Welcome to Sandwich shop 710069439’s documentation!

Indices and tables

- Index
- Module Index
- Search Page

Download and setup instructions

Installing and running sandwich_shop app:

- Download zip file
- Extract all files into an accesible file location
- **On your command line interface run:**

```
python -m pip install -e C:/Users/YOUR_DOWNLOAD_LOCATION/sandwich_shop
```

- **To launch the app run:**

```
python C:/Users/YOUR_DOWNLOAD_LOCATION/sandwich_shop/code/main.py
```

Running the app from within the package:

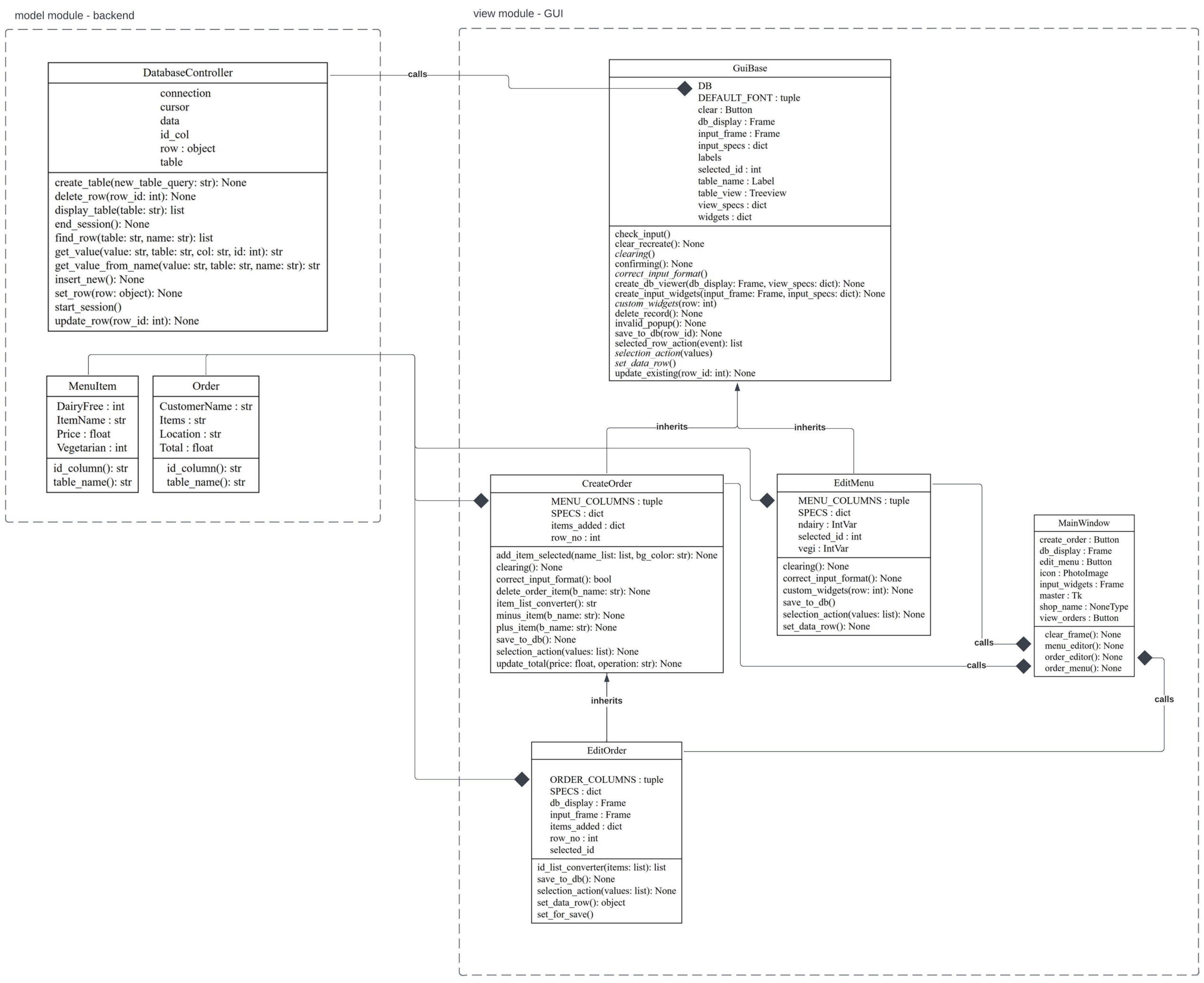
- Open the sandwich_shop folder in an IDE such as Visual Studio Code from the root directory ‘sandwich_shop’
- Run the terminal and check that the file path ends with ‘sandwich_shop’
- Install the package by running the command `python -m pip install -e .`
- **To launch the app run:**

```
python ../sandwich_shop/code/main.py
```

- OR go to sub-directory ‘code’, go into the module main.py, run the file
- OR in a python file, add the following lines and run the file:

```
from sandwich_shop.code import main
main.run()
```

UML Class Diagram



Python Module Index

c | m | v

code

code package

Subpackages

Submodules

code.main module

code.model package

Submodules

code.model.data module

Defines dataclasses in line with the database setup

Sources:

https://docs.python.org/3/library/dataclasses.html

https://www.youtube.com/watch?v=CvQ7e6yUtnw

class `model.data.MenuItem`*(ItemName: str, Price: float, Vegetarian: int = 0, DairyFree: int = 0)*

Bases: `object`

Creates dataclass instance containing all data for a row in the Menu table in database

Parameters:
- **ItemName** (*str*) – name of sandwich
- **Price** (*float*) – price in £
- **Vegetarian** (*int*) – binary value indicator, 1 means yes. Defaults to 0
- **DairyFree** (*int*) – binary value indicator, 1 means yes. Defaults to 0

Returns: MenuItem(ItemName='', Price=0., Vegetarian=0, DairyFree=0)

`DairyFree`: *int* = *0*

`ItemName`: *str*

`Price`: *float*

`Vegetarian`: *int* = *0*

static `id_column`() → str

Gets name of ID column in database

Returns: ID column name

Return type: str

static `table_name()`→ str

Gets name of associate table in the database

Returns: table name

Return type: str

class `model.data.Order`(*CustomerName: str, Location: str, Items: str, Total: float*)

Bases: `object`

Creates dataclass instance containing all data for a row in Orders table in the database

Parameters:

- **CustomerName** (*str*) –
- **Location** (*str*) – what3words address
- **Items** (*str*) – string of list of item IDs as found in menu table
- **Total** (*float*) – total price paid at time of order

Returns: Order(CustomerName='', Location='', Items=[], Total=0.)

`CustomerName`: *str*

`Items`: *str*

`Location`: *str*

`Total`: *float*

static `id_column()`→ str

Gets name of ID column in database

Returns: ID column name

Return type: str

static `table_name()`→ str

Gets name of associate table in the database

Returns: table name

Return type: str

`model.data.validate_types`(*obj: object*)→ None

Gets type hints defined in dataclasses to confirm input

Parameters: **obj** (*dataclass*) – MenuItem or Order

Raises: **TypeError** – if incorrect data type input, else None

code.model.db_controller module

All database manupulation functions

Sources:

https://docs.python.org/3/library/sqlite3.html

https://www.tutorialspoint.com/sqlite/sqlite_python.htm

https://www.geeksforgeeks.org/python-sqlite/

class `model.db_controller.DatabaseController`(*row: object | None = None*)

Bases: `object`

`create_table`(*new_table_query: str*)→ None

Create new sqlite table

Parameters: **new_table_query** (*str*) – SQLite query for creating new table

`delete_row`(*row_id: int*)→ None

Delete row from table where Name column matches

Parameters: **row_id** (*int*) – ID number of row to be updated

`display_table`(*table: str | None = None*)→ list

Returns all rows in a table based on input or table associated with set data row

`end_session`()→ None

Commit changes and close connection

`find_row`(*table: str*, *name: str*)→ list

Finds row od data in a given table based on Name column

Parameters:

- **table** (*str*) – name of table to query from
- **name** (*str*) – name of item to find in the NAme column of the table

Returns: list of tuples containing data in each row

Return type: list

`get_value`(*value: str*, *table: str*, *col: str*, *id: int*)→ str

Finds data in row in given table by ID of the item

Parameters:
- **value** (*str*) – column name to get value from
- **table** (*str*) – table name
- **col** (*str*) – name of ID column to use for search
- **id** (*str*) – ID number of the row

Returns: value from the specified column of the found row

Return type: str

`get_value_from_name`(*value: str*, *table: str*, *name: str*)→ str

Finds row of data in given table by item name

Parameters:
- **value** (*str*) – column name to get value from
- **table** (*str*) – table name
- **name** (*str*) – name of item

Returns: value from the specified column of the found row

Return type: str|int

`insert_new`()→ None

Parses data dictionary into new row in the relevant table

`set_row`(*row: object*)→ None

Takes instance of a dataclass object, parses into dictionary, gets associated table name

Parameters: **row** (*object*) – datalcass object MenuItem or Order

`start_session`()

`update_row`(*row_id: int*)→ None

Updates the row with given id number to data in set data row

Parameters: **row_id** (*int*) – ID number of row to be updated

code.model.utils module

Module for external what3words api call functions https://developer.what3words.com/public-api/docs

`model.utils.location_converter`(*lat_long: str*)→ str

Calls api to convert latitude, longutude to what3words

Parameters: **lat_long** (*str*) – two flaot numbers for coordinates

Returns: what3words

Return type: str

`model.utils.what3words_converter`(*words: str*)→ str

Calls api to convert what3words to coordinates

Parameters:	**words** (*str*) – what3words string
Returns:	latitude, logitude converted to string
Return type:	str

Module contents

code.view package

Submodules

code.view.create_order module

All input widgets and functionality for creating order on the UI

class `view.create_order.CreateOrder`(*db_display: Frame, input_frame: Frame*)

Bases: `GuiBase`

`MENU_COLUMNS`= *('Item Name', 'Price(£)', 'Vegetarian', 'Dairy Free')*

`SPECS`
= {'input_widgets': {'buttons': ['Confirm order'], 'input': {'CustomerName': {'label': 'Customer name:', 'type': 'entry'}, 'Items': {'label': 'Items in order:', 'type': 'frame'}, 'Location': {'label': 'Location (lat,long):', 'type': 'entry'}, 'Total': {'label': 'Total £ ', 'type': 'label'}}}, 'view': {'table_name': 'Select items from the menu', 'table_view': {'columns': ('Item Name', 'Price(£)', 'Vegetarian', 'Dairy Free'), 'table': 'menu'}}}

`add_item_selected`(*name_list: list, bg_color: str = 'white'*)→ None

Convert data from selected row into items listed in input widgets

Parameters:
- **name_list** (*list*) – data points containing item name, quantity, price
- **bg_color** (*str, optional*) – background color of component. Defaults to “white”.

`clearing`()→ None

Set all input widgets to empty values

`correct_input_format`()→ bool

Checks format of input values

Returns: True if all conditions are correct, else False

Return type: bool

`delete_order_item`(*b_name: str*)→ None

Destroys row of labels for a given item name

Parameters: **b_name** (*str*) – Item name

`item_list_converter`()→ str

Converts list of names of sandwiches added to order to list of associated ids

Returns: string of item ids list

Return type: str

`minus_item`(*b_name: str*)→ None

Adds the minus button to decrease quantity of item added

Parameters: **b_name** (*str*) – Item name in selected row from table

`plus_item`(*b_name: str*)→ None

Adds the plus button to increase quantity of item added

Parameters: **b_name** (*str*) – Item name in selected row from table

`save_to_db`()→ None

Convert inputs to dataclass object and insert into database

`selection_action`(*values: list*)→ None

Maps values from selected row to items added widgets

Parameters: **values** (*list*) – row of selected menu item data

`update_total`(*price: float*, *operation: str = '+'*)→ None

Adds or subtracts from total based on plus or minus button click

Parameters:
- **price** (*float*) – unit price of a menu item
- **operation** (*str, optional*) –
  - or - operation selection.
- "+". (*Defaults to*) –

code.view.edit_menu module

User interface and functionality for editing Menu saved in the database

class `view.edit_menu.EditMenu`(*db_display: Frame*, *input_frame: Frame*)

Bases: `GuiBase`

`MENU_COLUMNS`= *('Item Name', 'Price(£)', 'Vegetarian', 'Dairy Free')*

SPECS
= {'input_widgets': {'buttons': ['Add/Update', 'Delete record'], 'input': {'DairyFree': {'label': 'DairyFree:', 'type': 'checkbox'}, 'ItemName': {'label': 'Item Name:', 'type': 'entry'}, 'Price': {'label': 'Price £:', 'type': 'entry'}, 'Vegetarian': {'label': 'Vegetarian:', 'type': 'checkbox'}}}, 'view': {'table_name': 'Select menu item to edit', 'table_view': {'columns': ('Item Name', 'Price(£)', 'Vegetarian', 'Dairy Free'), 'table': 'menu'}}}

`clearing`()→ None

Sets input widgets to empty values

`correct_input_format`()→ None

Checks correct format of input values

Returns:	True if all formats are correct, else False
Return type:	bool

`custom_widgets`(*row: int*)→ None

Adds custom widgets to those defined in base class GuiBase

Parameters: **row** (*int*) – row number for placement in UI grid

`save_to_db`()

Updates data or saves to database, updates database viewer

Parameters: **row_id** (*int*) – id number of row in the database. Defaults to 0.

`selection_action`(*values: list*)→ None

Parses selected row in table view to populate in input widgets

Parameters: **values** (*list*) – values from the selected row

`set_data_row`()→ None

Turns input data into dataclass if datatypes are as expected

Returns:	dataclass from input values
Return type:	MenuItem

code.view.edit_order module

Widgets and functionality for editing recorded orders

class `view.edit_order.EditOrder`(*db_display: Frame, input_frame: Frame*)

Bases: `CreateOrder`

ORDER_COLUMNS= *('Customer Name', 'Location', 'Items', 'Total(£)')*

`SPECS` *= {'input_widgets': {'buttons': ['Update', 'Delete record'], 'input': {'CustomerName': {'label': 'Customer name:', 'type': 'entry'}, 'Items': {'label': 'Items in order:', 'type': 'frame'}, 'Location': {'label': 'Location (lat,long):', 'type': 'entry'}, 'Total': {'label': 'Total £ (when ordered): ', 'type': 'label'}}}, 'view': {'table_name': 'Select order to edit', 'table_view': {'columns': ('Customer Name', 'Location', 'Items', 'Total(£)'), 'table': 'orders'}}}*

`id_list_converter`(*items: list*)→ list

Converststs list of item ids to names for display

Parameters:	**items** (*list*) – ids of items in the order
Returns:	names of ordered items
Return type:	list

`save_to_db`()→ None

Updates data or saves to database, updates database viewer

`selection_action`(*values: list*)→ None

Maps values to widgets, re-creates ordered items list

Parameters:	**values** (*list*) – row of selected data

`set_data_row`()→ object

Maps data in input widgets to Order dataclass

Returns:	Order dataclass
Return type:	object

`set_for_save`()

code.view.gui_base module

Base class for view, edit, create user interfaces and functionalities

class `view.gui_base.GuiBase`(*db_display: Frame, input_frame: Frame, specs: dict*)

Bases: `object`

`DEFAULT_FONT`*= ('OpenSans', 12)*

`check_input`()

Checks correct input format as implemented by each child class

Returns:	True if all input values are in expected format and datatype, else False
Return type:	bool|None

clear_recreate()→ None

Destroys database viewer and recreates it so that any changes are reflected

abstract **clearing**()

Clears all input widgets

confirming()→ None

Saves to database, creates pop-up message confirming completion for correctly formatted input, clears all input widgets.

abstract **correct_input_format**()

Child class specific input checker

create_db_viewer(*db_display: Frame*, *view_specs: dict*)→ None

Creates Treeview table for viewing database tables

Parameters:
- **db_display** (*Frame*) –
- **view_specs** (*dict*) – SAMPLE_SPECS["view"] view related configurations

create_input_widgets(*input_frame: Frame*, *input_specs: dict*)→ None

Creates all input widgets

Parameters:
- **input_frame** (*Frame*) –
- **input_specs** (*dict*) – SAMPLE_SPECS["input_widgets"] widget related configurations

custom_widgets(*row: int*)

Additional widgets for child classes to specify

Parameters: **row** (*int*) – row number to place custom widgets on the UI

delete_record()→ None

Deletes record from the database

invalid_popup()→ None

Creates pop-up message box indicating invalid input

save_to_db(*row_id=0*)→ None

Updates data or saves to database, updates database viewer

Parameters: **row_id** (*int*) – id number of row in the database. Defaults to 0.

`selected_row_action`(*event*)→ list

Gets values of the selected row in Treeview, processes custom to each child class

Parameters:	**event** (*click*) – listens for click on any row
Returns:	values of data in the seleced row
Return type:	list

abstract `selection_action`(*values*)

Parses and operates on data from selected row in table view

Parameters:	**values** (*list*|*tuple*) – values from the selected row

`set_data_row`()

Gets data from input widgets into dataclass

`update_existing`(*row_id: int*)→ None

Updates existing record in the database by id

Parameters:	**row_id** (*int*) – id number of row in the database

code.view.main_menu module

Creates app window and the initial view, initiates relevant interfaces

class `view.main_menu.MainWindow`(*master: Tk*)

Bases: `object`

`clear_frame`()→ None

Delete all elements in editor interface frames

`menu_editor`()→ None

Clears frames, recreates interface for editing the menu

`order_editor`()→ None

Clears frames, recreates interface for editing orders

`order_menu`()→ None

Clears frames, recreates interface for creating order

Module contents

Package containing all Tkinter GUI modules

Sources for all modules:

https://www.youtube.com/watch?v=yQSEXcf6s2I&list=PLCC34OHNcOtoC6GglhF3ncJ5rLwQrLGnV&ab_channel=Codemy.com

    # noqa E501

https://github.com/flatplanet/Intro-To-TKinter-Youtube-Course

https://www.geeksforgeeks.org/python-gui-tkinter/

https://realpython.com/python-gui-tkinter/

Index