Smart Fridge System v.1.0

Documentation

1/2024 Stanislav Bozhimirov

Getting Started

Installation

1. Download [Python 3](https://www.python.org/downloads/)
2. Create folder for the project e.g. C:\Users\youruser\Desktop\SmartFridgeSystem
3. Navigate to folder from your terminal and create virtual environment with following command:

$ python -m venv myenv

1. Activate environment by navigating to myenv/Scripts and execute following command

$ activate.bat

1. Navigate back to C:\Users\youruser\Desktop\SmartFridgeSystem and copy all files of the project
2. Install requirements.txt with following command:

$ pip install -r requirements.txt

1. Run project with following command:

$ python smart\_fridge.py

Documentation of the project in html

Navigate to \SmartFridgeSystem\docs\\_build\html\index.html

## Main goals of the project

* 1. Add, remove, update, delete products in the fridge
  2. Get recipes – by chosen products, random recipes from the API or from saved recipes in favorites
  3. Choose from displayed recipes and if not interested get new recipes
  4. If recipe is chosen to generate shopping list with products that are not in the fridge
  5. The shopping list is displayed as text and as QR code with the shopping list as text, there is an option to send the shopping list to fridge’s users by email or to not registered person with email(where email have to be written)
  6. There is an option that recipe to be displayed on the screen

## User requests

* To set name of the Smart Fridge
* To set users with usernames and emails connected to the Smart Fridge
* Fridge’s name have an option to be changed
* Users can be added, updated, removed from the Smart Fridge
* Users can add products by categories in the Smart Fridge
* Users can change and delete products from the Smart Fridge
* If there is an expired product there is a prompt to delete the product from the Smart Fridge and to remove it from the Smart Fridge
* If all of the amount of a given product is used for the selected recipe, the product to be automatically removed from the Smart Fridge
* Users have 3 options to choose a recipe – from chosen products, from save recipes in favorites, from randomly generated recipes
* When name of a recipe is selected user can see the ingredients needed, the instructions for preparing the meal and the image of the meal (if provided)
* When user want to cook a recipe a shopping list is generated and displayed
* User can write the shopping list on a list of paper, or get it in its phone by QR code, there is an option the shopping list to be send by mail to registered users or to other user not registered in the Smart Fridge again with QR code
* If the user want to cook another meal it can return to the previous page and choose another recipe and add it to the shopping list
* Products from the shopping list can be removed (not all products are in the fridge, some of them may be in the garden and waiting to be collected) and QR code is updated
* User can display all products and recipe instructions and start cooking.

## Target auditory

This Smart Fridge System can be used by all users that use fridges in their home and wants to optimize food expenses and food waste. The Smart Fridge System can give easy access of all family members to many recipes to choose from and to optimize cooking by using most of the products in the fridge and to make less shopping expenses, by optimizing already bought food usage and eliminate food waste. Easily generated shopping list and easily sending it to other family members can optimize the time of shopping and expenses for cooking

## About the application

* 1. *Architecture of the application:*

**Model-View-Controller or MVC** – is a pattern in software design commonly used to implement user interfaces, data, and controlling logic. It emphasizes a separation between the software's business logic and display.

* 1. *Files of the application*
* smart\_fridge.py – This is the main file of the app that starts the application.
* model.py – File that contains the main models of the app.
* View.py = File that contains all the needed information to visualize the app.
* Controller.py – File that connects user choices with the models and DB and frontend of the app and contains all the business login of the project
* Init.py – file that makes the initial DB, needed to start the application
* Db.py – file that contains all of the logic manipulating with the DB
* .env – file that contains the project variables such as API keys etc.
* Requirements.txt – file that contains a list of packages or libraries needed to work on a project that can all be installed with the file.
* Images folder – contains all of the images needed for the application to work properly
* Docs folder – contains the documentation of the project generated as HTML file, it can be found in the \_build subfolder > html subfolder > index.html
  1. *Technoligies:*
* Code First approach – in this approach in programming, the main focus is the domain of the application and programmer start creating classes for the domain entity rather than design the database first and then create the classes which match the database design.
* Dependency injection – used to make a class independent of its dependencies or to create a loosely coupled program. Dependency injection is useful for improving the reusability of code. Likewise, by decoupling the usage of an object, more dependencies can be replaced without needing to change class.
  1. *Libraries and modules:*

*The application is developed with* ***PyCharm*** *- an*[*integrated development environment*](https://en.wikipedia.org/wiki/Integrated_development_environment)*(IDE) used for programming in*[*Python*](https://en.wikipedia.org/wiki/Python_(programming_language))*.*

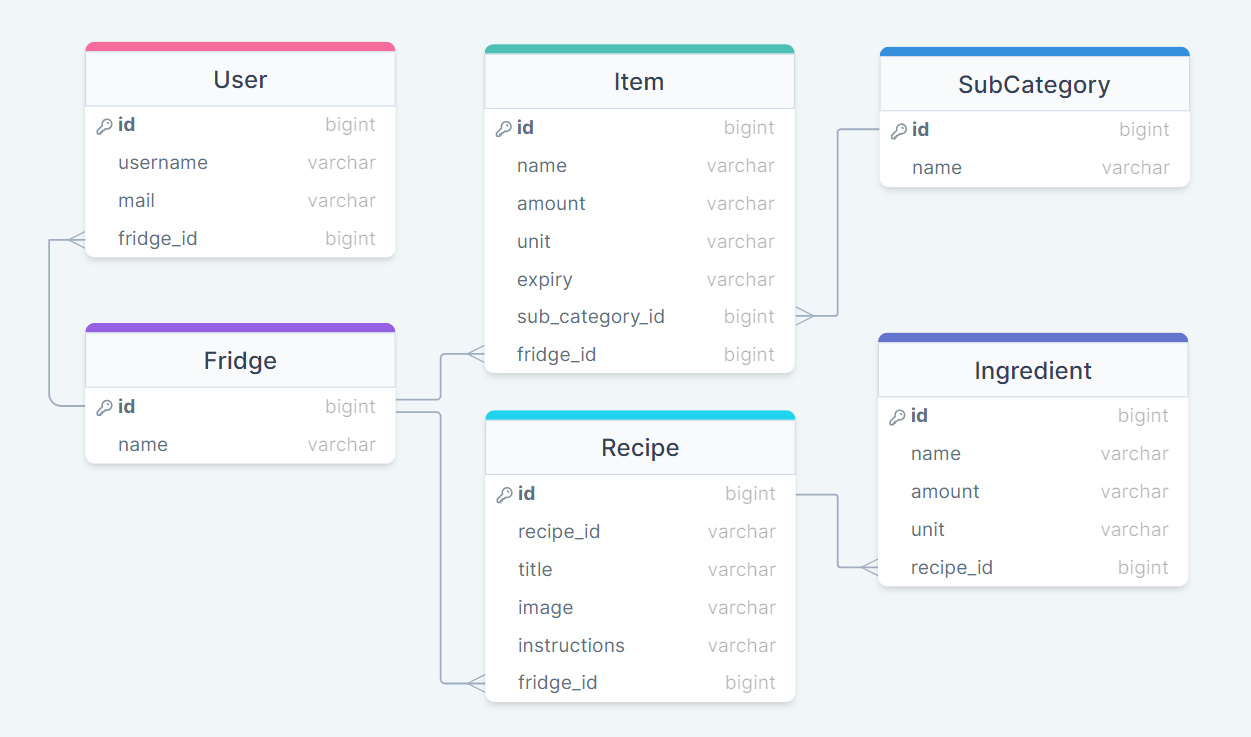
*For version control and code hosting platform is used* ***GitHub.***

*API for recipes:* [*spoonacular API*](https://spoonacular.com/food-api)

* ***Tkinter*** *- Tkinter is a Python binding to the Tk GUI* toolkit. It is the standard Python interface to the Tk GUI toolkit, and is Python's de facto standard GUI. Tkinter is included with standard Linux, Microsoft Windows and macOS installs of Python. License. Python license.
* **Sqlalchemy** – SQLAlchemy is the Python SQL toolkit and Object Relational Mapper that gives application developers the full power and flexibility of SQL. It provides a full suite of well known enterprise-level persistence patterns, designed for efficient and high-performing database access, adapted into a simple and Pythonic domain language.
* **Time** - This module provides various time-related functions.
* **Os** – This module provides a portable way of using operating system dependent functionality.
* **Typing** – This module provides runtime support for type hints.
* **Math** – This module provides access to the mathematical functions defined by the C standard.
* **Io** – The [io](https://docs.python.org/3/library/io.html?highlight=io" \l "module-io" \o "io: Core tools for working with streams.) module provides Python’s main facilities for dealing with various types of I/O. There are three main types of I/O: *text I/O*, *binary I/O* and *raw I/O*. These are generic categories, and various backing stores can be used for each of them. A concrete object belonging to any of these categories is called a [file object](https://docs.python.org/3/glossary.html#term-file-object). Other common terms are *stream* and *file-like object*.
* **Urllib** – a package that collects several modules for working with URLs
* **Datetime** – this module supplies classes for manipulating dates and times. While date and time arithmetic is supported, the focus of the implementation is on efficient attribute extraction for output formatting and manipulation.
* **Dotenv** – Python-dotenv reads key-value pairs from a .env file and can set them as environment variables.
* **PIL** – Python Imaging Library is a free and open-source additional library for the Python programming language that adds support for opening, manipulating, and saving many different image file formats. It is available for Windows, Mac OS X and Linux. The latest version of PIL is 1.1.
* **Segno** – Pure Python QR Code generator with no dependencies.
* **Validate\_email** – a package for Python that check if an email is valid, properly formatted and really exists.
* **Requests** - a simple, yet elegant, HTTP library.
  1. *Data Base Management System:*

SQLite – a database [library](https://en.wikipedia.org/wiki/Library_(computing)) that [software developers](https://en.wikipedia.org/wiki/Programmer) embed in their [apps](https://en.wikipedia.org/wiki/Application_software). As such, it belongs to the family of [embedded databases](https://en.wikipedia.org/wiki/Embedded_database). It is the most widely deployed database engine, as it is used by several of the top [web browsers](https://en.wikipedia.org/wiki/Web_browser), [operating systems](https://en.wikipedia.org/wiki/Operating_system), [mobile phones](https://en.wikipedia.org/wiki/Mobile_phone), and other [embedded systems](https://en.wikipedia.org/wiki/Embedded_system).

* 1. *Diagrams and connections in the application:*
* Models’ diagram



* 1. *Modules, Classes, Functions and methods:*
* *Smart\_fridge.py:*
* *The file where is defined the main method of the app. Here is created the DB and the controller and the main method of the controller is called*
* *InitDB.py:*
* *This file contains a class CreateDB that is an engine class that makes the database needed for the app*
* *Db.py:*
* *This file contains a wrapper class DataBase with only static methods that interacts with the DB.*

**Static add\_item\_to\_fridge(*session: Session*, *new\_item:***[***Item***](http://localhost:63342/main.py/project/docs/_build/html/model.html#model.Item)**)→ None**[**[source]**](http://localhost:63342/main.py/project/docs/_build/html/_modules/db.html#DataBase.add_item_to_fridge)

Add new Item to the DB

:param session: Session current app session

:param new\_item: Item new food item generated by user to be added to the DB

**Static add\_recipe\_to\_fridge(*session: Session*, *new\_recipe:***[***Recipe***](http://localhost:63342/main.py/project/docs/_build/html/model.html#model.Recipe)**)→ None**[**[source]**](http://localhost:63342/main.py/project/docs/_build/html/_modules/db.html#DataBase.add_recipe_to_fridge)

Add new Recipe to the DB

:param session: Session current app session

:param new\_recipe: Recipe new recipe object generated by user to be added to the DB

**Static check\_for\_fridge(*session: Session*)→**[**Fridge**](http://localhost:63342/main.py/project/docs/_build/html/model.html#model.Fridge)[**[source]**](http://localhost:63342/main.py/project/docs/_build/html/_modules/db.html#DataBase.check_for_fridge)

Check if there is a fridge in the DB

:param session: Session current app session :return: Fridge instance of the fridge in DB if any, else return empty list

**Static check\_fridge\_name(*name: str*)→ str[[source]](http://localhost:63342/main.py/project/docs/_build/html/_modules/db.html" \l "DataBase.check_fridge_name)**

Refactor fridge’s name by making it capitalize if not and adding Fridge to it :param name: str name given by user

**Static check\_if\_recipe\_in\_fridge(*session: Session*, *new\_recipe:***[***Recipe***](http://localhost:63342/main.py/project/docs/_build/html/model.html#model.Recipe)**)→ list**[**[source]**](http://localhost:63342/main.py/project/docs/_build/html/_modules/db.html#DataBase.check_if_recipe_in_fridge)

Get recipe object from the DB

:param session: Session current app session

:param new\_recipe: Recipe a recipe object that have to be checked if it is in the DB

:return list with only one result if recipe object found in DB or empty list of no such recipe in the DB

**Static del\_user\_by\_user\_obj(*session: Session*, *user:***[***User***](http://localhost:63342/main.py/project/docs/_build/html/model.html#model.User)**)→ None**[**[source]**](http://localhost:63342/main.py/project/docs/_build/html/_modules/db.html#DataBase.del_user_by_user_obj)

Delete specific user form the DB

:param session: Session current app session

:param user: User specific user given that have to be removed from the DB

**Static delete\_item\_from\_fridge(*session: Session*, *c\_item:***[***Item***](http://localhost:63342/main.py/project/docs/_build/html/model.html#model.Item)**)→ None**[**[source]**](http://localhost:63342/main.py/project/docs/_build/html/_modules/db.html#DataBase.delete_item_from_fridge)

Delete food item from the DB

:param session: Session current app session

:param c\_item: Item food item object that have to be deleted from the DB

**Static delete\_zero\_amount\_item\_from\_fridge(*session: Session*, *all\_items: list*)→ None**[**[source]**](http://localhost:63342/main.py/project/docs/_build/html/_modules/db.html#DataBase.delete_zero_amount_item_from_fridge)

Delete food items from the DB if their amount is 0 or if their unit is ‘g or ‘ml and their amount is less than 10

:param session: Session current app session

:param all\_items: list a list of all food item objects in DB that have to be checked if amount is 0 or less than 10 if their units are ‘g’ or ‘ml’

**Static get\_all\_items\_from\_fridge(*session: Session*, *fridge\_id: int*)→ list**[**[source]**](http://localhost:63342/main.py/project/docs/_build/html/_modules/db.html#DataBase.get_all_items_from_fridge)

Get all food items from the DB connected to specific fridge

:param session: Session current app session

:param fridge\_id: int id of the fridge that have to be checked for food items

:return list with all items from the DB connected to the given fridge id

**Static get\_all\_recipes(*session: Session*, *fridge\_id: int*)→ list**[**[source]**](http://localhost:63342/main.py/project/docs/_build/html/_modules/db.html#DataBase.get_all_recipes)

Get all recipes from the DB connected to specific fridge

:param session: Session current app session

:param fridge\_id: int id of the fridge that have to be checked for recipes

:return list with all recipes from the DB saved to the given fridge id

**Static get\_all\_sub\_cat(*session: Session*)→ list**[**[source]**](http://localhost:63342/main.py/project/docs/_build/html/_modules/db.html#DataBase.get_all_sub_cat)

Get all sub categories from the DB

:param session: Session current app session

:return list with all sub categories from the DB

**Static get\_all\_users(*session: Session*, *fridge\_id: int*)→ list**[**[source]**](http://localhost:63342/main.py/project/docs/_build/html/_modules/db.html#DataBase.get_all_users)

Get all users from the DB connected to specific fridge

:param session: Session current app session

:param fridge\_id: int id of the fridge that have to be checked for recipes

:return list with all users from the DB connected to the given fridge id

**Static get\_data\_for\_item\_from\_name(*session: Session*, *name\_item: str*)→ list**[**[source]**](http://localhost:63342/main.py/project/docs/_build/html/_modules/db.html#DataBase.get_data_for_item_from_name)

Get food items from the DB by given name

:param session: Session current app session

:param name\_item: str name of the searched item in the DB

:return list with only one result if item found or empty list of no item with such name in the DB

**Static get\_sub\_id(*session: Session*, *sub\_name: str*)→ int[[source]](http://localhost:63342/main.py/project/docs/_build/html/_modules/db.html" \l "DataBase.get_sub_id)**

Get the id of a sub category ot item by given sub category name

:param session: Session current app session

:param sub\_name: str name of the sub category that have to be found

:return: id of the searched sub category

**Static make\_fridge(*session: Session*, *fridge\_name: str*)→ None**[**[source]**](http://localhost:63342/main.py/project/docs/_build/html/_modules/db.html#DataBase.make_fridge)

Make fridge instance with given name and add it to DB

:param session: Session current app session

:param fridge\_name: str fridge name given by user

**Static set\_data\_for\_item\_from\_name(*session: Session*, *name\_item: str*, *new\_name: str*, *new\_amount: int*, *new\_unit: str*, *new\_expiry: str*, *new\_sub: str*)→ None**[**[source]**](http://localhost:63342/main.py/project/docs/_build/html/_modules/db.html#DataBase.set_data_for_item_from_name)

Set new data for food item in the DB

:param session: Session current app session

:param name\_item: str name of the food item in the DB

:param new\_name: str the new name of the food item in the DB :param new\_amount: int the new amount of the food item in the DB

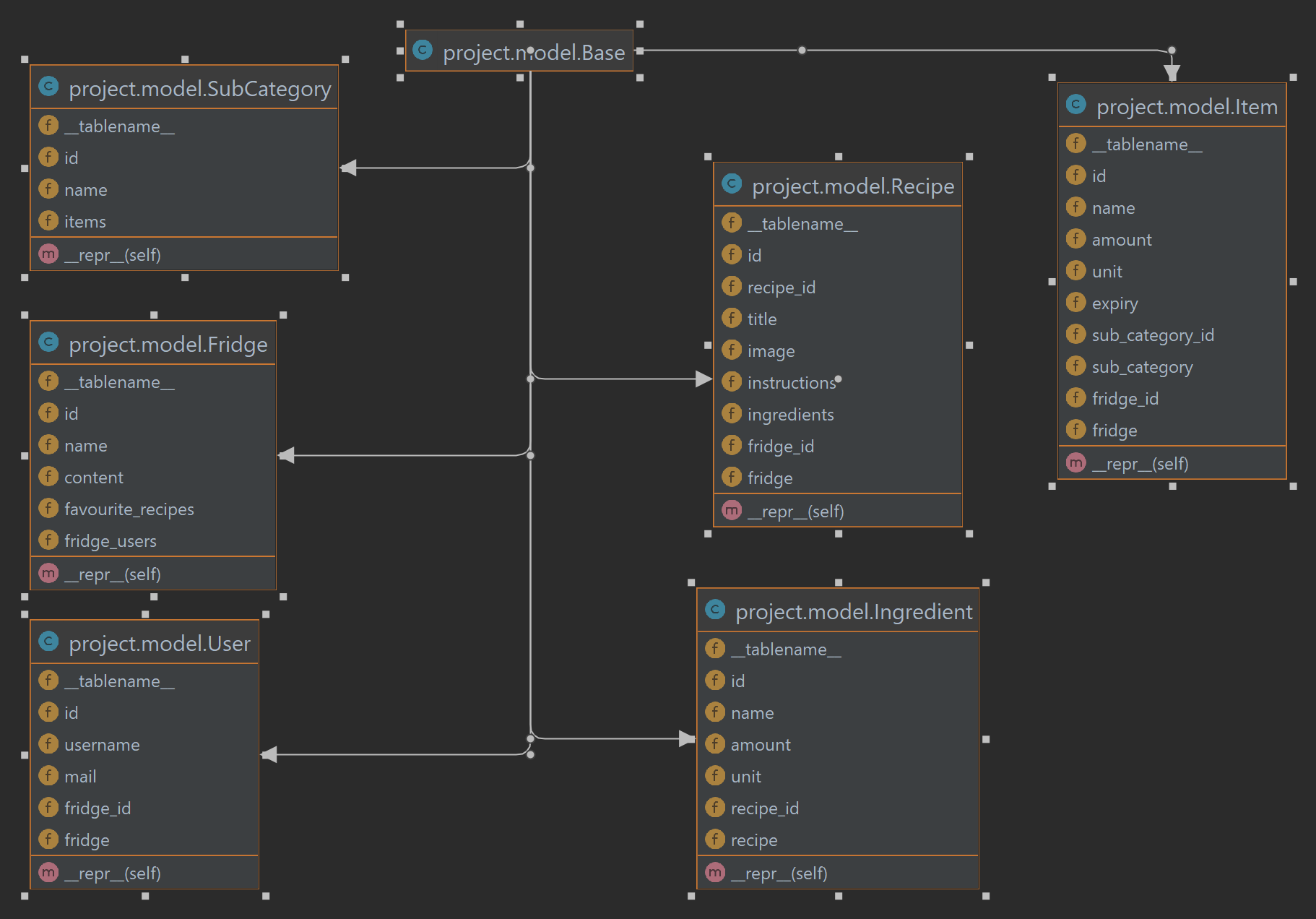
:param new\_unit: str the new unit of the food item in the DB

:param new\_expiry: str the new expiry date of the food item in the DB

:param new\_sub: str the new sub category of the food item in the DB

* *Model.py:*

This module contains several classes that are models in DB



* *Controllers.py:*
* *This file contains a class Controller that makes connections between users, views and db*

**action\_buttons(*action: str*)→ None**[**[source]**](http://localhost:63342/main.py/project/docs/_build/html/_modules/controller.html#Controller.action_buttons)

buttons that can change view frames according to action keyword

:param action: str keyword needed for buttons to perform different action

**add\_new\_user()→ None**[**[source]**](http://localhost:63342/main.py/project/docs/_build/html/_modules/controller.html#Controller.add_new_user)

add new user in DB if there are filled fields of name and email and display it in the view in empty location field

**change\_api\_key()→ None**[**[source]**](http://localhost:63342/main.py/project/docs/_build/html/_modules/controller.html#Controller.change_api_key)

change api key from the env file

**change\_spinbox(*e*)→ None**[**[source]**](http://localhost:63342/main.py/project/docs/_build/html/_modules/controller.html#Controller.change_spinbox)

changes values in unit spinbox according to if item is new, or got from fridge and has to be updated, if got from fridge, ml and l and g and kg are pairs possible for change, if the same item, else have to be made new item with different units

**Parameters:**

**e** – Event to trigger changes

**check\_for\_expired\_products()→ None**[**[source]**](http://localhost:63342/main.py/project/docs/_build/html/_modules/controller.html#Controller.check_for_expired_products)

check if there are expired items in the fridge and add them to expired\_products variable

**check\_if\_ingredient\_is\_in\_fridge(*ingredient:***[***Ingredient***](http://localhost:63342/main.py/project/docs/_build/html/model.html#model.Ingredient)**)→ None**[**[source]**](http://localhost:63342/main.py/project/docs/_build/html/_modules/controller.html#Controller.check_if_ingredient_is_in_fridge)

check if ingredient is in fridge, if in fridge and amount is enough for the recipe, the food item in fridge is updated with removed amount needed for the recipe, else ingredient is not removed from shopping list

:param ingredient: Ingredient needed to prepare the recipe

**check\_if\_ingredient\_is\_shopping\_list(*ingredient:***[***Ingredient***](http://localhost:63342/main.py/project/docs/_build/html/model.html#model.Ingredient)**)→**[**Ingredient**](http://localhost:63342/main.py/project/docs/_build/html/model.html#model.Ingredient)[**[source]**](http://localhost:63342/main.py/project/docs/_build/html/_modules/controller.html#Controller.check_if_ingredient_is_shopping_list)

check if ingredient is in shopping list by checking substrings of any name to eliminate duplications

:param ingredient: Ingredient to be checked if is in the shopping variable

:return: ingredient if not in shopping variable, else return None

**check\_if\_item\_in\_fridge(*n: str*, *e: str*, *q: str*, *u: str*, *l\_n: str*)→ tuple**[**[source]**](http://localhost:63342/main.py/project/docs/_build/html/_modules/controller.html#Controller.check_if_item_in_fridge)

check if item in fridge by all properties, if everything the same sum amounts, if no such item returned bool value is False, if such item but diff expiry or units, make new item with same name extended with number, else make new item

**Parameters:**

* **n** – str name of item to be checked
* **e** – str expiry date of item to be checked
* **q** – str amount of item to be checked
* **u** – str unit of item to be checked
* **l\_n** – str last name chosen to be compared with

**Returns:** tuple with all the properties and a bool variable to\_be\_updated

**check\_ingredients\_not\_to\_buy(*ingredient:***[***Ingredient***](http://localhost:63342/main.py/project/docs/_build/html/model.html#model.Ingredient)**)→**[**Ingredient**](http://localhost:63342/main.py/project/docs/_build/html/model.html#model.Ingredient)[**[source]**](http://localhost:63342/main.py/project/docs/_build/html/_modules/controller.html#Controller.check_ingredients_not_to_buy)

check if specific ingredient has to be bought

:param ingredient: Ingredient object that has to be checked if it has to be bought :return: ingredient if it has to be bought or empty string if not to be bought

***Static* check\_name\_len(*name1: str*, \**name2: str*)→ str[[source]](http://localhost:63342/main.py/project/docs/_build/html/_modules/controller.html" \l "Controller.check_name_len)**

choose shorter name form the API of the ingredient

:param name1: str one of the names taken from the API

:param name2: str one of the names taken from the API, if any

:returns string of the ingredients’ shorter name

***Static* chosen(*list\_chosen\_ingr: list*)→ str[[source]](http://localhost:63342/main.py/project/docs/_build/html/_modules/controller.html" \l "Controller.chosen)**

makes string of ingredients concatenated as needed to make the proper query for the APi

:param list\_chosen\_ingr: list chosen ingredients as list

:returns string of the ingredients as needed to make proper query to API

**clear\_fields()→ None**[**[source]**](http://localhost:63342/main.py/project/docs/_build/html/_modules/controller.html#Controller.clear_fields)

clear all fields from add item in fridge view and resets connected variables

**close\_app()→ None**[**[source]**](http://localhost:63342/main.py/project/docs/_build/html/_modules/controller.html#Controller.close_app)

close the application

**create\_recipe(*r\_id: int*, *r\_name: str*, *r\_image: str*, *r\_instructions: str*, *r\_ingredients: list*)→**[**Recipe**](http://localhost:63342/main.py/project/docs/_build/html/model.html#model.Recipe)[**[source]**](http://localhost:63342/main.py/project/docs/_build/html/_modules/controller.html#Controller.create_recipe)

create recipe object from given arguments and creates ingredients objects connected to that recipe

:param r\_id: int id of the recipe as given by the API

:param r\_name: str title of the recipe

:param r\_image: web link to recipe’s image as string

:param r\_instructions: instructions for making recipe with removed tags

:param r\_ingredients: ingredient objects connected to the given recipe

:return: Recipe object with all the information needed

**delete\_expired()→ None**[**[source]**](http://localhost:63342/main.py/project/docs/_build/html/_modules/controller.html#Controller.delete_expired)

delete items in expired\_products variable from DB and destroy pop up button with expired items from view

**delete\_user(*position: int*)→ None**[**[source]**](http://localhost:63342/main.py/project/docs/_build/html/_modules/controller.html#Controller.delete_user)

delete user from specific position in the view and delete user from the DB

:param position: int position of the user in view

**destroy\_recipe\_btn()→ None**[**[source]**](http://localhost:63342/main.py/project/docs/_build/html/_modules/controller.html#Controller.destroy_recipe_btn)

destroy recipe and ingredients pop up from view

**destroy\_top\_btn()→ None**[**[source]**](http://localhost:63342/main.py/project/docs/_build/html/_modules/controller.html#Controller.destroy_top_btn)

destroy pop up button from view

**fill\_shopping\_list\_content()→ None**[**[source]**](http://localhost:63342/main.py/project/docs/_build/html/_modules/controller.html#Controller.fill_shopping_list_content)

from shopping variable takes all ingredients and concatenate them in proper form for creating qr code, also display up to 16 ingredients in user’s view in 2 columns

**generate\_choices(*\*args*)→ None**[**[source]**](http://localhost:63342/main.py/project/docs/_build/html/_modules/controller.html#Controller.generate_choices)

set choices variable with all the items in the fridge so users can choose for items in fridge

:param args: if args not changes choices variable

**get\_bulk\_recipes\_by\_ids(*ids: list*)→ list**[**[source]**](http://localhost:63342/main.py/project/docs/_build/html/_modules/controller.html#Controller.get_bulk_recipes_by_ids)

from list of ids for each id make an API call and create recipe object with returned information

:param ids: list ids of searched recipes in list

:returns list with recipe objects for each of the searched id with full information

**get\_random\_recipes(*number: int*)→ list**[**[source]**](http://localhost:63342/main.py/project/docs/_build/html/_modules/controller.html#Controller.get_random_recipes)

get random recipes from API, makes a recipe object with all of the recipe’s information and add it to a list

Parameters: **number** – int number of recipes to be taken

:returns list of recipes object with information of each recipe

**get\_recipe\_by\_name(*name: str*)→**[**Recipe**](http://localhost:63342/main.py/project/docs/_build/html/model.html#model.Recipe)[**[source]**](http://localhost:63342/main.py/project/docs/_build/html/_modules/controller.html#Controller.get_recipe_by_name)

get recipe by name from recipes from chosen products variable, if not there check if recipe is not in fridge’s favourite recipes

Parameters: **name** – str title of recipe

:returns searched recipe

**get\_recipes\_by\_items(*number: int*)→ list**[**[source]**](http://localhost:63342/main.py/project/docs/_build/html/_modules/controller.html#Controller.get_recipes_by_items)

get recipes from API by chosen ingredients, then takes their ids and make another call to API to collect information for each recipe by their id and add it to a list

Parameters: **number** – int number of recipes to be taken

:returns list of recipes object with information of each recipe

**handle\_letter(*ltr: str*)→ None**[**[source]**](http://localhost:63342/main.py/project/docs/_build/html/_modules/controller.html#Controller.handle_letter)

take pressed letter or word and place it in filed for writing or call command if needed

:param ltr: str letter or word to be used for writing or call command

**initial\_data()→ None**[**[source]**](http://localhost:63342/main.py/project/docs/_build/html/_modules/controller.html#Controller.initial_data)

open view to initial set fridge’s name or to change it, also here user can add users to fridge, update their info or delete them, and after that takes user to welcome screen

**item\_action\_buttons(*action: str*)→ None**[**[source]**](http://localhost:63342/main.py/project/docs/_build/html/_modules/controller.html#Controller.item_action_buttons)

group of buttons in items view that act differently according to keyword, button can clear fields, can add item to DB, modify item, or delete item from view and DB, here is the action for removing specific amount of item taken for cooking :param action: str keyword that tells how a button from a group to act

**main()→ None**[**[source]**](http://localhost:63342/main.py/project/docs/_build/html/_modules/controller.html#Controller.main)

calls the main method of the View class

**make\_letter\_buttons(*parent: Frame*)→ None**[**[source]**](http://localhost:63342/main.py/project/docs/_build/html/_modules/controller.html#Controller.make_letter_buttons)

make keyboard buttons in specific frame and every button returns letter of calls function

:param parent: Frame where the keyboard to be displayed in

***Static* make\_qr\_code(*data: str*)→ PhotoImage[[source]](http://localhost:63342/main.py/project/docs/_build/html/_modules/controller.html" \l "Controller.make_qr_code)**

make qr code as PhotoImage from specific data and save it to a predefined destination, resizing the image to fit the view

:param data: str string of information that have to be converted into qr code :returns PhotoImage of the qr code to be displayed on the view

***static* make\_unit(*amount: str*, *unit: str*)→ tuple**[**[source]**](http://localhost:63342/main.py/project/docs/_build/html/_modules/controller.html#Controller.make_unit)

take amount and unit and convert them to usable amount and unit

:param amount: str amount of ingredient for specific unit as string as taken from the API

:param unit: str unit of ingredient taken from the API :returns tuple of modified amount and unit

**open\_settings(*command: str*)→ None**[**[source]**](http://localhost:63342/main.py/project/docs/_build/html/_modules/controller.html#Controller.open_settings)

set wait variable and open settings view

:param command: str to be places into wait variable

***Static* prepare\_text\_for\_display(*text: str*, *symbols: int*)→ str[[source]](http://localhost:63342/main.py/project/docs/_build/html/_modules/controller.html" \l "Controller.prepare_text_for_display)**[**ℑ**](http://localhost:63342/main.py/project/docs/_build/html/controller.html#controller.Controller.prepare_text_for_display)

prepare text for inserting into text box by placing words, so they can all be visible by the user :param text: str text of the recipe to be formatted, so it can be properly displayed on the screen :param symbols: int number of symbols to be displayed in one row, this value depends on the font of the text and width of the text filed :return: str with formatted text

**recipe\_action\_buttons(*action: str*)→ None**[**[source]**](http://localhost:63342/main.py/project/docs/_build/html/_modules/controller.html#Controller.recipe_action_buttons)

group of buttons that make different things according to action keyword, according to action are displayed recipes from chosen products, random recipes or favourites recipes, user can add recipe to favourites, add ingredients to shopping list and get the shopping list

Parameters: **action** – str according to this action button perform different action

**remove(*\*x: str*)→ None**[**[source]**](http://localhost:63342/main.py/project/docs/_build/html/_modules/controller.html#Controller.remove)

remove ingredient from shopping list view and shopping variable and create updated qr code :param x: str ingredient to be removed

***static* remove\_digits(*name: str*)→ str[[source]](http://localhost:63342/main.py/project/docs/_build/html/_modules/controller.html" \l "Controller.remove_digits)**

remove digits from items’ name if any, this is needed to take original ingredient’s name for choosing recipe or updating amount of the ingredient after cooking

**remove\_from\_fridge\_if\_any()→ None**[**[source]**](http://localhost:63342/main.py/project/docs/_build/html/_modules/controller.html#Controller.remove_from_fridge_if_any)

takes all items from the DB and check if there are items with zero or small amounts and remove them from the fridge

***static* remove\_li(*instructions: str*)→ str[[source]](http://localhost:63342/main.py/project/docs/_build/html/_modules/controller.html" \l "Controller.remove_li)**

removes html tags from instructions, so they can be displayed correctly in the view

:param instructions: str instructions taken from the API telling how a recipe can be prepared

:returns string with instructions without html tags

**search\_item(*name\_item: str*)→ list**[**[source]**](http://localhost:63342/main.py/project/docs/_build/html/_modules/controller.html#Controller.search_item)

search specific item by name and returns it if presented

:param name\_item: str name of the food item that is searched

:return: list with one item or empty list in order if the item is presented in the db

**send\_mail(*\*arg*)→ None**[**[source]**](http://localhost:63342/main.py/project/docs/_build/html/_modules/controller.html#Controller.send_mail)

generate qr code with shopping list and email if any to be sent to specific user get user email from DB or if list have to be sent to other user than email have to be manually written

:param arg: str email of the user that have to receive shopping list

**set\_fridge\_object()→ type[**[**Fridge**](http://localhost:63342/main.py/project/docs/_build/html/model.html#model.Fridge)**]**[**[source]**](http://localhost:63342/main.py/project/docs/_build/html/_modules/controller.html#Controller.set_fridge_object)[**ℑ**](http://localhost:63342/main.py/project/docs/_build/html/controller.html#controller.Controller.set_fridge_object)

get Fridge object from DB :return: Fridge object

**set\_recipe(*index\_of\_recipe: int = 0*)→ None**[**[source]**](http://localhost:63342/main.py/project/docs/_build/html/_modules/controller.html#Controller.set_recipe)[**ℑ**](http://localhost:63342/main.py/project/docs/_build/html/controller.html#controller.Controller.set_recipe)

set recipe data for cooking according to chosen recipe button :param index\_of\_recipe: int default value = 0, index of the recipe in the list of chosen recipes

**show\_recipe()→ None**[**[source]**](http://localhost:63342/main.py/project/docs/_build/html/_modules/controller.html#Controller.show_recipe)

display recipe and ingredients in the view as pop ups for easy cooking

**update\_user(*position: int*)→ None**[**[source]**](http://localhost:63342/main.py/project/docs/_build/html/_modules/controller.html#Controller.update_user)

update user info in specific position in the view and update user’s info in DB :param position: in t position of the user in the view

* *View.py:*
* *This file contains a class View that makes the interface of the application*

**check\_if\_chosen\_recipe\_image()→ None**[**[source]**](http://localhost:63342/main.py/project/docs/_build/html/_modules/view.html#View.check_if_chosen_recipe_image)

check if chosen recipe has image and if it has it is displayed in recipe view

**chosen\_item\_str(*b: Event*)→ None**[**[source]**](http://localhost:63342/main.py/project/docs/_build/html/_modules/view.html#View.chosen_item_str)

select item from fridge and add it to chosen products, remove from name any digits if any to prevent repetition of items selected

:param b: Event that triggers selection of item

**chosen\_product\_str(*c: Event*)→ None**[**[source]**](http://localhost:63342/main.py/project/docs/_build/html/_modules/view.html#View.chosen_product_str)

select product from list with chosen products to cook for and remove it from chosen\_products list and update chosen products view

:param c: Event that triggers selection of product

**clear\_chosen()→ None**[**[source]**](http://localhost:63342/main.py/project/docs/_build/html/_modules/view.html#View.clear_chosen)

delete all chosen products from view, set chosen\_products variable to empty list

**clear\_used\_missed\_instructions()→ None**[**[source]**](http://localhost:63342/main.py/project/docs/_build/html/_modules/view.html#View.clear_used_missed_instructions)

clear chosen products list view and variable, clear ingredients view, clear recipe description view

**display\_expired\_for\_deleting()→ None**[**[source]**](http://localhost:63342/main.py/project/docs/_build/html/_modules/view.html#View.display_expired_for_deleting)

check if items in expired\_products list and display pop up button for prompt to delete them

**enable\_buttons()→ None**[**[source]**](http://localhost:63342/main.py/project/docs/_build/html/_modules/view.html#View.enable_buttons)

enable add, update, clear, delete buttons from add item group buttons

**fall\_back\_under()→ None**[**[source]**](http://localhost:63342/main.py/project/docs/_build/html/_modules/view.html#View.fall_back_under)

displays last shown frame if any and update frame stack

**frame\_stack*= []***

**get\_values()→ tuple**[**[source]**](http://localhost:63342/main.py/project/docs/_build/html/_modules/view.html#View.get_values)

get item values - name, amount, unit, expiry date, sub category

:return: tuple with all values to make Item

**list\_ingredients()→ str[[source]](http://localhost:63342/main.py/project/docs/_build/html/_modules/view.html" \l "View.list_ingredients)**

get all ingredients in chosen\_recipe variable and returns all ingredients that have to be bought as string

:return: str of all ingredients in chosen\_recipe variable

**main()→ None**[**[source]**](http://localhost:63342/main.py/project/docs/_build/html/_modules/view.html#View.main)

calls the mainloop function of the View class

**make\_add\_item\_window(*parent: Frame*)→ Frame**[**[source]**](http://localhost:63342/main.py/project/docs/_build/html/_modules/view.html#View.make_add_item_window)

make add item frame with where user can add, update, delete items and displays all items in fridge

Parameters: **parent** – Frame of the parent Frame

Returns: currently made Frame

**make\_add\_window(*parent: Frame*)→ Frame**[**[source]**](http://localhost:63342/main.py/project/docs/_build/html/_modules/view.html#View.make_add_window)

make buttons for all the sub categories so user can place new item in correct sub category

Parameters: **parent** – Frame of the parent Frame

Returns: currently made Frame

**make\_aks\_window(*parent: Frame*)→ Frame**[**[source]**](http://localhost:63342/main.py/project/docs/_build/html/_modules/view.html#View.make_aks_window)

make ask view frame when user is asked to choose if to add item or to cook and display these 2 buttons

:param parent: Frame of the parent Frame

:return: currently made Frame

**make\_cook\_chosen\_window(*parent: Frame*)→ Frame**[**[source]**](http://localhost:63342/main.py/project/docs/_build/html/_modules/view.html#View.make_cook_chosen_window)

make cook chosen view where user can select recipes and make shopping list and cook adn displays all items needed

Parameters: **parent** – Frame of the parent Frame

Returns: currently made Frame

**make\_cook\_window(*parent: Frame*)→ Frame**[**[source]**](http://localhost:63342/main.py/project/docs/_build/html/_modules/view.html#View.make_cook_window)

make sub cook frame where user to choose where to get recipes from, make recipe from items in fridge, get random recipe, or choose among favourite resipes, if any and displays all needed items

Parameters: **parent** – Frame of the parent Frame

Returns: currently made Frame

**make\_initial\_window(*parent: Frame*, *\*args*)→ Frame**[**[source]**](http://localhost:63342/main.py/project/docs/_build/html/_modules/view.html#View.make_initial_window)

make initial settings frame where user can set or change fridge’s name and add, update or remove users and all needed items to display

Parameters: **parent** – Frame of the parent Frame

Returns: currently made Frame

**make\_letter\_buttons(*parent: Frame*)→ None**[**[source]**](http://localhost:63342/main.py/project/docs/_build/html/_modules/view.html#View.make_letter_buttons)[**ℑ**](http://localhost:63342/main.py/project/docs/_build/html/view.html#view.View.make_letter_buttons)

make keyboard buttons in specific frame and every button returns letter of calls function :param parent: Frame where the keyboard to be displayed in

**make\_menu\_buttons(*parent: Frame*, *colspan: int*, *\*name\_sub*)→ None**[**[source]**](http://localhost:63342/main.py/project/docs/_build/html/_modules/view.html#View.make_menu_buttons)

make menu buttons for most of the frames and place them, buttons are: add, cook, back

:param parent: parent Frame where current frame to be places

:param colspan: int number of columns to span

:param name\_sub: optional, name of frame

**make\_qr\_shopping\_list\_window(*parent: Frame*)→ Frame**[**[source]**](http://localhost:63342/main.py/project/docs/_build/html/_modules/view.html#View.make_qr_shopping_list_window)

make qr shopping list view where user can get shopping list and send it and cook and displays all items needed

Parameters: **parent** – Frame of the parent Frame

Returns: currently made Frame

**make\_send\_to\_users\_btns()→ None**[**[source]**](http://localhost:63342/main.py/project/docs/_build/html/_modules/view.html#View.make_send_to_users_btns)

make and display send shopping list buttons for all the users in DB with their username and one for another user not in DB

**make\_user\_field\_lines(*index: int*)→ tuple**[**[source]**](http://localhost:63342/main.py/project/docs/_build/html/_modules/view.html#View.make_user_field_lines)

make all needed fields to be displayed for user to be created and buttons to be updated or deleted

:param index: int place of the user in the view

:return: tuple with all of users fields that have to be displayed

**name\_buttons()→ list**[**[source]**](http://localhost:63342/main.py/project/docs/_build/html/_modules/view.html#View.name_buttons)[**ℑ**](http://localhost:63342/main.py/project/docs/_build/html/view.html#view.View.name_buttons)

only :returns list with names of all users’ fields used for proper user manipulation

**new\_date(*\*args*)→ str[[source]](http://localhost:63342/main.py/project/docs/_build/html/_modules/view.html" \l "View.new_date)**

make new date form current date and timedelta if not args or if args makes time delta from items expiry date

:param args of date as string, optional

:return: if args presented returns timedelta, else returns date as string

**on\_start()→ None**[**[source]**](http://localhost:63342/main.py/project/docs/_build/html/_modules/view.html#View.on_start)

if user didn’t set fridge’s name displays settings view, else displays welcome screen for 5 sec then ask frame

**raise\_above\_all(*win1: Frame*, *action: str*)→ None**[**[source]**](http://localhost:63342/main.py/project/docs/_build/html/_modules/view.html#View.raise_above_all)

display selected frame or if setting button pressed display settings window, clear specific fields according to keyword and if selected frame is the initial frame clears frame stack

:param win1: view frame to be lifted :

param action: keyword that can modify lifting behaviour

**recipe\_find(*d: Event*)→ None**[**[source]**](http://localhost:63342/main.py/project/docs/_build/html/_modules/view.html#View.recipe_find)

select recipe name from list and then search that recipe in recipes from chosen products, random recipes or favourites according to list of recipes displayed, in view displays image, ingredients, instructions and name of the chosen recipe, if any

:param d: Event that triggers selection of recipe

**selected\_item\_str(*a: Event*)→ None**[**[source]**](http://localhost:63342/main.py/project/docs/_build/html/_modules/view.html#View.selected_item_str)

select item form fridge and return it into item fields to be removed or modified, if expired, add, update, clear buttons are disabled and only delete is active to prompts user to delete item and remove it from fridge, spinbox for units is modified according to item’s unit

Parameters: **a** – Event that triggers selection of item

**set\_choices(*value: list*)→ None**[**[source]**](http://localhost:63342/main.py/project/docs/_build/html/_modules/view.html#View.set_choices)

set chosen products from list into items list box view

:param value: list of chosen products to be displayed into items list box view

**set\_chosen(*value: list*)→ None**[**[source]**](http://localhost:63342/main.py/project/docs/_build/html/_modules/view.html#View.set_chosen)

display all chosen products in products list box view to cook for

:param value: list of chosen products to be displayed

**set\_recipes(*value: list*)→ None**[**[source]**](http://localhost:63342/main.py/project/docs/_build/html/_modules/view.html#View.set_recipes)

set chosen recipes from a list into recipe list box view

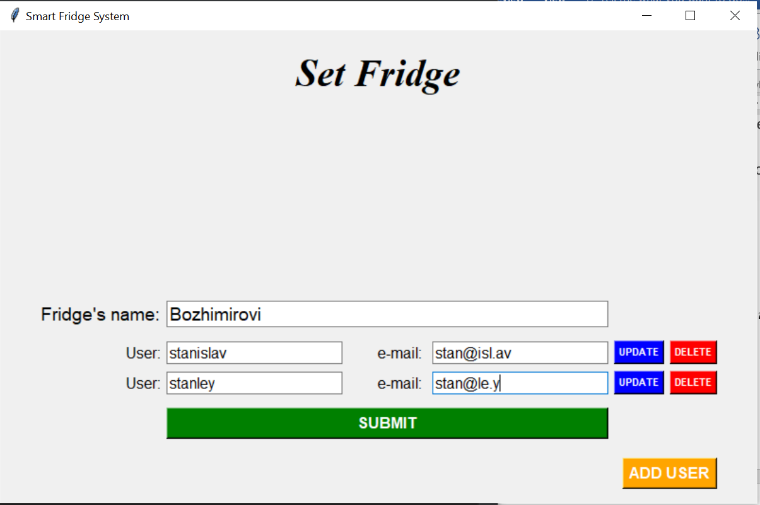
:param value: list of chosen recipes to be displayed

**set\_values(*\*args:***[***Item***](http://localhost:63342/main.py/project/docs/_build/html/model.html#model.Item)**)→ None**[**[source]**](http://localhost:63342/main.py/project/docs/_build/html/_modules/view.html#View.set_values)

set initial values in item fields if args not presented, else set values from args in item fields

:param args: list if args set values of item with values of the selected item placed in args

## Usage of application

*On start there will be initial setup window so user can set fridge name and set users that will use the fridge with usernames and emails.*

*On submit fridge’s name is saved and users are added to the DB and welcome screen appears with fridge’s name. On the upper right corner of the screen a settings icon appeared and if it is clicked the user is returned on setup page where can change fridge name, add, update or remove users*



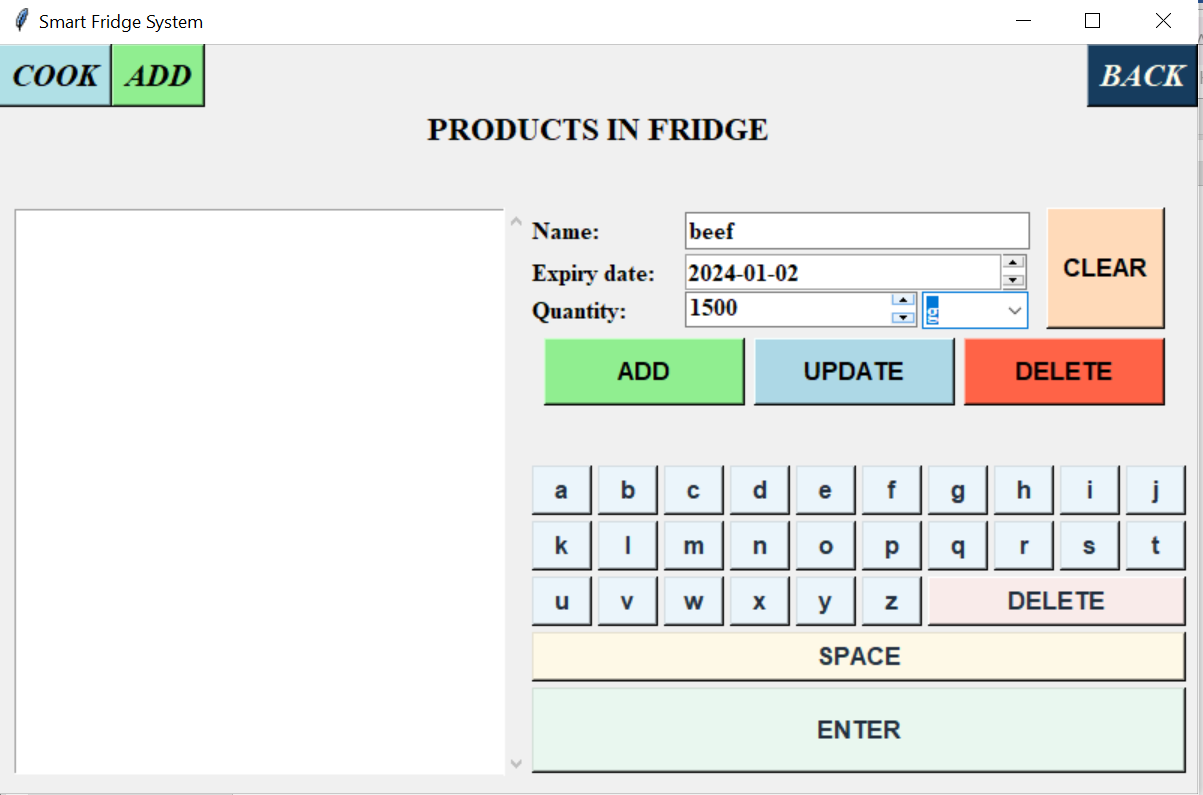
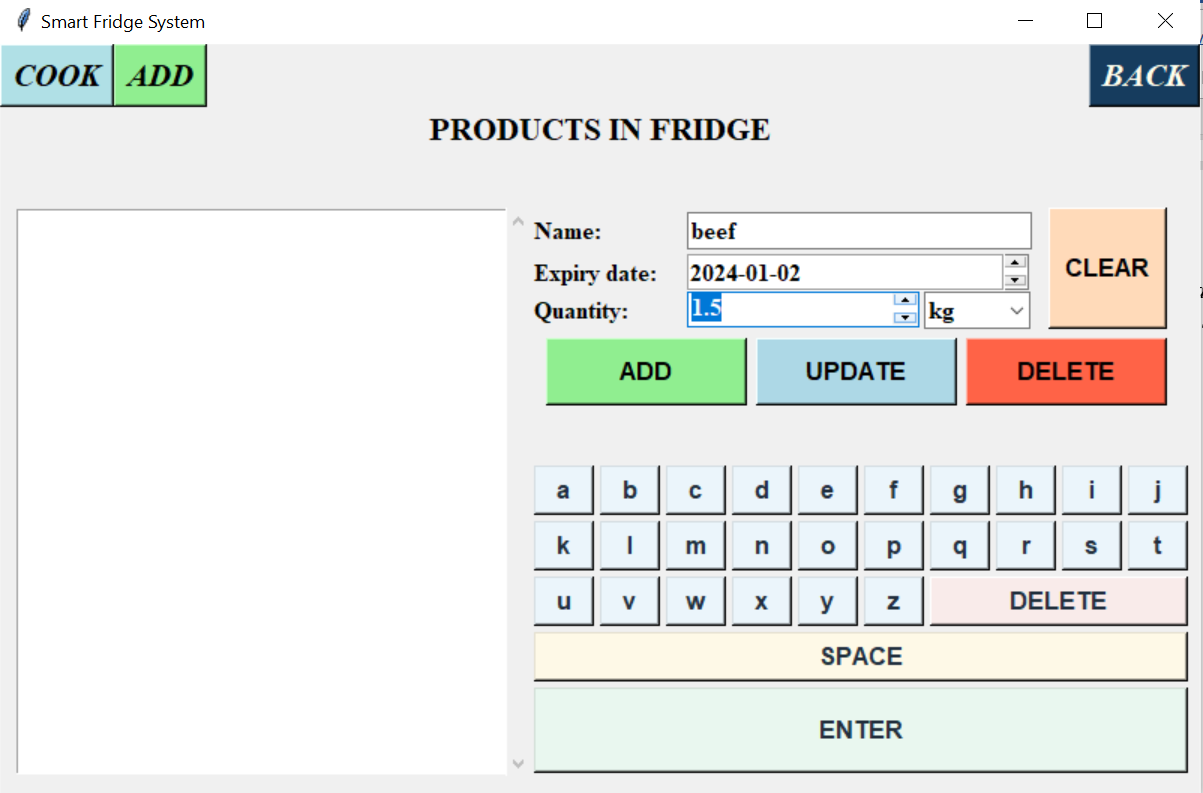
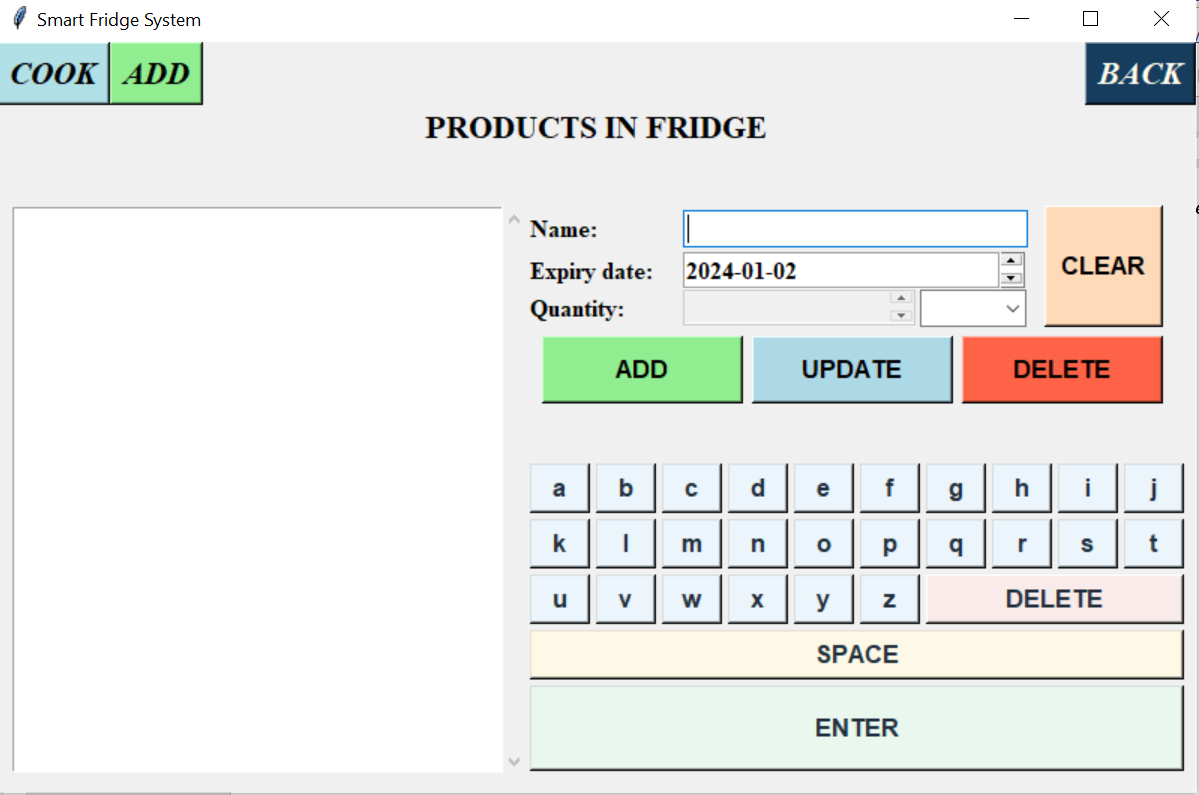
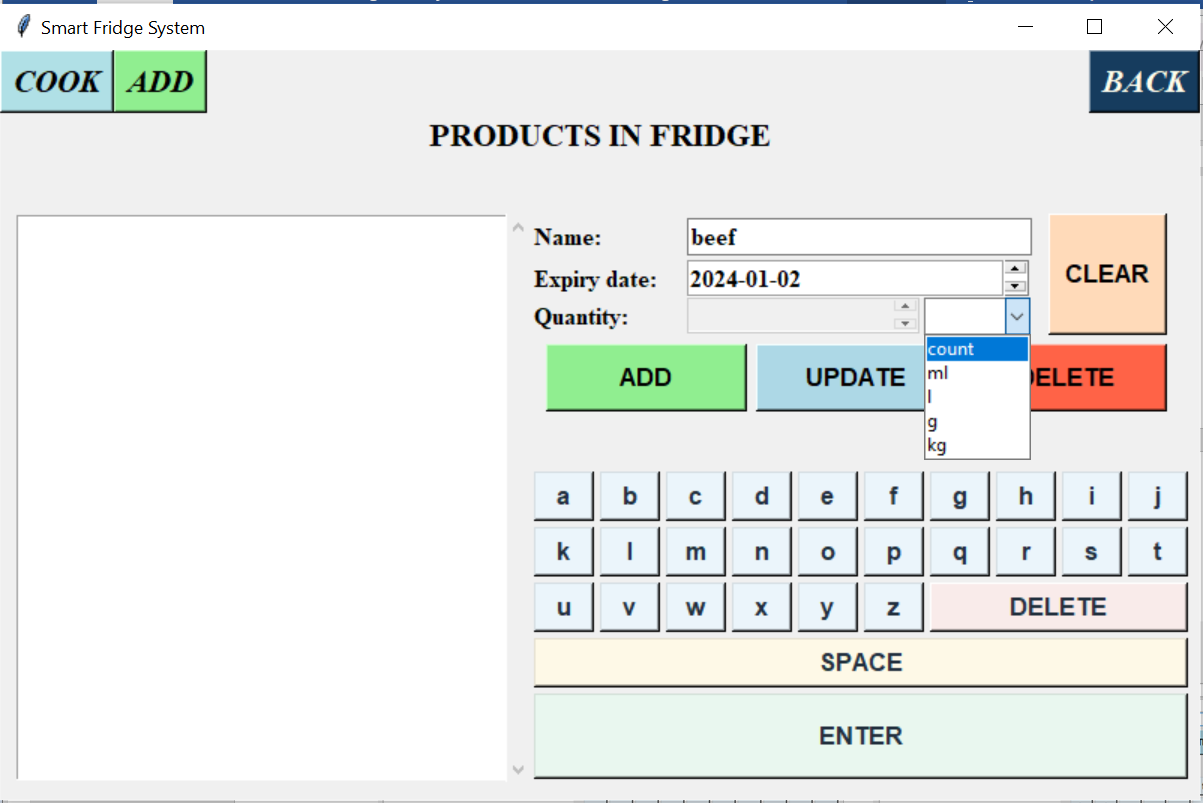
*If this gear icon is not pressed user is moved to choose screen. Here the user can choose to add product or to cook a meal.*



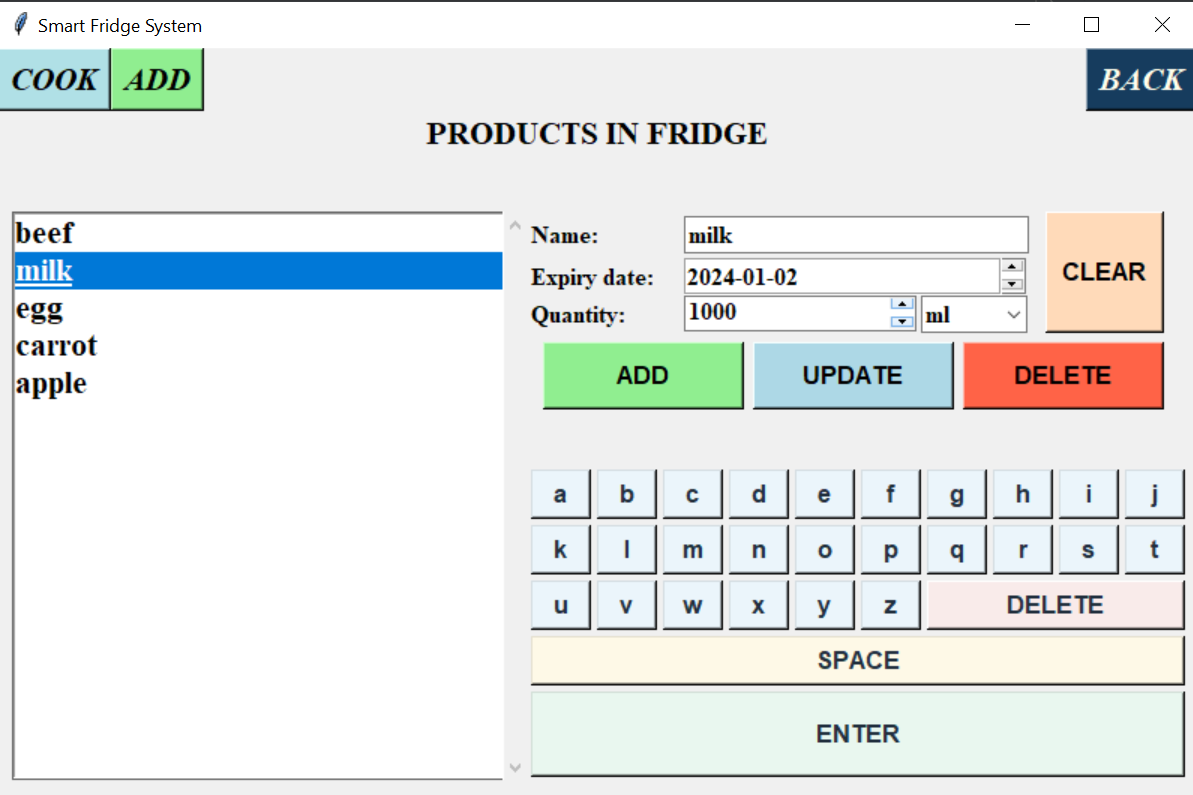
*When add product button is clicked the user is moved to a screen where he can choose a food category of the product he wants to add.*



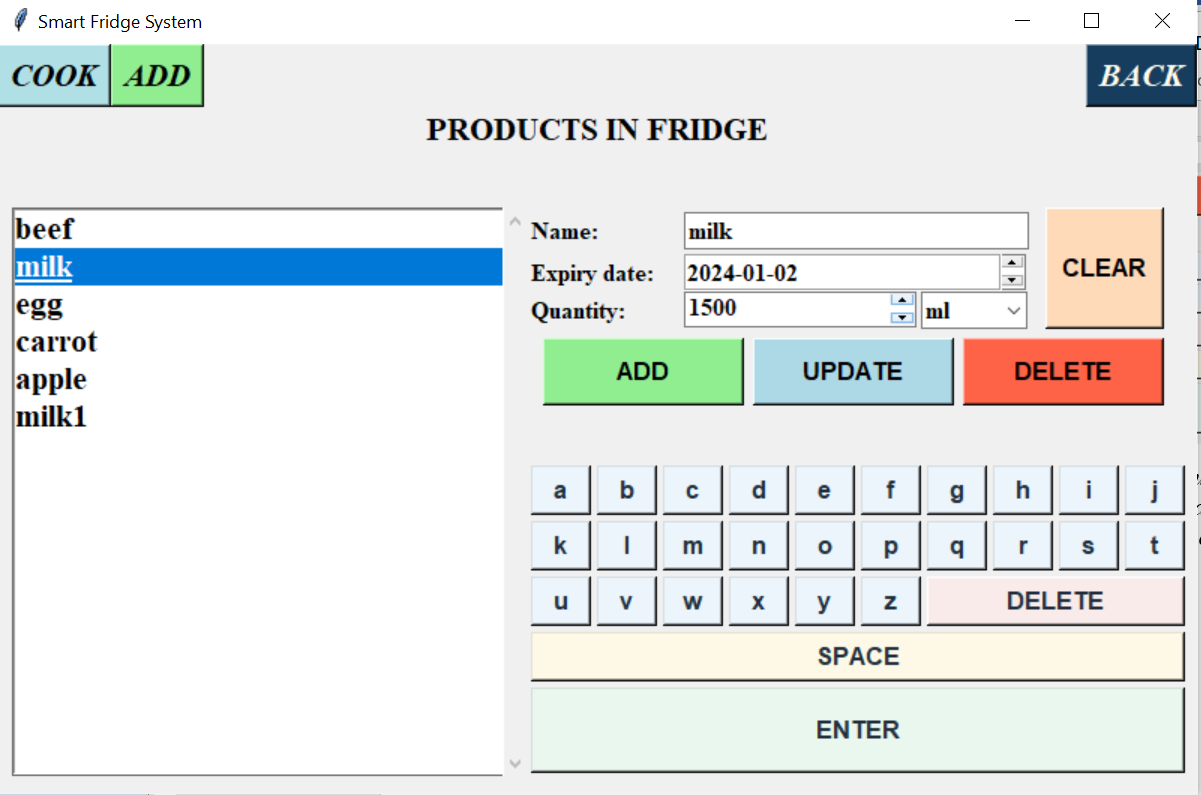
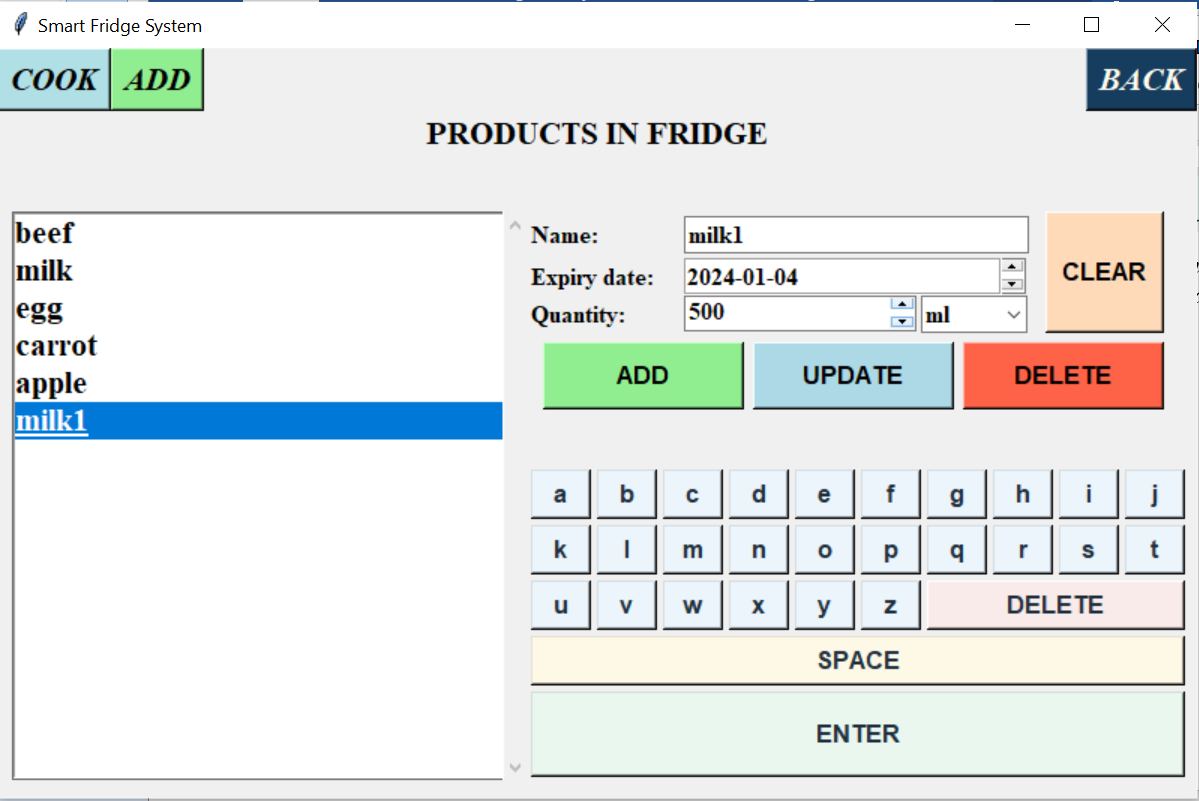
*After choosing category, the user is moved to add product screen. Here the user can add product’s name to choose expiry date (default expiry date is tomorrow), next he have to choose unit of the product from a dropdown menu and for each unit the step of the scrollbar for quantity is different*



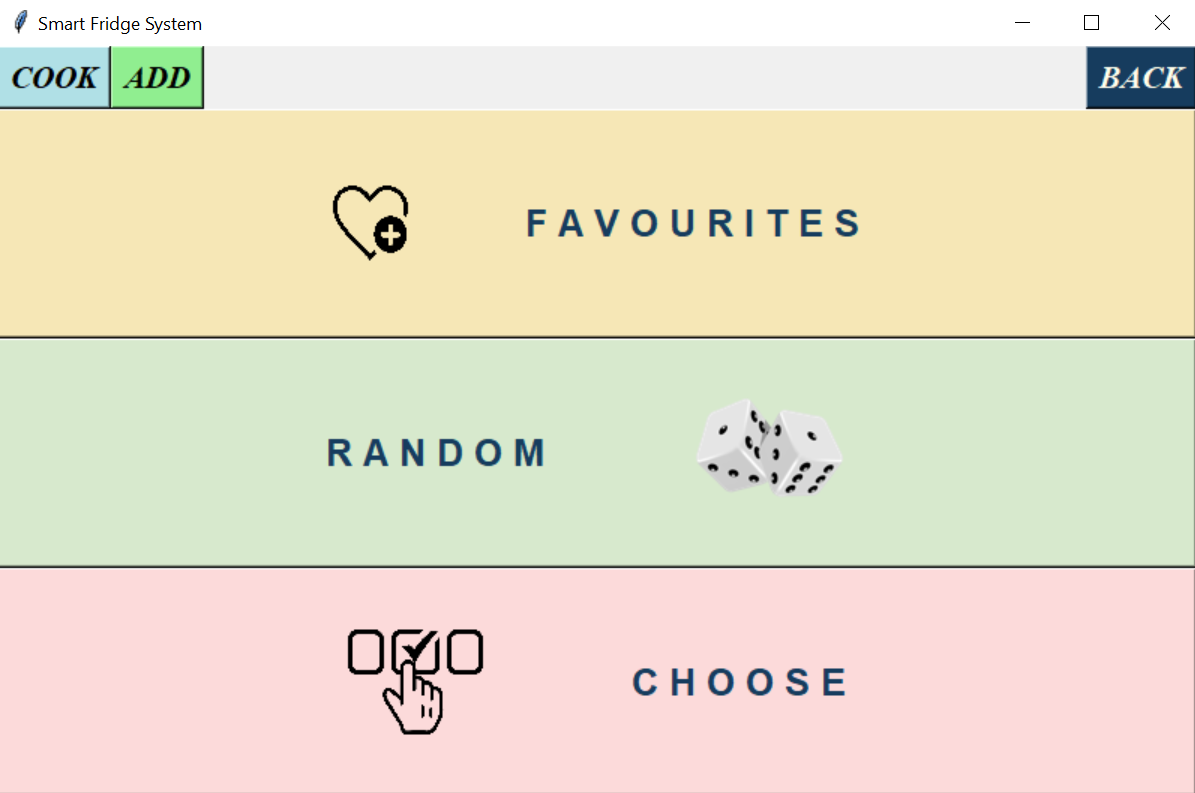
*Here users can add, update and delete products from fridge, also to clear the fields. Products in fridge are shown on the left side of the screen. If a product is chosen, all of the information is inserted into the fields and it can be updated or deleted.*



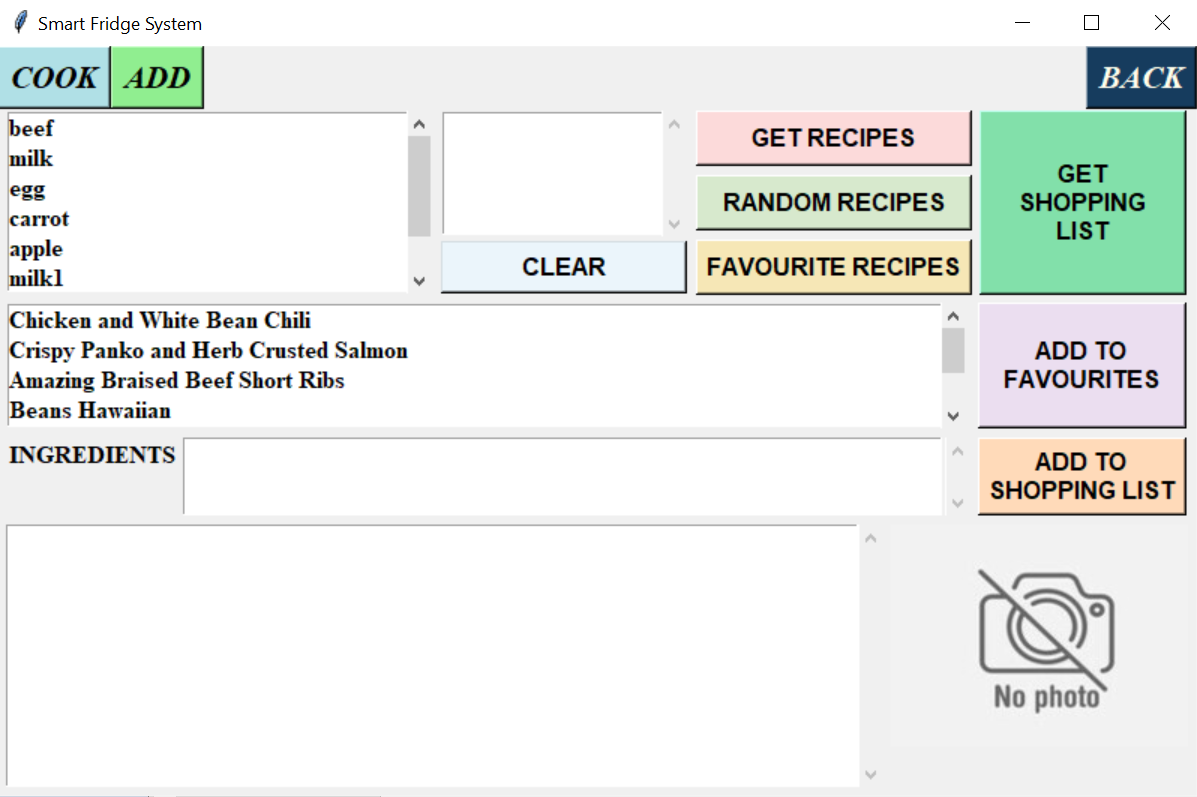
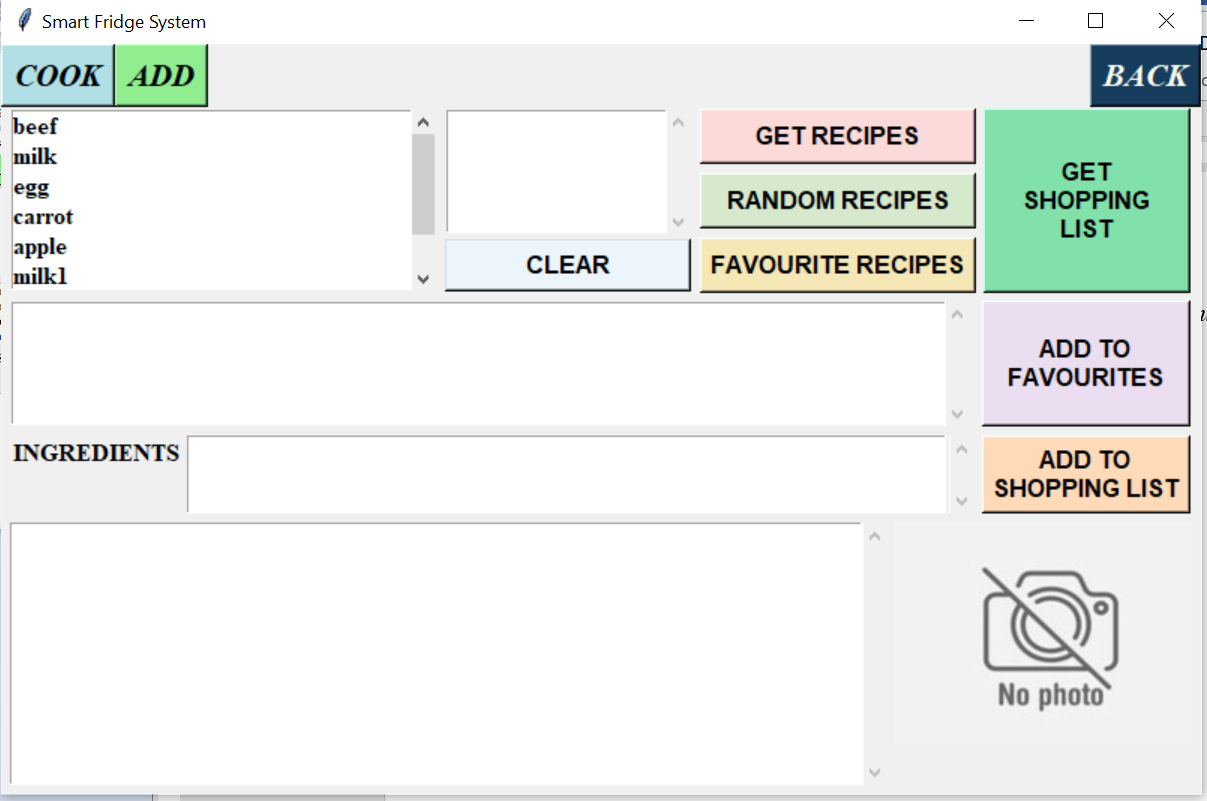
*If product’s quantity is updated the product is updated with the new quantity, but if the unit or expiry date is updated, the fridge make a new instance of the product as it recognize it as a different entity and that’s why an index appear at the end of the name of the product.*



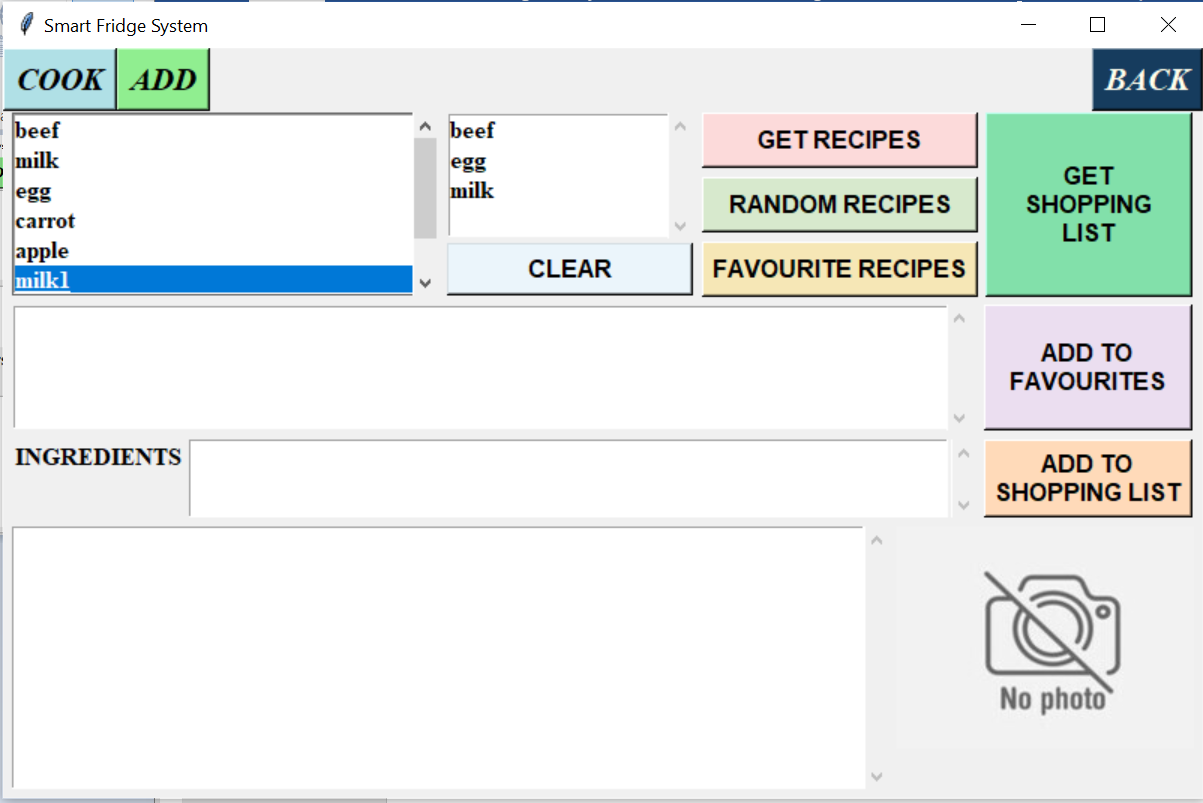
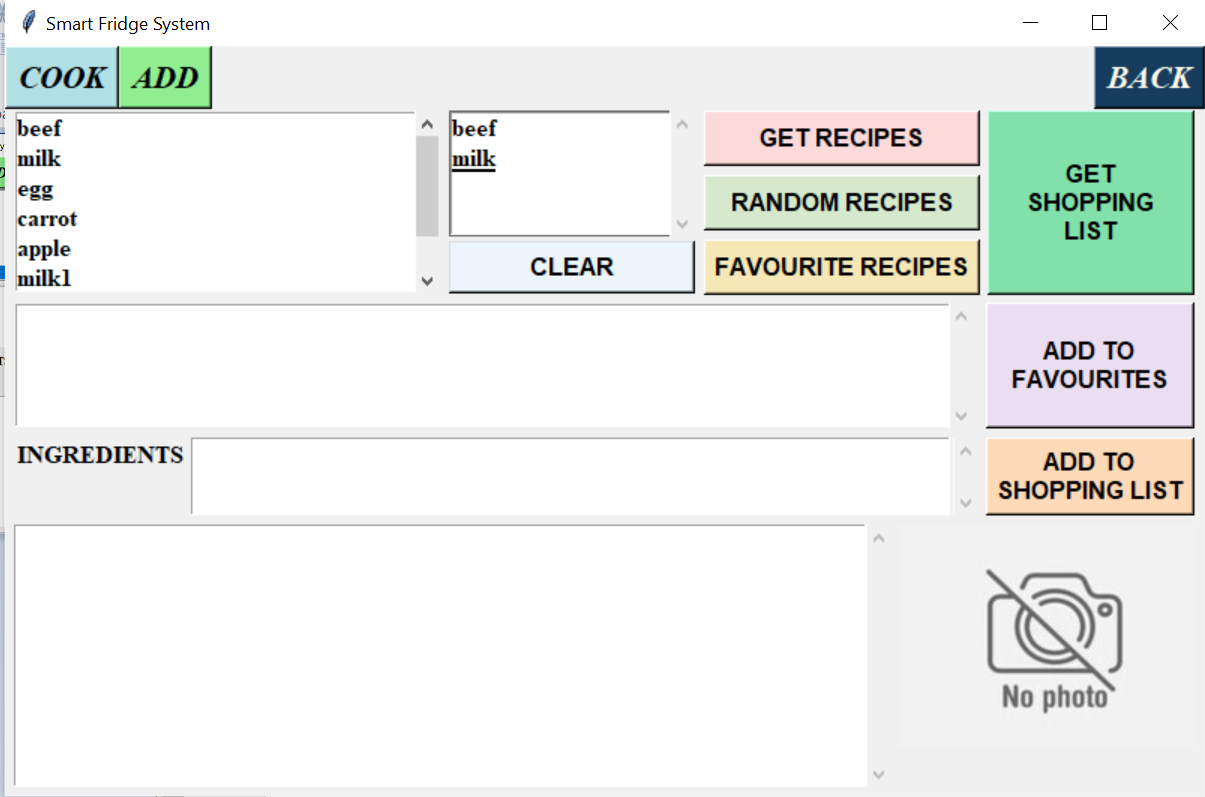
*After finish with adding products to the fridge and pressing cook button on the menu or if on the choose screen the user press cook recipe button, he is moved to choose cook screen where the user have to choose if he wants to look through favourite recipes (if there are any saved in the fridge), to choose from random recipes or to generate recipes by choosing products from the fridge. All of the buttons lead to a screen where the difference is if the recipe list is filled with random recipes, favourite recipes or empty waiting for user to choose products from fridge.*



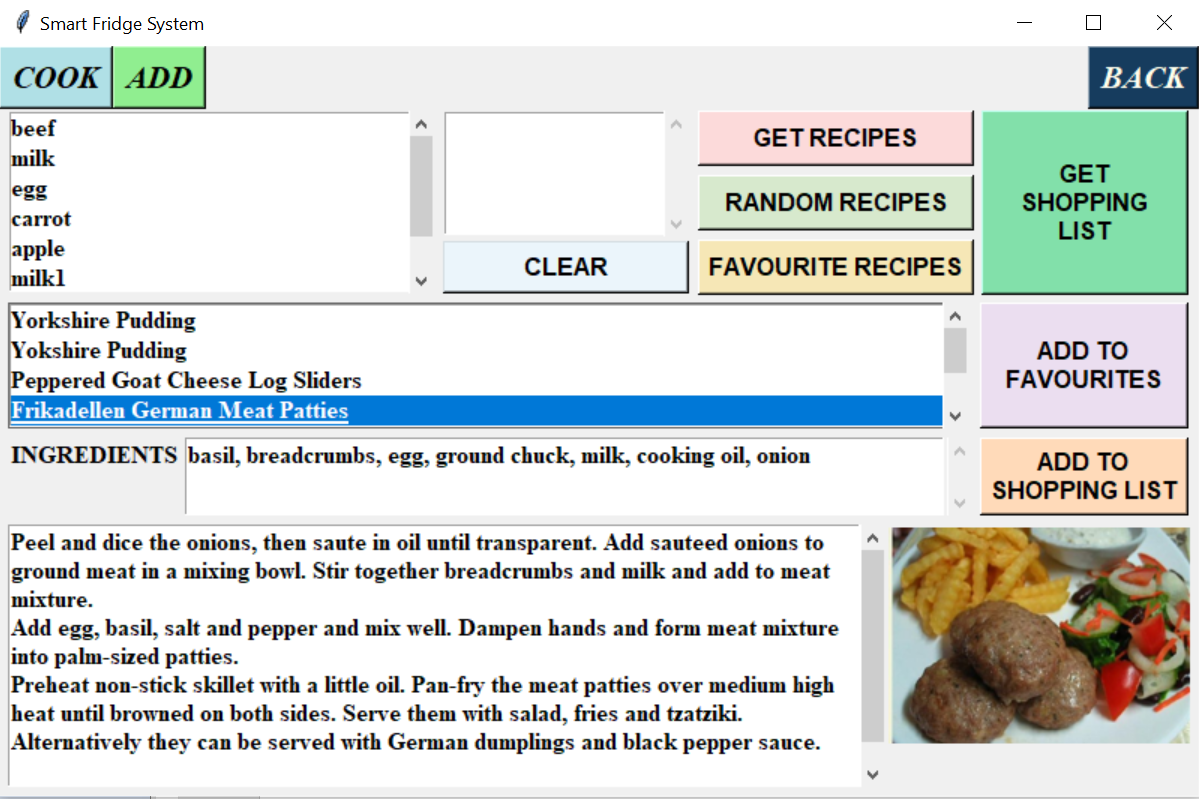
*If the user click on the random or favourite button the next screen should look like the left picture, if click on the choose button the next screen should look like the right picture.*



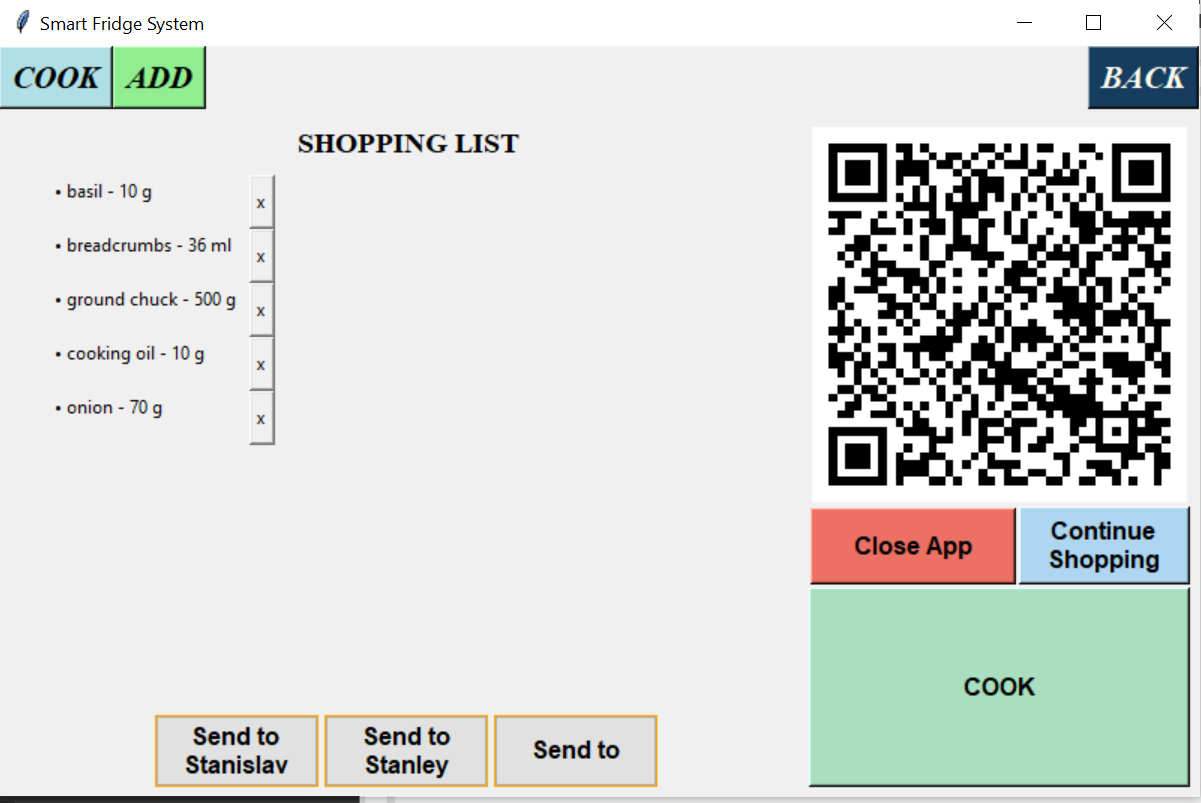
*By clicking on the products the user add chosen products to middle section, products with and without indexes are not repeated. If user want to remove already chosen product he have to click on it or he can clear the whole section by clicking on the clear button.*



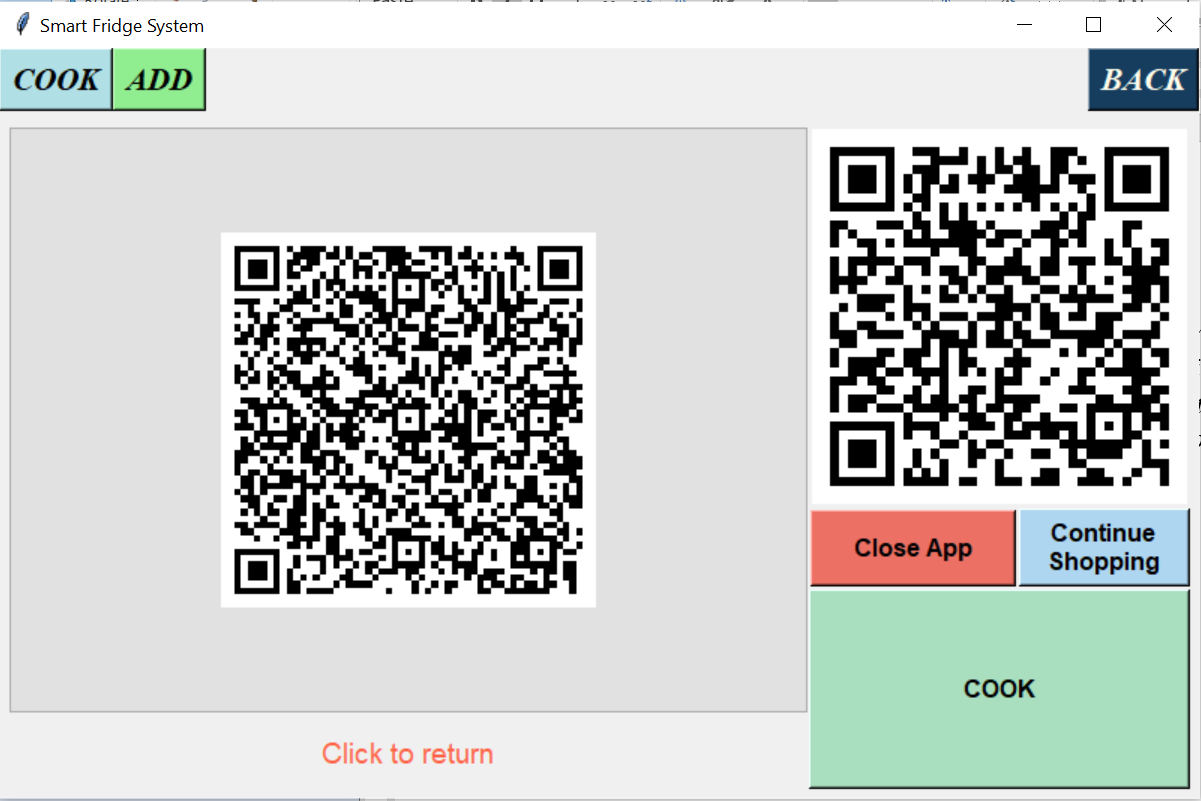
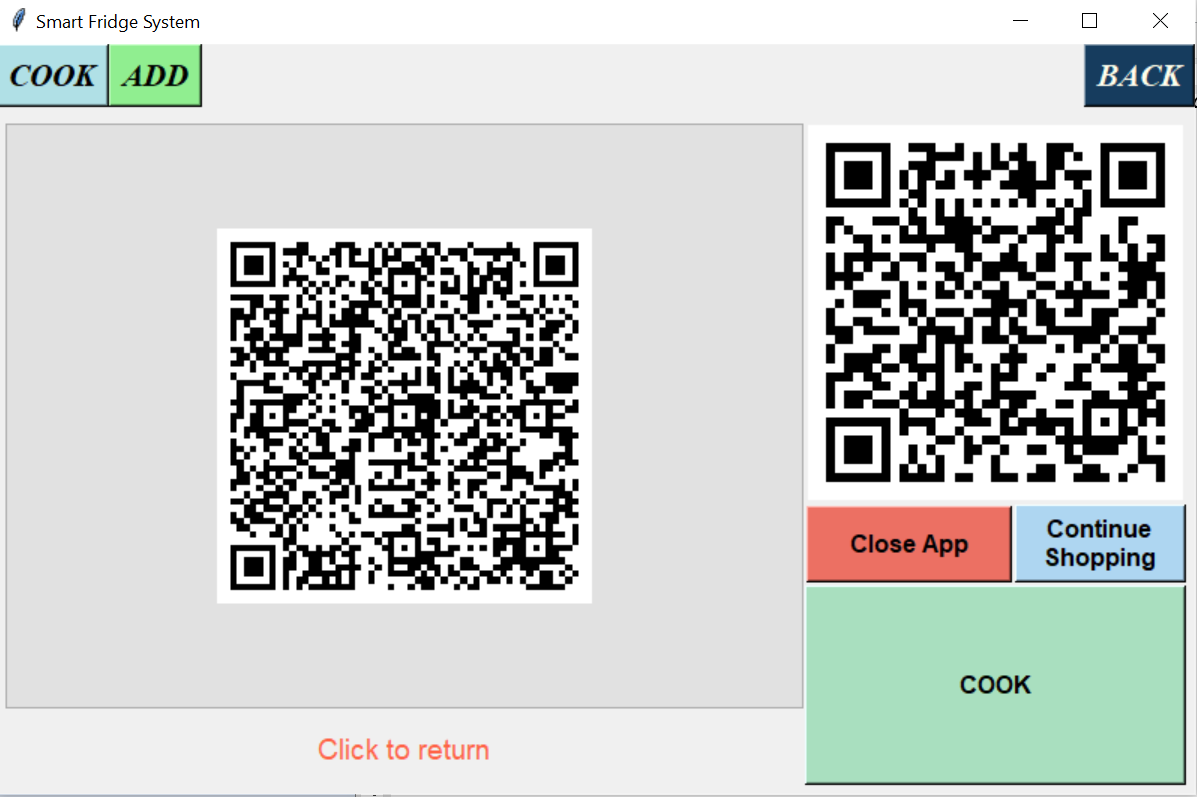
*Then the user have to click on the GET RECIPES button to get the recipes with chosen products from the API. Here he can click and RANDOM RECIPES or FAVOURITE RECIPES also. List of recipes are shown in the second row and when name of a recipe is clicked the user can see the ingredients needed for preparation, instruction for preparation and picture of the meal. The user can keep choosing and changing products of getting random recipes while he choose what to cook. There is a purple button where user can put the recipe in favourite recipes. When the choice is made, the user click on ADD TO SHOPPING LIST button. And the products that are in the fridge and in enough amount are excluded from the shopping list and the product in the fridge is updated with the new decreased amount.*



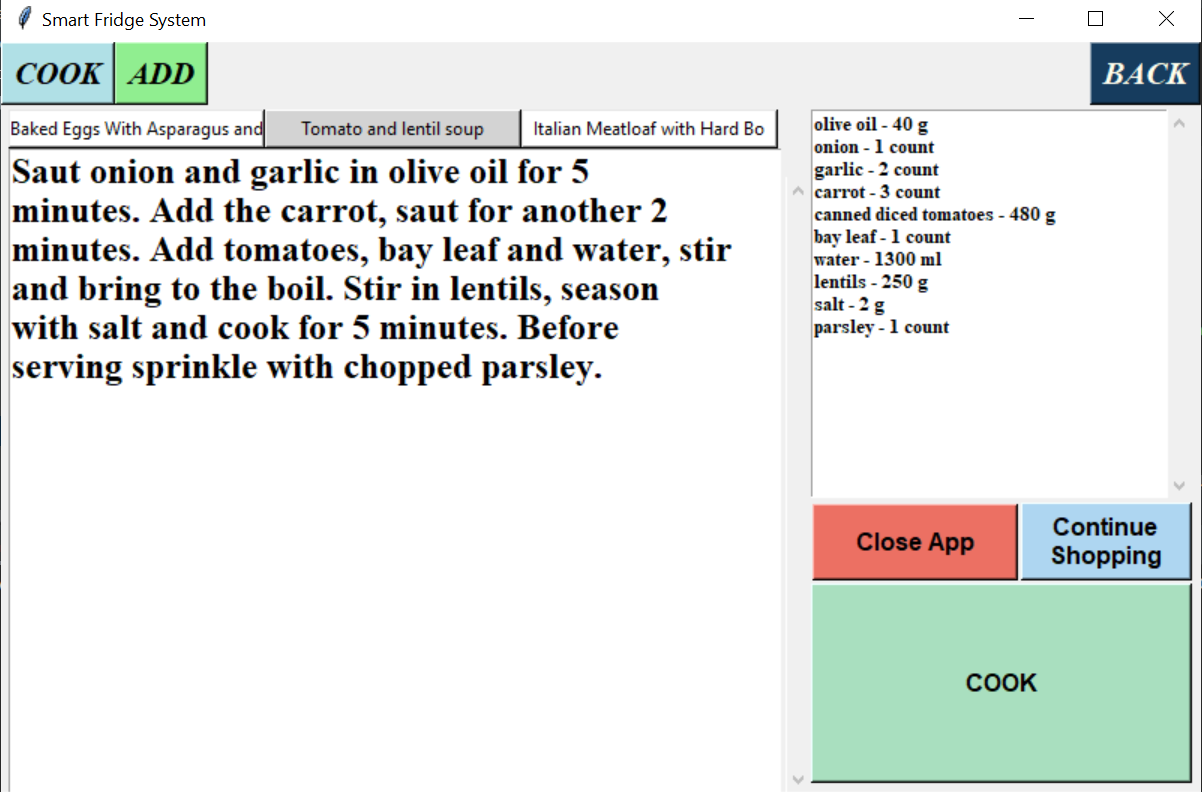
*The next screen shows a shopping list with products that are not in the fridge, there is a little delete button after each ingredient (in case that not all products are in the fridge but are in the garden for example or you will get it from your neighbour). The QR code on the right will show the shopping list on the users phone and will be modified with every change in the shopping list. Also if the user wants to continue with the shopping list there is a button that will return him to the previous page and he can choose another recipe to make. The shopping list can be send through email to registeres users or to another user. There are buttons on the bottom of the page.*



*If the user click on the Send to {username} button QR code appears on screen and when the code is scanned the phone prepares a mail with shopping list to the mail of the registered user(the left picture). If the user click on the Send to button, the mail prepared will have only title and body but the user will have to write manually email of the person that have to make the purchase (the right picture).*

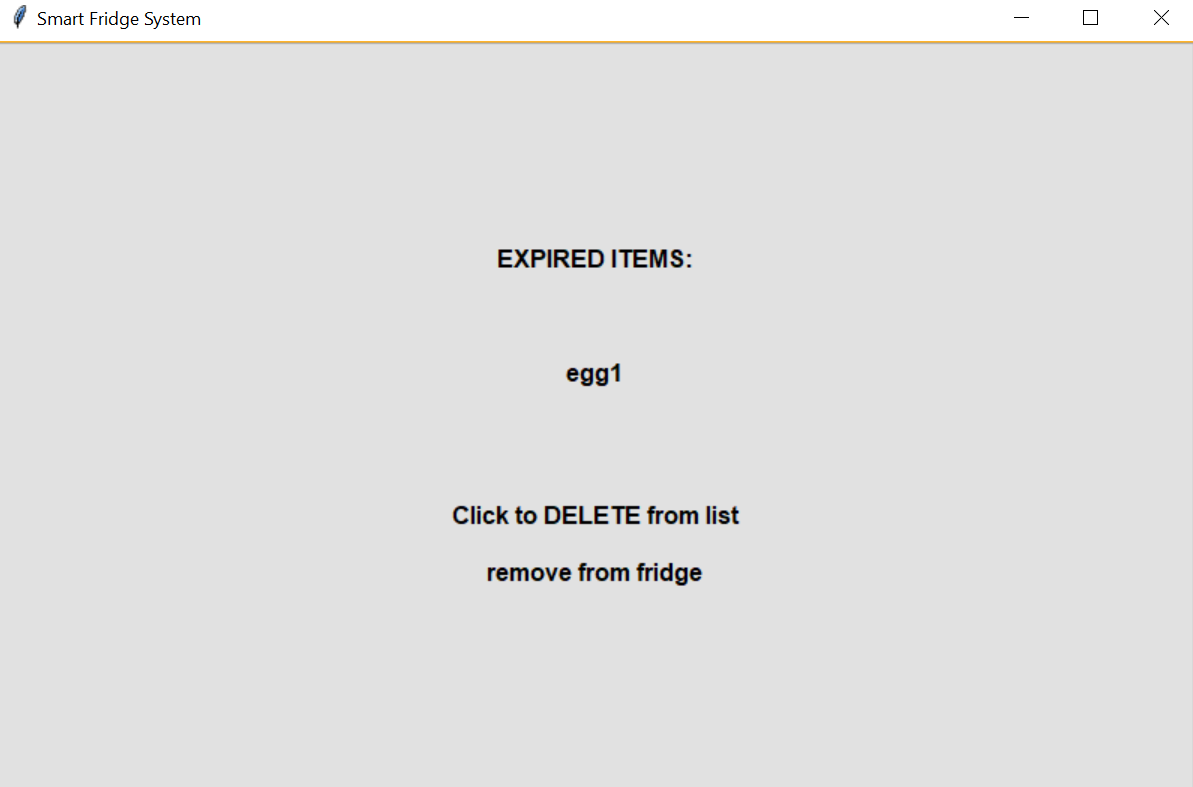


*If the user click on the COOK button the instructions of the recipe and all of the ingredients appear on the screen. When clicking on continue shopping button user is returned to the screen where he can choose other recipe and add it to the shopping list. Ingredients needed are summed and the user can choose which recipe to display on screen.*

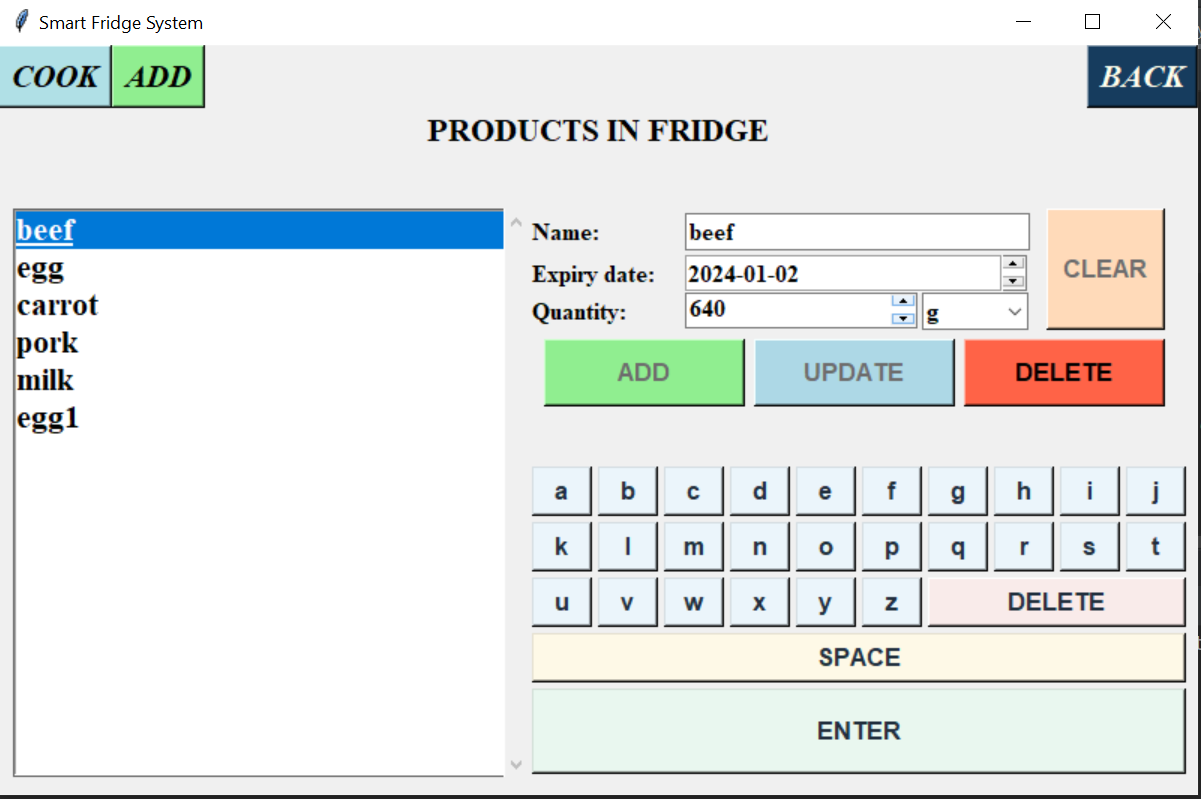


*By clicking on the Close App button the app closes. Next time when the user wants to use the app it have to be started again.*

*On opening the app if there is an expired product the next screen is shown to the user before all other screens. The only option for the user is to delete product and remove it from the fridge.*



*If a product’s expiry date is today it cannot be updated but only deleted, the buttons clear, add and update are disabled.*



<https://github.com/bozhimirov/Smart-Fridge-System>

[Back to top](#back_to_top)