_dump` folder


User
+ username: String
+ hashed_password: String
+ vegetarian_status: Boolean
+ user_id: Int
+ recipes: list of Recipes
+ __init__()
+ printRecipes()
Recipe
+ recipe_id: Int
+ title: String
+ ready_in_minutes: Int
+ servings: Int
+ vegetarian: Boolean
+ source_url: String
+ aggregate_likes: Int
+ health_score: Int
+ ingredients: list of Ingredients
+ instructions: list of Instructions
+ __init__()
+ printRecipeInformations()
Ingredient
+ name: String
+ ingredient_id: Int
+ amount: Int
+ unit: String
+ __init__()
+ printInformation()
Instruction
+ number: Int
+ step: String
+ ingredients: list of ingredients
+ equipment: list of equipment
+ __init__()
+ printInformation()
Equipment
+ name: String
+ equipment_id: Int
+ __init__()
+ printInformation()


Users
user_id INT(11)
username VARCHAR(100)
vegetarian TINYINT(4)
password VARCHAR(100)
datetime_added TIMESTAMP
user_ingredients
user_ingredients_id INT(11)
user_id INT(11)
ingredient_ id INT(11)
ingredients
ingredients_id INT(11)
name VARCHAR (500)
datetime_added TIMESTAMP
user_recipes
user_recipe_id INT(11)
user_id INT(11)
recipe_id INT(11)
recipe_ingredients
recipe_ingredient_id INT(11)
recipe_id INT(11)
ingredient_id INT(11)
recipes
recipe_id INT(11)
title TEXT
ready_in_minutes INT (11)
servings INT (11)
vegetarian TINYINT(4)
source_url TEXT
aggregate_likes INT (11)
health_score INT(11)
datetime_added TIMESTAMP
user_equipment
user_equipment_id INT(11)
user_id INT(11)
equipment_id INT(11)
instruction_ingredient
instruction_ingredient_id INT(11)
recipe_instruction_id INT(11)
ingredient_id INT(11)
amount FLOAT
unit VARCHAR (200)
recipe_equipment
recipe_equipment_id INT(11)
recipe_instruction_id INT(11)
equipment_id INT(11)
recipe_instructions
recipe_instruction_id INT(11)
recipe_id INT(11)
instruction_number INT(11)
step TEXT
equipment
equipment_id INT(11)
name VARCHAR (500)
datetime_added TIMESTAMP


Command and View
User Interaction
Pictures
Variable number
of pictures
main.py runs the program
Controller
collect images
Google Vision API
main.py
googlevision.py
json_parser.py
spoonacular.py
name search
spoonacular API
APIs
Responses are saved as objects
Models
Data Access Object
Login/Sign Up
equipment.py
ingredient.py
instruction.py
recipe.py
user.py
Obects are passed
to DAO
master_helpers.py
db_helpers.py
equipment_helpers.py
ingredient_helpers.py
instruction_helpers.py
login_helpers.py
recipe_helpers.py
user_helpers.py
SQL