SYSDEV2 documentation

Release

anonymous

Mar 14, 2023

customer facing add form

class frontend.add_form.**BaseAddForm** (*page: str*, *fields: Tuple[str, str]*, *title: str*, *order_id: str | None = None*)

Bases: object

base form for adding things to the db

__init__ (*page: str*, *fields: Tuple[str, str]*, *title: str*, *order_id: str | None = None*) → None

__weakref__
list of weak references to the object (if defined)

add (*inputs: list*) → None
add the item to backend

Parameters **inputs** (*list*) – inputs from the form

cancel ( ) → None
cancel action to go back to main page, on cancel button

destroy ( )
destroy root error message

destroy_both ( )
destroy error message and input window

fetch (*entries: tuple*) → None
get input text to list to input to backend

Parameters **entries** (*tuple*) – tkinter input fields, .get() transforms to strings

init_ui (*root: Frame*, *fields: tuple*) → tuple
initialise what is shown in the window namely labels and input boxes

Parameters
- **root** (*Frame*) – root frame
- **fields** (*tuple*) – fields to be labled

Returns _description_

Return type tuple

message (*message: str*, *command: object*)
displays message to user

Parameters
- **message** (*str*) – message to be displayed ot the user
- **command** (*object*) – command after pressing the ok button

return_back ( ) → None
go back to main page

base order/order page ui - NOTE: this is hideously long - as there is no way to make it shorter without either tkinter breaking or having lots of duplicate code

class frontend.base_page.**orderListForm** (*page_type: str*)
Bases: object
base order page

__init__ (*page_type: str*) → None
constructure for the order page class,
shows everything seen in the page when it first pops up
understandably its not the most pythonic
but I would rather make the app more usable
as everything can be controled from two pages

__weakref__
list of weak references to the object (if defined)

create_items_tree (*listframe: object*) → Treeview
create list tree full of items :param listframe: frame to put list tree into :type listframe: object

Returns tree view full of items

Return type ttk.Treeview

create_menu_tree (*listframe: object*) → Treeview
create tkinter treeview of menu

Parameters **listframe** (*object*) – tk inter frame to put list tree in

Returns list tree

Return type ttk.Treeview

create_order_tree (*listframe: object*) → Treeview
create list tree for orders :param listframe: frame to put list tree into :type listframe: object

Returns list tree full of orders

Return type ttk.Treeview

delete_menu_item_backend (*item: str*) → None
deete an item from the db

delete_order_backend (*id_value: str*) → None
delete an order from the order DB :param id_value: selected order id from listtree :type id_value: str

populate_items_tree (*order_id: str*) → None
get items from order and populate tree :param order_id: order id to populate :type order_id: string

populate_listree (*listtree: Treeview*) → None
populate a tree view with data :param listtree: tk tree view to populate :type listtree: ttk.Treeview :param page: what type of page its in,
to tell the function what type of data to populate with.

update_menu_item_backend (*item: str*, *new_price: str*) → None
update the price of an item in the backend

update_order_backend (*order_id: int*, *customer: str*, *location: str*) → None
backend methods to update an item in the db :param id: order if :type id: int :param customer: name of customer (unique) :type customer: str :param location: location placed with order (unique) :type location: str

common utils code used in the front end

frontend.front_end_utils.**configure_listree** (*listtree: Treeview*, *listframe: Frame*) → Treeview
configure a tree view for the main pages

Parameters
- **listtree** (*ttk.Treeview*) – tree view to configure
- **listframe** (*Frame*) – frame which tree veiw is placed

Returns configured tree view

Return type ttk.Treeview

frontend.front_end_utils.**create_cmdframe** (*detailsframe: Frame*) → Frame
command frame to put buttons into

Parameters **detailsframe** (*Frame*) – control panel to place into

Returns a command frame

Return type Frame

frontend.front_end_utils.**create_list_frame** (*baseframe: Frame*, *column: int = 0*, *row: int = 1*) → Frame
create a frame for tree views

Parameters
- **baseframe** (*Frame*) – base window frame ot put into
- **column** (*int, optional*) – column to place this into in the baseframe. Defaults to 0.
- **row** (*int, optional*) – row to place this into in the baseframe. Defaults to 1.

Returns frame for tree views

Return type Frame

frontend.handle_exceptions.**handle_3words_exceptions** (*func: object*)
handle exceptions from the what three words api

Parameters **func** (*function*) – function to be decorated

Returns decorator

frontend.handle_exceptions.**handle_db_exceptions** (*func: object*)
decorator to handle exceptions from the database

Parameters **func** (*function*) – function to be decorated

Returns decorator

class frontend.pop_up.**UpdateMsg** (*message: str*)
Bases: object
display message when something is succesfully updated or otherwise

__init__ (*message: str*) → None
constructor for class for displaying update messages

Parameters **message** (*str*) – message to be displayed

__weakref__
list of weak references to the object (if defined)

destroy ( ) → None
destroy window

utils code for the backend

class backend.common.**DBClass**
Bases: object
class used by all backend classes for interacting with sqllite

__sql_attempt (*sql: str*) → list
wrapper for a sql attempt to connect/commit to db (private)

Parameters **sql** (*str*) – sql query in a string

Returns return from db

Return type list

__weakref__
list of weak references to the object (if defined)

execute_sql (*sql: str*) → list
wrapper for a sql attempt with try/ except

Parameters **sql** (*str*) – sql in string

Returns return from db

Return type list

init_tables ( )
creates tables if hasnt already been created

backend.common.**coordinates_to_words** (*lat: str, lon: str*) → dict
convert to co ords to words using the what three words api

Parameters
- **lat** (*str*) – latitude
- **lon** (*str*) – longitude

Returns threewords location string

Return type dict

backend.common.**words_to_coordinates** (*words: str*) → str
turn what three words location into latitude and location

Parameters **words** (*str*) – what three words as a string with each word separated with a .

Returns co ords location in a string e.e. (1.1,1.1)

Return type str

handler class for returning objects for interacting with the db or select all type functions

class backend.main.**Backend**
Bases: object
handle returning of objects for interacting with the database

__weakref__
list of weak references to the object (if defined)

existing_item (*name: str*) → ExistingMenuItem
instantiates an existing item object

Parameters **name** (*str*) – of existing menu item

Returns instantiates object

Return type ExistingMenuItem

existing_order (*order_id: str*) → ExistingOrder
instantiates na existing order object

Parameters **order_id** (*int*) – order id of exisitng order

Returns instantiated order object

Return type ExistingOrder

classmethod **init_db** ( ) → None
initialise the db

new_item (*name: str*, *price: str*) → NewMenuItem
instantiates a new item object

Parameters
- **name** (*str*) – Name of new menu item
- **price** (*int*) – Price

Returns instanciated object

Return type NewMenuItem

new_order (*customer: str*, *location_co_ords: str*) → NewOrder
instantiates a new order object

Parameters
- **customer** (*str*) – customer username
- **location_co_ords** (*str*) – latitude and longitude separated by a comma

Returns instantiated new order object

Return type NewOrder

classmethod **view_menu** ( ) → list
shows alll menu items and their prices

Returns list returned by db of all menu items i.e. [(“hotdog”, 1)]

Return type list

classmethod **view_orders** ( ) → list
show all orders from the db (no items attatched)

Returns list returned by db of all orders

Return type list

all interactions with the menu

class backend.menu.**ExistingMenuItem** (*name: str*)
Bases: Menu
instantiates a new menu item object

__init__ (*name: str*) → None
init method for a new menu item

Parameters **name** (*str*) – unqiue name for menu item already in db

property price: int
fetches price of a menu item from the db

Returns price of menu item

Return type int

save ( ) → None
save the menu item stored in the object to the db

class backend.menu.**Menu** (*name: str | None = None*)
Bases: DBClass
SUperclass with basic methods for interacting with the menu

__init__ (*name: str | None = None*) → None

delete_from_db ( ) → None
Delete teh menu item (stored in the object) from the db

view_menu ( ) → list
view the menu as it currently is in the db

Returns e.g. [("name1", price)]

Return type list

class backend.menu.**NewMenuItem** (*name: str, price_input: str*)
Bases: Menu
instantiates a new menu item object

__init__ (*name: str, price_input: str*) → None
constructor for a new menu item

Parameters
- **name** (*str*) – name of menu item (must be unique!)
- **price** (*int*) – price fo menu item

save ( ) → None
save the menu item stored in the object

all interactions with orders

class backend.order.**ExistingOrder** (*order_id: str*)
Bases: Order
class for all interactions with existing orders

__init__ (*order_id: str*) → None
constructor for existing order, found by id

__set_date ( ) → datetime
get the date of the order from the db and
overwrite the date in the super class,
if not found set to none, added to super class both classes needed set order

Returns date of the order

Return type datetime

add_items (*name: str, quantity: int*) → None
add a given quantity of an item to the db

Parameters
- **name** (*str*) – name of menu item (must be unique!)
- **quantity** (*int*) – amount of the item to add the db

property location_co_ords
converts what three words to co ords

Returns lat and long in a string e.g. "1,1"

Return type str

remove_items (*name: str*) → None
remove an item and all of its quantities from the db

Parameters **name** (*str*) – name of menu item (must be unique!)

update_order ( ) → None
updates order in db with new values stored in object

class backend.order.**NewOrder** (*customer: str, location_co_ords: str*)
Bases: Order
new order object which can later be added to db

__init__ (*customer: str, location_co_ords: str*) → None
constructor for a new order

Parameters
- **customer** (*str*) – custoemr name (must be unique)
- **location_co_ords** (*str*) – lat and long in a string e.g. "1,1"

property location_words: str
converts co ords to what three words

Returns what threee words str separated by . e.g. "hello.my.name"

Return type str

property order_id: int
find next available order id for order to take

Returns id of the orders

Return type int

save ( ) → None
save teh order stored in the object to the database

class backend.order.**Order**
Bases: DBClass
super class for all order interactions

__init__ ( ) → None
constructor for superclass

delete ( ) → None
delete the order stored in the object from the db

get_total ( ) → int
return total cost of items in an order

Returns total cost of items in an order

Return type int

set_order_date ( ) → None
set the date of the order to now

view_order_items ( ) → list
view items and their quantities in a specific order

Returns return from db; [(x,y),(a,b)]

Return type list

view_orders ( ) → list
view all orders in teh db without their items

Returns db return i.e. [(x,y,z),(a,b,c)]

Return type list

custom exceptions used by both front end and backend whislt this technically counts as higher coupling, I would argue that this would normally be imported in a package and therfore would not be a problem, I cannot so instead have produced this. It could have been avoided by misusing python exceptions but theis was less pythonic than making custom ones and sufferign the higher coupling

exception custom.exceptions.**NoKeyError**
Bases: Exception
raised when no key is found in the environment

__weakref__
list of weak references to the object (if defined)

exception custom.exceptions.**WhatThreeWordsError**
Bases: Exception
raised when there is a problem with the what three words api

__weakref__
list of weak references to the object (if defined)

- genindex
- modindex
- search

Symbols

A

B

C

D

U

V

W