­­­­

2024 Year 12 Comp Sci

Database and Programming Project

Great southern grammar

Task 6

Alec McDonald

# Table Of Contents

[Table Of Contents 1](#_Toc175324124)

[Part 1 – Planning 2](#_Toc175324125)

[Tasks to be done part 1 Investigate. 2](#_Toc175324126)

[Tasks to be done part 1 Design. 2](#_Toc175324127)

[Tasks to be done part 2 Develop. 2](#_Toc175324128)

[Tasks to be done part 2 Evaluate. 2](#_Toc175324129)

[Time frame 3](#_Toc175324130)

[Problem Outline 3](#_Toc175324131)

[Problem Description: 4](#_Toc175324132)

[Describing Queries 4](#_Toc175324133)

[Sample Data 4](#_Toc175324134)

[Part 1 Design 5](#_Toc175324135)

[Data Dictionary: 5](#_Toc175324136)

[ER Diagram: 6](#_Toc175324137)

[Part 2 Develop 7](#_Toc175324138)

[Required Files 7](#_Toc175324139)

[SQL Queries Used to Create Database: 7](#_Toc175324140)

[SQL Queries Used to Insert Data into the Database: 10](#_Toc175324141)

[Python Code 13](#_Toc175324142)

[Part 2 Evaluate 44](#_Toc175324143)

[SQL Queries Used to Manipulate Data 44](#_Toc175324144)

[Reflection 46](#_Toc175324145)

[Developer retrospect 46](#_Toc175324146)

[Sources 47](#_Toc175324147)

[AI Conversation 48](#_Toc175324148)

# Part 1 – Planning

## Tasks to be done part 1 Investigate.

* Break down tasks to do.
* Outline problem.
* Problem Description.

## Tasks to be done part 1 Design.

* ER Diagram
* Relational Notation
* Data Dictionary
* Describe several queries.

## Tasks to be done part 2 Develop.

* Create an empty database.
* Create a script to insert data.
* Data validation
* Create several different queries.
* Create front end.

## Tasks to be done part 2 Evaluate.

* Reflects on success of your solution.
* Compare ER Diagram to database.
* Extra features implemented.
* Documentation of any known bugs or limitations
* Perform a developer retrospect.
* Document sources used to get information.

## Time frame

I have 5 weeks to complete this project.

Starting week 1 term 3 and to be completed by week 5 term 3.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | = Not Started |  | = Doing |  | = Finished |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Part | Key Point | Item | Due Date | Status |
| 1 | Investigate | Project Breakdown | Week 1 |  |
| Problem Outline | Week 1 |  |
| Problem Description | Week 1 |  |
| Design | ER Diagram | Week 2 |  |
| Relational Notation | Week 2 |  |
| Data Dictionary | Week 2 |  |
| Describe several queries | Week 2 |  |
| 2 | Develop | Create an empty database. | Week 3 |  |
| Create a script to insert data. | Week 3 |  |
| Data validation | Week 3 |  |
| Create several different queries. | Week 4 |  |
| Create front end. | Week 4 |  |
| Evaluate | Reflects on success of your solution | Week 5 |  |
| Compare ER Diagram to database. | Week 5 |  |
| Extra features implemented. | Week 5 |  |
| Documentation of any known bugs or limitations | Week 5 |  |
| Perform a developer retrospect. | Week 5 |  |
| Document sources used to get information. | Week 5 |  |

## Problem Outline

The purpose of this project is to create a database and interactive terminal, or GUI made from python for the Great Southern Grammar catering team. Their current method of dealing with catering requests is through an online form which contains a series of checkboxes and user input boxes. This form does not reference any data sources to provide accurate or timely information to the users of the form. I aim to create a well organised and functional database to store information like requesters first and last name, email address, function name, function date, catering charges, type of catering, location of function, number of people catering for, time catering is requested for, what meal is being catered for, the costs of the meal that is being requested, a menu, specific catering requests, special dietary requirements, and an urgent request notice.

## Problem Description:

The database will need to consist of multiple tables such as requester information, location information, meal item (breakfast, morning tea, lunch, afternoon tea, dinner, other), event table, a catering table, a catering charges table and a menu table. To address the requirements of the client to have a space to store important information on the requester and event in a safe and easily retrievable way the database also includes a table that’s have meal items and their prices so that users can easily retrieve the cost of the meals being catered for. The database will also include the location table as it is important that the caterers have easy access to find out the location that they need to bring their services to. The Database would need bank details so the catering team can bill the expenses to the right people. This would need to be stored in a safe place as it is personal information like the requesters name and contact details.

Users might want to retrieve location information so that the caters know where they need to go, users might want to retrieve menu items so they know what they need to prepare for the function, they might want to be able to retrieve the function time and how many people are being catered for. These are all very important details that the catering team would be able to retrieve with relative ease. The catering team might want to retrieve the contact information of the requester as they might encounter a problem and need to contact the requester to clear up some details.

To achieve all this, I will be using SQLite and the SQL language to create a relational database. Using SQL, I will create around 7 tables to store the required data in order to meet the client’s requests. SQL queries will be used to make sure the database is functioning and in working order to be handed over to the client. The database will be integrated with a basic python interface that will easily allow the catering team to access all the information they need to provide their services to the customers functions.

## Describing Queries

There are multiple queries that users will need to use such as select and inner join SQL queries. This will be used to retrieve information like retrieving the event details and requester name and contact details and showing them together. You could also retrieve the dietary requirement that need to be retrieved by selecting the dietary requirements for the event. The users can also retrieve the menu for the event so they can see what and how many meals they would specifically need to prepare. I have used a variety of select and inner join queries to extract information from the data base like catering information, menu information, dietary information and requester information. The join is used to substitute the foreign keys in the tables with the name value of the key so a viewer can understand what it is showing. Insert queries where used to insert catering requests that requesters make into the database.

## Sample Data

Sample data can be found in the sql script called ‘InsertData’

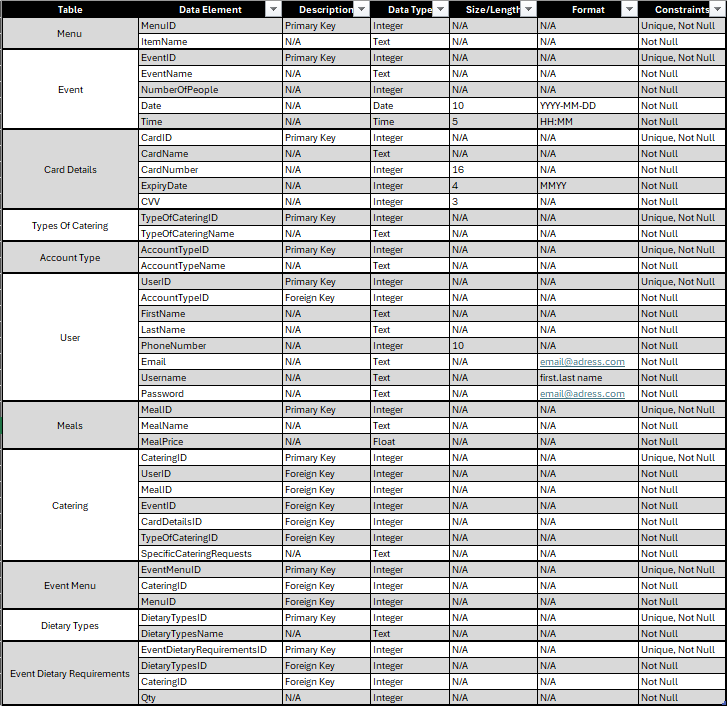
# Part 1 Design

## Data Dictionary:

The normalised data can be found in the excel spreadsheet bellow.

[Click me to view the data dictionary](https://greatsoutherngrammar-my.sharepoint.com/personal/alec_mcdonald_student_gsg_wa_edu_au/Documents/.Year%2012/Computer%20Science/Task_6/Data%20Dictionary.xlsx)

Alternatively, can be viewed in the photo bellow:



## ER Diagram:

**A computer screen shot of a computer

Description automatically generated**

# Part 2 Develop

## Required Files

****

<https://greatsoutherngrammar-my.sharepoint.com/personal/alec_mcdonald_student_gsg_wa_edu_au/Documents/.Year%2012/Computer%20Science/Task_6/Task_6_Code.py>

<https://greatsoutherngrammar-my.sharepoint.com/personal/alec_mcdonald_student_gsg_wa_edu_au/Documents/.Year%2012/Computer%20Science/Task_6/Task_6_Database.db>

You will need to install custom tkinter. To do this you will need to run the command “pip install customtkinter==0.3”

Alternatively you could place the two custom tkinter files that are in this main folder and also linked bellow in this location: “C:\Users\[YOUR USER]\AppData\Roaming\Python\Python312\site-packages

<https://greatsoutherngrammar-my.sharepoint.com/personal/alec_mcdonald_student_gsg_wa_edu_au/Documents/.Year%2012/Computer%20Science/Task_6/customtkinter-5.2.2.dist-info>

<https://greatsoutherngrammar-my.sharepoint.com/personal/alec_mcdonald_student_gsg_wa_edu_au/Documents/.Year%2012/Computer%20Science/Task_6/customtkinter>

## SQL Queries Used to Create Database:

**CREATE** **TABLE** "Menu" ( /\* Creates a table with the name that is inside the "" \*/

MenuID **integer** **PRIMARY** **KEY** AUTOINCREMENT **NOT** **NULL**, /\* Creates a primary key that is an integer, autoicrements and in not null \*/

ItemName text **NOT** **NULL**); /\* Creates a row that is an integer and is not null \*/

**CREATE** **TABLE** "Event" ( /\* Creates a table with the name that is inside the "" \*/

EventID **integer** **PRIMARY** **KEY** AUTOINCREMENT **NOT** **NULL**, /\* Creates a primary key that is an integer, autoicrements and in not null \*/

EventName text **NOT** **NULL**, /\* Creates a row that is a text and is not null \*/

NumberOfPeople text **NOT** **NULL**, /\* Creates a row that is a text and is not null \*/

"Date" **date**(10) **NOT** **NULL**, /\* Creates a row that is a date with max of 10 characters and is not null \*/

"Time" **time**(5) **NOT** **NULL**); /\* Creates a row that is a time with max of 5 characters and is not null \*/

**CREATE** **TABLE** "TypesOfCatering" ( /\* Creates a table with the name that is inside the "" \*/

TypesOfCateringID **integer** **PRIMARY** **KEY** AUTOINCREMENT **NOT** **NULL**, /\* Creates a primary key that is an integer, autoicrements and in not null \*/

TypesOfCateringName text **NOT** **NULL**); /\* Creates a row that is a text and is not null \*/

**CREATE** **TABLE** "AccountType" ( /\* Creates a table with the name that is inside the "" \*/

AccountTypeID **integer** **PRIMARY** **KEY** AUTOINCREMENT **NOT** **NULL**, /\* Creates a primary key that is an integer, autoicrements and in not null \*/

AccountTypeName text **NOT** **NULL**); /\* Creates a row that is a text and is not null \*/

**CREATE** **TABLE** "Meals" ( /\* Creates a table with the name that is inside the "" \*/

MealID **integer** **PRIMARY** **KEY** AUTOINCREMENT **NOT** **NULL**, /\* Creates a primary key that is an integer, autoicrements and in not null \*/

MealName text **NOT** **NULL**, /\* Creates a row that is a text and is not null \*/

MealPrice **float** **NOT** **NULL**); /\* Creates a row that is a float and is not null \*/

**CREATE** **TABLE** "DietaryTypes" ( /\* Creates a table with the name that is inside the "" \*/

DietaryTypesID **integer** **PRIMARY** **KEY** AUTOINCREMENT **NOT** **NULL**, /\* Creates a primary key that is an integer, autoicrements and in not null \*/

DietaryTypesName text **NOT** **NULL**); /\* Creates a row that is a text and is not null \*/

**CREATE** **TABLE** "CardStatus" ( /\* Creates a table with the name that is inside the "" \*/

CardStatusID **integer** **PRIMARY** **KEY** AUTOINCREMENT **NOT** **NULL**, /\* Creates a primary key that is an integer, autoicrements and in not null \*/

CardStatusName text **NOT** **NULL**); /\* Creates a row that is a text and is not null \*/

**CREATE** **TABLE** "User" ( /\* Creates a table with the name that is inside the "" \*/

UserID **integer** **PRIMARY** **KEY** AUTOINCREMENT **NOT** **NULL**, /\* Creates a primary key that is an integer, autoicrements and in not null \*/

AccountTypeID **integer** **NOT** **NULL**, /\* Creates a row that is an integer and is not null \*/

FirstName text **NOT** **NULL**, /\* Creates a row that is a text and is not null \*/

LastName text **NOT** **NULL**, /\* Creates a row that is a text and is not null \*/

PhoneNumber **integer**(10) **NOT** **NULL**, /\* Creates a row that is an integer with max of 10 characters and is not null \*/

Email text **NOT** **NULL**, /\* Creates a row that is a text and is not null \*/

Username text **NOT** **NULL**, /\* Creates a row that is a text and is not null \*/

Password text **NOT** **NULL**, /\* Creates a row that is a text and is not null \*/

**CONSTRAINT** User\_FK\_1 **FOREIGN** **KEY** (AccountTypeID) **REFERENCES** AccountType(AccountTypeID)); /\* Sets a foreign key \*/

**CREATE** **TABLE** "CardDetails" ( /\* Creates a table with the name that is inside the "" \*/

CardID **integer** **PRIMARY** **KEY** AUTOINCREMENT **NOT** **NULL**, /\* Creates a primary key that is an integer, autoicrements and in not null \*/

UserID **integer** **NULL**, /\* Creates a row that is an integer and can be null \*/

CardNumber **integer**(16) **NOT** **NULL**, /\* Creates a row that is an integer with max of 16 characters and is not null \*/

ExpiryDate **integer**(4) **NOT** **NULL**, /\* Creates a row that is an integer with max of 4 characters and is not null \*/

CVV **integer**(3) **NOT** **NULL**, /\* Creates a row that is an integer with max of 3 characters and is not null \*/

CardStatusID **integer** **NOT** **NULL**, /\* Creates a row that is an integer and is not null \*/

**CONSTRAINT** CardDetails\_FK\_1 **FOREIGN** **KEY** (UserID) **REFERENCES** **User**(UserID) /\* Sets a foreign key \*/

**CONSTRAINT** CardDetails\_FK\_2 **FOREIGN** **KEY** (CardStatusID) **REFERENCES** CardStatus(CardStatusID)); /\* Sets a foreign key \*/

**CREATE** **TABLE** "Catering" ( /\* Creates a table with the name that is inside the "" \*/

CateringID **integer** **PRIMARY** **KEY** AUTOINCREMENT **NOT** **NULL**, /\* Creates a primary key that is an integer, autoicrements and in not null \*/

UserID **integer** **NOT** **NULL**, /\* Creates a row that is an integer and is not null \*/

MealID **integer** **NOT** **NULL**, /\* Creates a row that is an integer and is not null \*/

EventID **integer** **NOT** **NULL**, /\* Creates a row that is an integer and is not null \*/

CardID **integer** **NOT** **NULL**, /\* Creates a row that is an integer and is not null \*/

TypesOfCateringID **integer** **NOT** **NULL**, /\* Creates a row that is an integer and is not null \*/

SpecificCateringRequests text **NOT** **NULL**, /\* Creates a row that is a text and is not null \*/

**CONSTRAINT** Catering\_FK\_1 **FOREIGN** **KEY** (UserID) **REFERENCES** **User**(UserID), /\* Sets a foreign key \*/

**CONSTRAINT** Catering\_FK\_2 **FOREIGN** **KEY** (MealID) **REFERENCES** Meals(MealID), /\* Sets a foreign key \*/

**CONSTRAINT** Catering\_FK\_3 **FOREIGN** **KEY** (EventID) **REFERENCES** Event(EventID), /\* Sets a foreign key \*/

**CONSTRAINT** Catering\_FK\_4 **FOREIGN** **KEY** (CardID) **REFERENCES** CardDetails(CardID), /\* Sets a foreign key \*/

**CONSTRAINT** Catering\_FK\_5 **FOREIGN** **KEY** (TypesOfCateringID) **REFERENCES** TypesOfCatering(TypesOfCateringID)); /\* Sets a foreign key \*/

**CREATE** **TABLE** "EventMenu" ( /\* Creates a table with the name that is inside the "" \*/

EventMenuID **integer** **PRIMARY** **KEY** AUTOINCREMENT **NOT** **NULL**, /\* Creates a primary key that is an integer, autoicrements and in not null \*/

CateringID **integer** **NOT** **NULL**, /\* Creates a row that is an integer and is not null \*/

MenuID **integer** **NOT** **NULL**, /\* Creates a row that is an integer and is not null \*/

**CONSTRAINT** Catering\_FK\_1 **FOREIGN** **KEY** (CateringID) **REFERENCES** Catering(CateringID), /\* Sets a foreign key \*/

**CONSTRAINT** Catering\_FK\_2 **FOREIGN** **KEY** (MenuID) **REFERENCES** Menu(MenuID)); /\* Sets a foreign key \*/

**CREATE** **TABLE** "EventDietaryRequirements" ( /\* Creates a table with the name that is inside the "" \*/

EventDietaryRequirementsID **integer** **PRIMARY** **KEY** AUTOINCREMENT **NOT** **NULL**, /\* Creates a primary key that is an integer, autoicrements and in not null \*/

DietaryTypesID **integer** **NOT** **NULL**, /\* Creates a row that is an integer and is not null \*/

CateringID **integer** **NOT** **NULL**, /\* Creates a row that is an integer and is not null \*/

Qty **integer** **NOT** **NULL**, /\* Creates a row that is an integer and is not null \*/

**CONSTRAINT** EventDietaryRequirements\_FK\_1 **FOREIGN** **KEY** (DietaryTypesID) **REFERENCES** DietaryTypes(DietaryTypesID), /\* Sets a foreign key \*/

**CONSTRAINT** EventDietaryRequirements\_FK\_2 **FOREIGN** **KEY** (CateringID) **REFERENCES** Catering(CateringID)); /\* Sets a foreign key \*/

## SQL Queries Used to Insert Data into the Database:

**INSERT** **INTO** Menu (ItemName) /\* Insets data into the selected table and rows \*/

**VALUES** /\* Telling the program that the following lines are vaues that need to be inserted into database \*/

('Chicken\_Nuggets'), /\* Data that is being inserted \*/

('Pizza'), /\* Data that is being inserted \*/

('Fruit\_Kabobs'), /\* Data that is being inserted \*/

('Vegetable\_Sticks\_with\_Dip'), /\* Data that is being inserted \*/

('Sandwiches'), /\* Data that is being inserted \*/

('Macaroni\_and\_Cheese'), /\* Data that is being inserted \*/

('Cheese\_and\_Crackers'), /\* Data that is being inserted \*/

('Cupcakes'), /\* Data that is being inserted \*/

('Brownies'), /\* Data that is being inserted \*/

('Bacon\_and\_Eggs'); /\* Data that is being inserted \*/

**INSERT** **INTO** Event (EventName, NumberOfPeople, 'Date', 'Time') /\* Insets data into the selected table and rows \*/

**VALUES** /\* Telling the program that the following lines are vaues that need to be inserted into database \*/

('Science\_Fair','150','2024-09-15','09:00'), /\* Data that is being inserted \*/

('Math\_Olympiad','100','2024-10-03','08:30'), /\* Data that is being inserted \*/

('Art\_Exhibition','200','2024-11-20','10:00'), /\* Data that is being inserted \*/

('Music\_Concert','250','2024-12-05','18:00'), /\* Data that is being inserted \*/

('Drama\_Play','180','2024-10-25','19:00'), /\* Data that is being inserted \*/

('Sports\_Day','300','2024-09-30','08:00'), /\* Data that is being inserted \*/

('Book\_Fair','220','2024-11-10','09:30'), /\* Data that is being inserted \*/

('Career\_Day','120',' 2024-12-01','10:00'), /\* Data that is being inserted \*/

('Parent\_Teacher\_Conference','60','2024-10-15','16:00'), /\* Data that is being inserted \*/

('Cultural\_Festival','350','2024-11-25','14:00'); /\* Data that is being inserted \*/

**INSERT** **INTO** TypesOfCatering (TypesOfCateringName) /\* Insets data into the selected table and rows \*/

**VALUES** /\* Telling the program that the following lines are vaues that need to be inserted into database \*/

('On\_Site\_Catering'), /\* Data that is being inserted \*/

('Off\_Sight\_Catering'); /\* Data that is being inserted \*/

**INSERT** **INTO** AccountType (AccountTypeName) /\* Insets data into the selected table and rows \*/

**VALUES** /\* Telling the program that the following lines are vaues that need to be inserted into database \*/

('Catering\_Staff'), /\* Data that is being inserted \*/

('Users'), /\* Data that is being inserted \*/

('NotActive'); /\* Data that is being inserted \*/

**INSERT** **INTO** Meals (MealName, MealPrice) /\* Insets data into the selected table and rows \*/

**VALUES** /\* Telling the program that the following lines are vaues that need to be inserted into database \*/

('Breakfast','5.00'), /\* Data that is being inserted \*/

('Morning\_Tea','5.00'), /\* Data that is being inserted \*/

('Lunch','7.50'), /\* Data that is being inserted \*/

('Afternoon\_Tea','5.00'), /\* Data that is being inserted \*/

('Dinner','10.00'); /\* Data that is being inserted \*/

**INSERT** **INTO** DietaryTypes (DietaryTypesName) /\* Insets data into the selected table and rows \*/

**VALUES** /\* Telling the program that the following lines are vaues that need to be inserted into database \*/

('Gluten\_Free'), /\* Data that is being inserted \*/

('Nut\_Free'), /\* Data that is being inserted \*/

('Dairy\_Free'), /\* Data that is being inserted \*/

('Vegetarian'), /\* Data that is being inserted \*/

('Vegan'), /\* Data that is being inserted \*/

('Halal'), /\* Data that is being inserted \*/

('Kosher'), /\* Data that is being inserted \*/

('Low\_Sodium'), /\* Data that is being inserted \*/

('Low\_Sugar'), /\* Data that is being inserted \*/

('Pescatarian'), /\* Data that is being inserted \*/

('Organic'), /\* Data that is being inserted \*/

('Soy\_Free'); /\* Data that is being inserted \*/

**INSERT** **INTO** CardStatus (CardStatusName) /\* Insets data into the selected table and rows \*/

**VALUES** /\* Telling the program that the following lines are vaues that need to be inserted into database \*/

('Active'), /\* Data that is being inserted \*/

('Inactive'); /\* Data that is being inserted \*/

**INSERT** **INTO** "User" (AccountTypeID, FirstName, LastName, PhoneNumber, Email, Username, Password) /\* Insets data into the selected table and rows \*/

**VALUES** /\* Telling the program that the following lines are vaues that need to be inserted into database \*/

('1','Alice','Smith','0412345678','alice.smith@gsg.wa.ed.au','alice.smith','Alice@1234'), /\* Data that is being inserted \*/

('2','Brian','Johnson','0498765432','brian.johnson@gsg.wa.ed.au','brian.johnson','Brian@1234'), /\* Data that is being inserted \*/

('2','Carol','Davis','0456789123','carol.davis@gsg.wa.ed.au','carol.davis','Carol@1234'), /\* Data that is being inserted \*/

('3','Olivia','Miller','0456123789','olivia.miller@gsg.wa.ed.au','olivia.miller','Olivia@1234'), /\* Data that is being inserted \*/

('1','Daniel','Brown','0423456789','daniel.brown@gsg.wa.ed.au','daniel.brown','Daniel@1234'), /\* Data that is being inserted \*/

('2','Emma','Wilson','0434567890','emma.wilson@gsg.wa.ed.au','emma.wilson','Emma@1234'), /\* Data that is being inserted \*/

('2','Frank','Taylor','0487654321','frank.taylor@gsg.wa.ed.au','frank.taylor','Frank@1234'), /\* Data that is being inserted \*/

('2','Grace','Anderson','0445678901','grace.anderson@gsg.wa.ed.au','grace.anderson','Grace@1234'), /\* Data that is being inserted \*/

('3','Noah','Harris','0491234567','noah.harris@gsg.wa.ed.au','noah.harris','Noah@1234'), /\* Data that is being inserted \*/

('1','Henry','Martinez','0478901234','henry.martinez@gsg.wa.ed.au','henry.martinez','Henry@1234'), /\* Data that is being inserted \*/

('2','Isabella','Thomas','0467890123','isabella.thomas@gsg.wa.ed.au','isabella.thomas','Isabella@1234'), /\* Data that is being inserted \*/

('2','Jack','Lee','0412340987','jack.lee@gsg.wa.ed.au','jack.lee','Jack@1234'), /\* Data that is being inserted \*/

('1','1','1','0123456789','1.1@gsg.wa.ed.au','1','1'), /\* Data that is being inserted \*/

('2','2','2','0987654321','2.2@gsg.wa.ed.au','2','2'), /\* Data that is being inserted \*/

('3','3','4','0918273645','3.3@gsg.wa.ed.au','3','3'); /\* Data that is being inserted \*/

**INSERT** **INTO** CardDetails (UserID, CardNumber, ExpiryDate, CVV, CardStatusID) /\* Insets data into the selected table and rows \*/

**VALUES** /\* Telling the program that the following lines are vaues that need to be inserted into database \*/

('2','1234567890123456','1126','456','1'), /\* Data that is being inserted \*/

('3','2345678901234567','1028','789','1'), /\* Data that is being inserted \*/

('6','3456789012345678','0927','101','1'), /\* Data that is being inserted \*/

('7','4567890123456789','0825','202','1'), /\* Data that is being inserted \*/

('8','5678901234567890','0728','303','1'), /\* Data that is being inserted \*/

('11','6789012345678901','0626','404','1'), /\* Data that is being inserted \*/

('12','7890123456789012','0529','505','1'), /\* Data that is being inserted \*/

('2','4536123478901234','1123','567','2'), /\* Data that is being inserted \*/

('6','4789123456012345','0522','789','2'), /\* Data that is being inserted \*/

('11','4123456789012345','0923','901','2'), /\* Data that is being inserted \*/

('14','4567123456789012','1225','123','1'), /\* Data that is being inserted \*/

('15','1234567899876543','1024','901','1'); /\* Data that is being inserted \*/

**INSERT** **INTO** Catering (UserID, MealID, EventID, CardID, TypesOfCateringID, SpecificCateringRequests) /\* Insets data into the selected table and rows \*/

**VALUES** /\* Telling the program that the following lines are vaues that need to be inserted into database \*/

('2','1','1','1','1',''), /\* Data that is being inserted \*/

('3','2','2','3','2','Cupcake\_with\_numbers\_on\_them'), /\* Data that is being inserted \*/

('5','3','3','4','1',''), /\* Data that is being inserted \*/

('6','4','4','5','2','Cupcakes\_with\_music\_notes\_one\_them'), /\* Data that is being inserted \*/

('7','5','5','6','1',''), /\* Data that is being inserted \*/

('9','1','6','7','2','Packed\_food\_in\_a\_cooler'), /\* Data that is being inserted \*/

('10','2','7','8','1',''), /\* Data that is being inserted \*/

('2','3','8','2','2',''), /\* Data that is being inserted \*/

('3','4','9','1','1',''), /\* Data that is being inserted \*/

('5','5','10','4','2','Include\_some\_food\_from\_other\_cultures'); /\* Data that is being inserted \*/

**INSERT** **INTO** EventMenu (CateringID, MenuID) /\* Insets data into the selected table and rows \*/

**VALUES** /\* Telling the program that the following lines are vaues that need to be inserted into database \*/

('1','4'), /\* Data that is being inserted \*/

('1','1'), /\* Data that is being inserted \*/

('1','10'), /\* Data that is being inserted \*/

('2','3'), /\* Data that is being inserted \*/

('2','8'), /\* Data that is being inserted \*/

('2','7'), /\* Data that is being inserted \*/

('3','4'), /\* Data that is being inserted \*/

('3','6'), /\* Data that is being inserted \*/

('3','5'), /\* Data that is being inserted \*/

('3','4'), /\* Data that is being inserted \*/

('3','9'), /\* Data that is being inserted \*/

('4','3'), /\* Data that is being inserted \*/

('4','8'), /\* Data that is being inserted \*/

('4','7'), /\* Data that is being inserted \*/

('5','4'), /\* Data that is being inserted \*/

('5','2'), /\* Data that is being inserted \*/

('5','9'), /\* Data that is being inserted \*/

('6','3'), /\* Data that is being inserted \*/

('6','5'), /\* Data that is being inserted \*/

('6','7'), /\* Data that is being inserted \*/

('6','10'), /\* Data that is being inserted \*/

('7','4'), /\* Data that is being inserted \*/

('7','8'), /\* Data that is being inserted \*/

('7','7'), /\* Data that is being inserted \*/

('8','3'), /\* Data that is being inserted \*/

('8','5'), /\* Data that is being inserted \*/

('8','8'), /\* Data that is being inserted \*/

('9','4'), /\* Data that is being inserted \*/

('9','9'), /\* Data that is being inserted \*/

('9','7'), /\* Data that is being inserted \*/

('10','3'), /\* Data that is being inserted \*/

('10','4'), /\* Data that is being inserted \*/

('10','1'), /\* Data that is being inserted \*/

('10','2'), /\* Data that is being inserted \*/

('10','5'), /\* Data that is being inserted \*/

('10','9'); /\* Data that is being inserted \*/

**INSERT** **INTO** EventDietaryRequirements (DietaryTypesID, CateringID, Qty) /\* Insets data into the selected table and rows \*/

**VALUES** /\* Telling the program that the following lines are vaues that need to be inserted into database \*/

('1','1','7'), /\* Data that is being inserted \*/

('2','2','8'), /\* Data that is being inserted \*/

('3','2','5'), /\* Data that is being inserted \*/

('4','3','2'), /\* Data that is being inserted \*/

('5','3','1'), /\* Data that is being inserted \*/

('6','3','6'), /\* Data that is being inserted \*/

('7','4','9'), /\* Data that is being inserted \*/

('8','4','5'), /\* Data that is being inserted \*/

('9','4','5'), /\* Data that is being inserted \*/

('10','4','4'), /\* Data that is being inserted \*/

('1','6','9'), /\* Data that is being inserted \*/

('2','6','5'), /\* Data that is being inserted \*/

('3','6','3'), /\* Data that is being inserted \*/

('4','6','3'), /\* Data that is being inserted \*/

('5','8','8'), /\* Data that is being inserted \*/

('6','8','6'), /\* Data that is being inserted \*/

('7','9','8'); /\* Data that is being inserted \*/

## Python Code

#Imports all python libraties required for the code

from tkinter import \*

from tkinter import ttk

import sqlite3

import customtkinter as ctk

import os

#Connects the database to the python program

connection = sqlite3.connect("Task\_6\_Database.db")

# Defining variables to make it easier to change the size of everything

standard\_height = 30

standard\_width = 200

standard\_font = "", 15

standard\_y\_padding = 5

standard\_x\_padding = 5

# Defining colour mode for the program (light or dark)

ctk.set\_appearance\_mode("dark")

ctk.set\_default\_color\_theme("dark-blue")

#Creting the window

root = ctk.CTk()

root.title("Task\_6")

screen\_width = root.winfo\_screenwidth()

screen\_height = root.winfo\_screenheight()

root.geometry(f"500x275")

root.resizable(False, False)

class ErrorWindow:

    #Defines all variables in the class

    def \_\_init\_\_(self, parent, message, on\_close):

        self.parent = parent

        self.message = message

        self.on\_close = on\_close

        self.error\_window = None

    #Methad that creates the error window

    def create(self):

        #Creates a new window

        self.error\_window = ctk.CTkToplevel(self.parent)

        self.error\_window.title("Error")

        self.error\_window.resizable(False, False)

        #Make the window modal

        self.error\_window.grab\_set()

        #Make it transient to the main window

        self.error\_window.transient(self.parent)

        #Creates a label

        error\_label = ctk.CTkLabel(

            self.error\_window,

            text=self.message,

            font=standard\_font)

        error\_label.pack(pady=standard\_y\_padding)

        #Checks if variable contains a value

        if self.on\_close == "N/A":

            #Creates a button

            close\_button = ctk.CTkButton(

                self.error\_window,

                text="Close",

                font=standard\_font,

                width=standard\_width,

                height=standard\_height,

                command=self.close\_error\_window)

            close\_button.pack(pady=standard\_y\_padding)

        else:

            #Creates a button

            close\_button = ctk.CTkButton(

                self.error\_window,

                text="Close",

                font=standard\_font,

                width=standard\_width,

                height=standard\_height,

                command=self.close\_error\_window\_with\_callback)

            close\_button.pack(pady=standard\_y\_padding)

        #Center the window on the parent

        self.center\_window()

        #Wait for the window to be closed

        self.parent.wait\_window(self.error\_window)

    #Defines the method that closes the error window

    def close\_error\_window(self):

        self.error\_window.destroy()

    #Defines the method that closes the error window and runs the onther command.

    def close\_error\_window\_with\_callback(self):

        self.error\_window.destroy()

        self.on\_close()

    #Defines the method that centers the error window ontop of the main program

    def center\_window(self):

        self.error\_window.update\_idletasks()

        width = self.error\_window.winfo\_width()

        height = self.error\_window.winfo\_height()

        x = self.parent.winfo\_x() + (self.parent.winfo\_width() // 2) - (width // 2)

        y = self.parent.winfo\_y() + (self.parent.winfo\_height() // 2) - (height // 2)

        self.error\_window.geometry(f'+{x}+{y}')

#Creates a class that handles credentials

class CredentialsChecker:

    #Defines all the variables in the class

    def \_\_init\_\_(self, username, password):

        self.username = username

        self.password = password

    #Defines the method that checks if the username is valid

    def username\_checker(self):

        usernames = str(connection.cursor().execute("SELECT Username FROM User").fetchall()).replace("(","").replace(")","").replace("'","").replace(",","").replace("[","").replace("]","").replace(" ",",")

        usernames = usernames.split(",")

        username\_in\_usernames = False

        for item in usernames:

            if self.username == item:

                username\_in\_usernames = True

                break

        return username\_in\_usernames

    #Defines the method that checks if the password is valid

    def password\_checker(self):

        database\_password = connection.cursor().execute(f"SELECT Password FROM User WHERE Username= '{self.username}'").fetchone()[0]

        if self.password != database\_password:

            correct\_password = False

            return correct\_password

        else:

            correct\_password = True

            return correct\_password

#Defines the function that contains the code to start the program.

def main(button\_type):

    #Clears all widgits

    for widget in root.winfo\_children():

        widget.destroy()

    #Sets the variable to False

    valid\_login = False

    #Checks if the log in button has been pressed yet

    if button\_type == None:

        #Calls the function to create the log in gui

        log\_in\_window()

    else:

        #Makes the username a global variable

        global username

        #Gets all the information attached to the button

        button\_value = button\_type[0]

        username = button\_type[1]

        password = button\_type[2]

        #Creates items for the classes

        credentials\_checker\_1 = CredentialsChecker(username, password)

        credentials\_checker\_2 = CredentialsChecker(username, password)

        error\_window\_1 = ErrorWindow(root, "Your password or username was incorect.\nPlease go back and try again.", lambda: log\_in\_window())

        error\_window\_2 = ErrorWindow(root, "We encountered a problem, please try again.", lambda: log\_in\_window())

        #Checks if was a valid login. If it was it will set the value to True

        if button\_value == "login":

            if credentials\_checker\_1.username\_checker() == True:

                if credentials\_checker\_2.password\_checker() == True:

                    valid\_login = True

                else:

                    error\_window\_1.create()

            else:

                error\_window\_1.create()

        else:

            error\_window\_2.create()

    #Checks if was a valid login

    if valid\_login == True:

        #Gathers required information

        account\_type\_ID = int(account\_type\_ID\_retreval(username))

        #Modifies the window to be full screen

        screen\_width = root.winfo\_screenwidth()

        screen\_height = root.winfo\_screenheight()

        root.geometry(f"{screen\_width}x{screen\_height}")

        root.resizable(True, True)

        #Launches the correct gui bassed off of the account type

        if account\_type\_ID == 1:

            catterer\_gui()

        elif account\_type\_ID == 2:

            requester\_gui()

        else:

            deactivated\_account()

#Defines a function that is used to display all events

def events(left\_frame,right\_frame):

    #Clears all widgits

    for widgets in left\_frame.winfo\_children():

        widgets.destroy()

    for widgets in right\_frame.winfo\_children():

        widgets.destroy()

    #Extracting data from the data base

    data=[]

    cursor = connection.cursor()

    data = cursor.execute("""

        SELECT

            Catering.CateringID AS Catering\_ID,

            Event.EventName AS Event\_Name,

            User.FirstName ||' '|| User.LastName AS Requester,

            Meals.MealName AS Meal,

            Catering.CardID AS Card\_ID,

            TypesOfCatering.TypesOfCateringName AS Type\_Of\_Catering,

            Catering.SpecificCateringRequests AS Specific\_Catering\_Requests

        FROM Catering

        JOIN "User"

        ON User.UserID = Catering.UserID

        JOIN Meals

        ON Meals.MealID = Catering.MealID

        JOIN Event

        ON Event.EventID = Catering.EventID

        JOIN TypesOfCatering

        ON TypesOfCatering.TypesOfCateringID = Catering.TypesOfCateringID;""").fetchall()

    #Creating a table to display the data

    treeview\_frame= ctk.CTkFrame(right\_frame, fg\_color= "#292929")

    treeview\_frame.pack()

    ttk.Style().theme\_use("clam")

    ttk.Style().configure("Treeview", background="#292929",foreground="White", fieldbackground="#292929")

    ttk.Style().configure('Treeview.Heading', background='#292929', foreground='White')

    columns = ("CateringID", "Event\_Name", "Requester\_Name", "Meal", "CardID", "Specific\_Requests", "Specific\_Requests")

    events\_treeview = ttk.Treeview(treeview\_frame, columns=columns, show="headings", height= 40)

    events\_treeview.column("# 1",anchor=W, stretch=True, width = 100)

    events\_treeview.heading("# 1", text="CateringID")

    events\_treeview.column("# 2", anchor=W, stretch=True,width = 200)

    events\_treeview.heading("# 2", text="Event\_Name")

    events\_treeview.column("# 3", anchor=W, stretch=True,width = 200)

    events\_treeview.heading("# 3", text="Requester\_Name")

    events\_treeview.column("# 4", anchor=W, stretch=True,width = 200)

    events\_treeview.heading("# 4", text="Meal")

    events\_treeview.column("# 5", anchor=W, stretch=True,width = 100)

    events\_treeview.heading("# 5", text="CardID")

    events\_treeview.column("# 6", anchor=W, stretch=True,width = 150)

    events\_treeview.heading("# 6", text="Type\_Of\_Catering")

    events\_treeview.column("# 7", anchor=W, stretch=True,width = 400)

    events\_treeview.heading("# 7", text="Specific\_Requests")

    #Entering the data into the table

    for row in data:

        events\_treeview.insert("", "end", values=row)

    #Adding a scroll bar

    treeviewScrollbarY= ttk.Scrollbar(treeview\_frame, command=events\_treeview.yview)

    treeviewScrollbarY.pack(side=LEFT, fill=Y)

    events\_treeview.config(yscrollcommand=treeviewScrollbarY.set)

    events\_treeview.pack(side=RIGHT)

#Creating all the login widgits

def log\_in\_window():

    #Clears all widgits on the window

    for widget in root.winfo\_children():

        widget.destroy()

    #Creates a label

    password\_label = ctk.CTkLabel(

        root,

        text="Please Enter Your \n Username And Password",

        font=standard\_font)

    password\_label.pack(pady=standard\_y\_padding)

    #Creates a user entry

    username\_entry = ctk.CTkEntry(

        root,

        placeholder\_text="Enter Username",

        font=standard\_font,

        width=standard\_width,

        height=standard\_height)

    username\_entry.pack(pady=standard\_y\_padding)

    #Creates a user entry

    password\_entry = ctk.CTkEntry(

        root,

        placeholder\_text="Enter Password",

        font=standard\_font,

        width=standard\_width,

        height=standard\_height,

        show='\*')

    password\_entry.pack(pady=standard\_y\_padding)

    #Creates a button

    login\_button = ctk.CTkButton(

        root,

        text="Log In",

        font=standard\_font,

        width=standard\_width,

        height=standard\_height,

        command=lambda: main(button\_type = ["login",username\_entry.get(),password\_entry.get(),"N/A"]))

    login\_button.pack(pady=standard\_y\_padding)

    #Creates a button

    close\_button = ctk.CTkButton(

        root,

        text="Close Window",

        font=standard\_font,

        width=standard\_width,

        height=standard\_height,

        command=root.destroy)

    close\_button.pack(pady=standard\_y\_padding)

#Retrieving the account type id and returning it to the program

def account\_type\_ID\_retreval(username):

    account\_type\_ID = str(connection.cursor().execute(f"SELECT AccountTypeID FROM User WHERE Username= '{username}'").fetchall()).replace("(","").replace(")","").replace("'","").replace(",","").replace("[","").replace("]","").replace(" ","")

    return(account\_type\_ID)

#Creates the catering staff gui

def catterer\_gui():

    #Creates the frames everything sits in

    main\_frame = ctk.CTkFrame(root)

    main\_frame.pack(fill="both", expand=True)

    main\_frame.rowconfigure(0, weight=1)

    main\_frame.columnconfigure(0, weight=1)

    main\_frame.columnconfigure(1, weight=5)

    left\_frame = ctk.CTkFrame(main\_frame, fg\_color="#292929")

    left\_frame.grid(row=0, column=0, sticky="nsew", padx=standard\_x\_padding)

    left\_frame.grid\_propagate(False)

    left\_frame.pack\_propagate(False)

    right\_frame = ctk.CTkFrame(main\_frame, fg\_color="#292929")

    right\_frame.grid(row=0, column=1, sticky="nsew", pady=standard\_y\_padding, padx=standard\_x\_padding)

    right\_frame.grid\_propagate(False)

    right\_frame.pack\_propagate(False)

    #Creates a menu bar up the top of the window

    menu\_bar = Menu(root)

    root.config(menu=menu\_bar)

    #Adds dropdown menu to menu bar

    exit\_menu = Menu(menu\_bar)

    exit\_menu.add\_command(label='Exit', command=root.destroy)

    events\_menu = Menu(menu\_bar)

    events\_menu.add\_command(label='View All', command= lambda: events(left\_frame,right\_frame))

    events\_menu.add\_command(label='Select Event', command= lambda: select\_event(left\_frame,right\_frame))

    events\_menu.add\_command(label='Event Menu', command= lambda: select\_event\_menu(left\_frame,right\_frame))

    events\_menu.add\_command(label='Event Dietary Requirements', command= lambda: select\_event\_dietary\_requirements(left\_frame,right\_frame))

    menu\_bar.add\_cascade(label="Exit", menu=exit\_menu)

    menu\_bar.add\_cascade(label="Event", menu=events\_menu)

#Defines a function that lets you select events to view

def select\_event\_menu(left\_frame,right\_frame):

    #Clears all widgits

    for widgets in left\_frame.winfo\_children():

        widgets.destroy()

    for widgets in right\_frame.winfo\_children():

        widgets.destroy()

    #Creating a label

    label = ctk.CTkLabel(left\_frame, text="Select The Event You Wish To View", fg\_color="transparent", font= standard\_font)

    label.pack(pady = standard\_y\_padding)

    #Creating list box frame

    listbox\_frame= ctk.CTkFrame(left\_frame, fg\_color= "#292929")

    listbox\_frame.pack(pady = standard\_y\_padding)

    #Creating a list box and inserting data from the database into it

    events\_listbox = Listbox(listbox\_frame, bg= "#292929", fg= "Silver", width= 30, height= 25, font= standard\_font)

    cursor = connection.cursor()

    data = cursor.execute("SELECT EventID,EventName FROM Event").fetchall()

    for row in data:

        events\_listbox.insert(END, row)

    events\_listbox.pack(side=LEFT)

    #Creating a scroll bar

    listboxScrollbar= ctk.CTkScrollbar(listbox\_frame, command=events\_listbox.yview)

    listboxScrollbar.pack(side="right", fill=Y)

    events\_listbox.config(yscrollcommand=listboxScrollbar.set)

    #Creating a button

    changeRankButton = ctk.CTkButton(

        left\_frame,

        text= "View Event Menu",

        font= standard\_font,

        width= standard\_width,

        height= standard\_height,

        command= lambda: event\_menu(right\_frame,events\_listbox),

        )

    changeRankButton.pack(pady = standard\_y\_padding)

#Defines a function that shows all the menu items for the selected event

def event\_menu(right\_frame,events\_listbox):

    #Clears all widgits

    for widgets in right\_frame.winfo\_children():

        widgets.destroy()

    #Gets the user selection from the listbox and extracts the eventID from the selection

    selection = events\_listbox.curselection()

    eventID = selection[0]

    eventID = eventID + 1

    #Extracting data from the database

    cursor = connection.cursor()

    data = cursor.execute(f"""SELECT Menu.ItemName AS Item

        FROM Event

        INNER JOIN Catering

        ON Event.EventID = Catering.EventID

        INNER JOIN EventMenu

        ON Catering.CateringID = EventMenu.CateringID

        INNER JOIN Menu

        ON EventMenu.MenuID = Menu.MenuID

        WHERE Event.EventID = {eventID}

        ORDER BY Event.EventName, Menu.ItemName;""").fetchall()

    #Creating list box frame

    listbox\_frame= ctk.CTkFrame(right\_frame, fg\_color= "#292929")

    listbox\_frame.pack(pady = standard\_y\_padding)

    #Creating a list box and inserting data from the databse into it

    events\_menu\_listbox = Listbox(listbox\_frame, bg= "#292929", fg= "Silver", width= 50, height= 35, font= standard\_font)

    for row in data:

        events\_menu\_listbox.insert(END, row)

    events\_menu\_listbox.pack(side=LEFT)

    #Creates a scroll bar

    listboxScrollbar= ctk.CTkScrollbar(listbox\_frame, command=events\_listbox.yview)

    listboxScrollbar.pack(side="right", fill=Y)

    events\_menu\_listbox.config(yscrollcommand=listboxScrollbar.set)

#Defines a function that lets you select events to view

def select\_event\_dietary\_requirements(left\_frame,right\_frame):

    #Clears all widgits

    for widgets in left\_frame.winfo\_children():

        widgets.destroy()

    for widgets in right\_frame.winfo\_children():

        widgets.destroy()

    #Creating a label

    label = ctk.CTkLabel(left\_frame, text="Select The Event You Wish To View", fg\_color="transparent", font= standard\_font)

    label.pack(pady = standard\_y\_padding)

    #Creating list box frame

    listbox\_frame= ctk.CTkFrame(left\_frame, fg\_color= "#292929")

    listbox\_frame.pack(pady = standard\_y\_padding)

    #Creates a list box that pulls data from the database to display

    events\_listbox = Listbox(listbox\_frame, bg= "#292929", fg= "Silver", width= 30, height= 25, font= standard\_font)

    cursor = connection.cursor()

    data = cursor.execute("SELECT EventID,EventName FROM Event").fetchall()

    for row in data:

        events\_listbox.insert(END, row)

    events\_listbox.pack(side=LEFT)

    #Creates a scroll bar

    listbox\_scrollbar= ctk.CTkScrollbar(listbox\_frame, command=events\_listbox.yview)

    listbox\_scrollbar.pack(side="right", fill=Y)

    events\_listbox.config(yscrollcommand=listbox\_scrollbar.set)

    #Creating a button

    dietary\_requirements\_button = ctk.CTkButton(

        left\_frame,

        text= "View Dietary Requirements",

        font= standard\_font,

        width= standard\_width,

        height= standard\_height,

        command= lambda: event\_dietary\_requirements(right\_frame,events\_listbox),

        )

    dietary\_requirements\_button.pack(pady = standard\_y\_padding)

#Defines a function that shows all the dietary requirements for the selected event

def event\_dietary\_requirements(right\_frame,events\_listbox):

    #Clears all the widgits

    for widgets in right\_frame.winfo\_children():

        widgets.destroy()

    #Gets the user selection from the listbox and extracts the eventID from the selection

    selection = events\_listbox.curselection()

    eventID = selection[0]

    eventID = eventID + 1

    #Extracting data from the database

    cursor = connection.cursor()

    data = cursor.execute(f"""SELECT DietaryTypes.DietaryTypesName AS Dietary\_Types, EventDietaryRequirements.Qty AS Qty

        FROM Event

        JOIN Catering

        ON Event.EventID = Catering.EventID

        JOIN EventDietaryRequirements

        ON Catering.CateringID = EventDietaryRequirements.CateringID

        JOIN DietaryTypes

        ON EventDietaryRequirements.DietaryTypesID = DietaryTypes.DietaryTypesID

        WHERE Event.EventID = {eventID};""").fetchall()

    #Creating list box frame

    listbox\_frame= ctk.CTkFrame(right\_frame, fg\_color= "#292929")

    listbox\_frame.pack(pady = standard\_y\_padding)

    #Creates a list box and inserts data from the listbox into it

    events\_dietary\_requirements\_listbox = Listbox(listbox\_frame, bg= "#292929", fg= "Silver", width= 50, height= 35, font= standard\_font)

    for row in data:

        events\_dietary\_requirements\_listbox.insert(END, row)

    events\_dietary\_requirements\_listbox.pack(side=LEFT)

    #Creates a scroll bar

    listboxScrollbar= ctk.CTkScrollbar(listbox\_frame, command=events\_listbox.yview)

    listboxScrollbar.pack(side="right", fill=Y)

    events\_dietary\_requirements\_listbox.config(yscrollcommand=listboxScrollbar.set)

#Defines a function that lets you select events to view

def select\_event(left\_frame,right\_frame):

    #Clears all the widgits

    for widgets in left\_frame.winfo\_children():

        widgets.destroy()

    for widgets in right\_frame.winfo\_children():

        widgets.destroy()

    #Creating a label

    label = ctk.CTkLabel(left\_frame, text="Select The Event You Wish To View", fg\_color="transparent", font= standard\_font)

    label.pack(pady = standard\_y\_padding)

    #Creating list box frame

    listbox\_frame= ctk.CTkFrame(left\_frame, fg\_color= "#292929")

    listbox\_frame.pack(pady = standard\_y\_padding)

    #Creates a list box that extracts data frmo the database to display

    events\_listbox = Listbox(listbox\_frame, bg= "#292929", fg= "Silver", width= 30, height= 25, font= standard\_font)

    cursor = connection.cursor()

    data = cursor.execute("SELECT EventID,EventName FROM Event").fetchall()

    for row in data:

        events\_listbox.insert(END, row)

    events\_listbox.pack(side=LEFT)

    #Creates a scroll bar

    listboxScrollbar= ctk.CTkScrollbar(listbox\_frame, command=events\_listbox.yview)

    listboxScrollbar.pack(side="right", fill=Y)

    events\_listbox.config(yscrollcommand=listboxScrollbar.set)

    #Creating a button

    view\_event\_button = ctk.CTkButton(

        left\_frame,

        text= "View Event Menu",

        font= standard\_font,

        width= standard\_width,

        height= standard\_height,

        command= lambda: show\_event(right\_frame,events\_listbox),

        )

    view\_event\_button.pack(pady = standard\_y\_padding)

#Defines a function that shows the selected event

def show\_event(right\_frame,events\_listbox):

    #Clears all widgits

    for widgets in right\_frame.winfo\_children():

        widgets.destroy()

    #Gets the user selection from the listbox and extracts the eventID from the selection

    selection = events\_listbox.curselection()

    eventID = selection[0]

    eventID = eventID + 1

    #Extracting data from the database

    cursor = connection.cursor()

    data = cursor.execute(f"""SELECT

            User.FirstName ||' '|| User.LastName AS User,

            Event.EventName AS Event,

            User.PhoneNumber AS Phone,

            User.Email AS Email,

            Event.NumberOfPeople AS Number\_of\_People,

            Event.Date AS Date,

            Event.Time AS Time

        FROM Catering

        INNER JOIN User

        ON User.UserID = Catering.UserID

        INNER JOIN Event

        ON Event.EventID = Catering.EventID

        WHERE Event.EventID = {eventID};""").fetchall()

    #Creates a table to display the data

    treeview\_frame= ctk.CTkFrame(right\_frame, fg\_color= "#292929")

    treeview\_frame.pack()

    ttk.Style().theme\_use("clam")

    ttk.Style().configure("Treeview", background="#292929",foreground="White", fieldbackground="#292929")

    ttk.Style().configure('Treeview.Heading', background='#292929', foreground='White')

    columns = ("Requester\_Name", "Event\_Name", "Phone\_Number", "Email", "Number\_Of\_People", "Date", "Time")

    events\_treeview = ttk.Treeview(treeview\_frame, columns=columns, show="headings", height= 40)

    events\_treeview.column("# 1",anchor=W, stretch=True, width = 200)

    events\_treeview.heading("# 1", text="Requester\_Name")

    events\_treeview.column("# 2", anchor=W, stretch=True,width = 200)

    events\_treeview.heading("# 2", text="Event\_Name")

    events\_treeview.column("# 3", anchor=W, stretch=True,width = 200)

    events\_treeview.heading("# 3", text="Phone\_Number")

    events\_treeview.column("# 4", anchor=W, stretch=True,width = 200)

    events\_treeview.heading("# 4", text="Email")

    events\_treeview.column("# 5", anchor=W, stretch=True,width = 150)

    events\_treeview.heading("# 5", text="Number\_Of\_People")

    events\_treeview.column("# 6", anchor=W, stretch=True,width = 100)

    events\_treeview.heading("# 6", text="Date")

    events\_treeview.column("# 7", anchor=W, stretch=True,width = 100)

    events\_treeview.heading("# 7", text="Time")

    #Inserts data into the table

    for row in data:

        events\_treeview.insert("", "end", values=row)

    #Creates a scroll bar

    treeviewScrollbarY= ttk.Scrollbar(treeview\_frame, command=events\_treeview.yview)

    treeviewScrollbarY.pack(side=LEFT, fill=Y)

    events\_treeview.config(yscrollcommand=treeviewScrollbarY.set)

    events\_treeview.pack(side=RIGHT)

#Creates the requester gui

def requester\_gui():

    #Clears all widgits

    for widget in root.winfo\_children():

        widget.destroy()

    #Creates a menu bar

    menu\_bar = Menu(root)

    root.config(menu=menu\_bar)

    #Creates the main frame the widgits will sit in

    main\_frame = ctk.CTkFrame(root)

    main\_frame.pack(fill="both", expand=True)

    main\_frame.rowconfigure(0, weight=1)

    main\_frame.columnconfigure(0, weight=1)

    main\_frame.columnconfigure(1, weight=1)

    main\_frame.columnconfigure(2, weight=1)

    #Clears all widgits

    for widget in main\_frame.winfo\_children():

        widget.destroy()

    #Creates the widgits

    left\_frame = ctk.CTkFrame(main\_frame, fg\_color="#1f1f1f")

    left\_frame.grid(row=0, column=0, sticky="nsew", padx=standard\_x\_padding)

    left\_frame.grid\_propagate(False)

    left\_frame.pack\_propagate(False)

    middle\_frame = ctk.CTkFrame(main\_frame, fg\_color="#1f1f1f")

    middle\_frame.grid(row=0, column=1, sticky="nsew", pady=standard\_y\_padding, padx=standard\_x\_padding)

    middle\_frame.grid\_propagate(False)

    middle\_frame.pack\_propagate(False)

    right\_frame = ctk.CTkFrame(main\_frame, fg\_color="#1f1f1f")

    right\_frame.grid(row=0, column=2, sticky="nsew", pady=standard\_y\_padding, padx=standard\_x\_padding)

    right\_frame.grid\_propagate(False)

    right\_frame.pack\_propagate(False)

    #Clears all widgits

    for widget in left\_frame.winfo\_children():

        widget.destroy()

    for widget in middle\_frame.winfo\_children():

        widget.destroy()

    for widget in right\_frame.winfo\_children():

        widget.destroy()

    #Creates the menu bar

    exit\_menu = Menu(menu\_bar)

    exit\_menu.add\_command(label='Exit', command=root.destroy)

    #Adds dropdown menu to the menubar

    catering\_menu = Menu(menu\_bar)

    catering\_menu.add\_command(label='Catering Requests', command= lambda: cattering\_request(left\_frame,middle\_frame,right\_frame))

    menu\_bar.add\_cascade(label="Exit", menu=exit\_menu)

    menu\_bar.add\_cascade(label="Catering", menu=catering\_menu)

#Creates the catering request form

def cattering\_request(left\_frame,middle\_frame,right\_frame):

    #Clears all widgits

    for widget in left\_frame.winfo\_children():

        widget.destroy()

    for widget in middle\_frame.winfo\_children():

        widget.destroy()

    for widget in right\_frame.winfo\_children():

        widget.destroy()

    #Creates a label

    event\_name\_label = ctk.CTkLabel(left\_frame, text="Please enter event name:", fg\_color="transparent", font= standard\_font)

    event\_name\_label.grid(row = 0, column = 0, pady = 10)

    #Creates a user entry

    event\_name\_entry = ctk.CTkEntry(

        left\_frame,

        font= standard\_font,

        width= standard\_width,

        height= standard\_height,

        )

    event\_name\_entry.grid(row = 0, column = 1, pady = 10)

    #Creates a label

    number\_of\_people\_label = ctk.CTkLabel(left\_frame, text="Please enter number of people:", fg\_color="transparent", font= standard\_font)

    number\_of\_people\_label.grid(row = 1, column = 0, pady = 10)

    #Creates a user entry

    number\_of\_people\_entry = ctk.CTkEntry(

        left\_frame,

        font= standard\_font,

        width= standard\_width,

        height= standard\_height,

        )

    number\_of\_people\_entry.grid(row = 1, column = 1, pady = 10)

    #Creating a label

    date\_label = ctk.CTkLabel(left\_frame, text="Select The Date Of The Event", fg\_color="transparent", font= standard\_font)

    date\_label.grid(row = 2, column = 0)

    #Creates a frame

    date\_dropdown\_frame= ctk.CTkFrame(left\_frame, fg\_color= "#292929")

    date\_dropdown\_frame.grid(row = 3, column = 0, pady = 10)

    #Creates a dropdown menu

    day\_dropdown = ctk.CTkOptionMenu(master=date\_dropdown\_frame, values=["01","02","03","04","05","06","07","08","09","10","11","12","13","14","15","16","17","18","19","20","21","22","23","24","25","26","27","28","29","30","31"])

    day\_dropdown.grid(row = 0, column = 0)

    day\_dropdown.set("Day")

    #Creates a dropdown menu

    month\_dropdown = ctk.CTkOptionMenu(master=date\_dropdown\_frame, values=["01","02","03","04","05","06","07","08","09","10","11","12"])

    month\_dropdown.grid(row = 1, column = 0)

    month\_dropdown.set("Month")

    #Creates a dropdown menu

    year\_dropdown = ctk.CTkOptionMenu(master=date\_dropdown\_frame, values=["2024","2025","2026","2027","2028","2029","2030","2031","2032","2033","2034"])

    year\_dropdown.grid(row = 2, column = 0)

    year\_dropdown.set("Year")

    #Creating a label

    time\_label = ctk.CTkLabel(left\_frame, text="Enter The Time Of The Event", fg\_color="transparent", font= standard\_font)

    time\_label.grid(row = 4, column = 0, pady = 10)

    #Creates a dropdown menu

    hour\_dropdown = ctk.CTkOptionMenu(master=left\_frame, values=["01","02","03","04","05","06","07","08","09","10","11","12","13","14","15","16","17","18","19","20","21","22","23","24"])

    hour\_dropdown.grid(row = 5, column = 0)

    hour\_dropdown.set("Hour")

    #Creates a dropdown menu

    minute\_dropdown = ctk.CTkOptionMenu(master=left\_frame, values=["00","15","30","45"])

    minute\_dropdown.grid(row = 5, column = 1)

    minute\_dropdown.set("Minute")

    #Creates a label

    type\_of\_catering\_label = ctk.CTkLabel(left\_frame, text="Select The Type Of Catering", fg\_color="transparent", font= standard\_font)

    type\_of\_catering\_label.grid(row = 6, column = 0, pady = 10)

    #Creates a dropdown menu

    type\_of\_catering\_dropdown = ctk.CTkOptionMenu(master=left\_frame, values=["On Sight-Catering","Off Sight-Catering"])

    type\_of\_catering\_dropdown.grid(row = 6, column = 1, pady = 10)

    type\_of\_catering\_dropdown.set("On Sight-Catering")

    #Creates a label

    cattering\_for\_label = ctk.CTkLabel(left\_frame, text="Catering For:", fg\_color="transparent", font= standard\_font)

    cattering\_for\_label.grid(row = 7, column = 0, pady = 10)

    #Creates a dropdown menu

    cattering\_for\_dropdown = ctk.CTkOptionMenu(master=left\_frame, values=["Breakfast","Morning Tea","Lunch","Afternoon Tea","Dinner"])

    cattering\_for\_dropdown.grid(row = 7, column = 1, pady = 10)

    cattering\_for\_dropdown.set("Breakfast")

    #Creates a label

    cattering\_requirements\_label = ctk.CTkLabel(left\_frame, text="Special Catering Requirements:", fg\_color="transparent", font= standard\_font)

    cattering\_requirements\_label.grid(row = 8, column = 0, pady = 10)

    #Creates a user entry

    cattering\_requirements\_entry = ctk.CTkEntry(

        left\_frame,

        font= standard\_font,

        width= standard\_width,

        height= standard\_height,

        )

    cattering\_requirements\_entry.grid(row = 8, column = 1, sticky = W, pady = 10)

    #Creates a label

    confirm\_label = ctk.CTkLabel(left\_frame, text="WHEN DONE CLICK \nTHIS BUTTON", fg\_color="transparent", font=("",20))

    confirm\_label.grid(row = 9, column = 0, pady = 200)

    #Creating a button

    confirm\_button = ctk.CTkButton(

        left\_frame,

        text= "Enter Request",

        font= standard\_font,

        width= standard\_width,

        height= standard\_height,

        command= lambda: data\_validation(left\_frame,middle\_frame,right\_frame,event\_name\_entry,number\_of\_people\_entry,day\_dropdown,month\_dropdown,year\_dropdown,minute\_dropdown,hour\_dropdown,type\_of\_catering\_dropdown,cattering\_for\_dropdown,cattering\_requirements\_entry,event\_menu\_listbox,event\_dietary\_types\_listbox),

        )

    confirm\_button.grid(row = 9, column = 1, pady = 200)

    #Creates a ctk label

    label = ctk.CTkLabel(middle\_frame, text="Select An Item Then Click The Button Bellow \nThe Box To Add/Remove It From The Order \n\nFORMAT = MenuID, ItemName", fg\_color="transparent", font= standard\_font)

    label.grid(row = 0, column = 3)

    #Creating list box frame

    listbox\_frame= ctk.CTkFrame(middle\_frame, fg\_color= "#292929")

    listbox\_frame.grid(row = 1, column = 3)

    #Creates a listbox and inserts data from the database into it

    menu\_listbox = Listbox(listbox\_frame, bg= "#292929", fg= "Silver", width= 30, height= 12, font= standard\_font)

    cursor = connection.cursor()

    data = cursor.execute("SELECT MenuID,ItemName FROM Menu").fetchall()

    for row in data:

        menu\_listbox.insert(END, row)

    menu\_listbox.pack(side=LEFT)

    #Creates a scroll bar

    listboxScrollbar= ctk.CTkScrollbar(listbox\_frame, command=menu\_listbox.yview)

    listboxScrollbar.pack(side="right", fill=Y)

    menu\_listbox.config(yscrollcommand=listboxScrollbar.set)

    #Creating a button

    add\_button = ctk.CTkButton(

        middle\_frame,

        text= "Add Item",

        font= standard\_font,

        width= standard\_width,

        height= standard\_height,

        command= lambda: add\_item\_menu(menu\_listbox,event\_menu\_listbox),

        )

    add\_button.grid(row = 2, column = 3, pady = 10)

    #Creates a ctk frame

    listboxFrameTwo= ctk.CTkFrame(middle\_frame, fg\_color= "#292929")

    listboxFrameTwo.grid(row = 4, column = 3, pady = 10)

    #Creates a list box

    event\_menu\_listbox = Listbox(listboxFrameTwo, bg= "#292929", fg= "Silver", width= 30, height= 12, font= standard\_font)

    event\_menu\_listbox.pack(side=LEFT)

    #Creates a scroll bar

    listboxScrollbar= ctk.CTkScrollbar(listboxFrameTwo, command=event\_menu\_listbox.yview)

    listboxScrollbar.pack(side="right", fill=Y)

    event\_menu\_listbox.config(yscrollcommand=listboxScrollbar.set)

    #Creating a button

    remove\_button = ctk.CTkButton(

        middle\_frame,

        text= "Remove Item",

        font= standard\_font,

        width= standard\_width,

        height= standard\_height,

        command= lambda: remove\_item\_menu(menu\_listbox,event\_menu\_listbox),

        )

    remove\_button.grid(row = 5, column = 3, pady = 10)

    #Creates a ctk label

    label = ctk.CTkLabel(right\_frame, text="Select An Item Then Click The Button Bellow \nThe Box To Add/Remove It From The Order \nIf Dietary Type Isn't In List Contact Catering Staff  \n\nFORMAT = DietaryTypesID, Name", fg\_color="transparent", font= standard\_font)

    label.grid(row = 0, column = 4)

    #Creating list box frame

    listbox\_frame= ctk.CTkFrame(right\_frame, fg\_color= "#292929")

    listbox\_frame.grid(row = 1, column = 4)

    #Creates a listbox and inserts data from the database into it

    dietary\_types\_listbox = Listbox(listbox\_frame, bg= "#292929", fg= "Silver", width= 30, height= 12, font= standard\_font)

    cursor = connection.cursor()

    data = cursor.execute("SELECT DietaryTypesID,DietaryTypesName FROM DietaryTypes").fetchall()

    for row in data:

        dietary\_types\_listbox.insert(END, row)

    dietary\_types\_listbox.pack(side=LEFT)

    #Creates a scroll bar

    listboxScrollbar= ctk.CTkScrollbar(listbox\_frame, command=dietary\_types\_listbox.yview)

    listboxScrollbar.pack(side="right", fill=Y)

    dietary\_types\_listbox.config(yscrollcommand=listboxScrollbar.set)

    #Creates a user entry

    qty\_entry = ctk.CTkEntry(

        right\_frame,

        placeholder\_text="Please enter Qty",

        font= standard\_font,

        width= standard\_width,

        height= standard\_height,

        )

    qty\_entry.grid(row = 2, column = 4, pady = 10)

    #Creating a button

    add\_button = ctk.CTkButton(

        right\_frame,

        text= "Add Item",

        font= standard\_font,

        width= standard\_width,

        height= standard\_height,

        command= lambda: add\_item\_dietary(dietary\_types\_listbox,event\_dietary\_types\_listbox,qty\_entry),

        )

    add\_button.grid(row = 3, column = 4, pady = 10)

    label = ctk.CTkLabel(right\_frame, text="FORMAT = DietaryTypesID, Name, Qty", fg\_color="transparent", font= standard\_font)

    label.grid(row = 4, column = 4)

    #Creates a ctk frame

    listboxFrameTwo= ctk.CTkFrame(right\_frame, fg\_color= "#292929")

    listboxFrameTwo.grid(row = 5, column = 4, pady = 10)

    event\_dietary\_types\_listbox = Listbox(listboxFrameTwo, bg= "#292929", fg= "Silver", width= 30, height= 12, font= standard\_font)

    event\_dietary\_types\_listbox.pack(side=LEFT)

    listboxScrollbar= ctk.CTkScrollbar(listboxFrameTwo, command=event\_dietary\_types\_listbox.yview)

    listboxScrollbar.pack(side="right", fill=Y)

    event\_dietary\_types\_listbox.config(yscrollcommand=listboxScrollbar.set)

    #Creating a button

    remove\_button = ctk.CTkButton(

        right\_frame,

        text= "Remove Item",

        font= standard\_font,

        width= standard\_width,

        height= standard\_height,

        command= lambda: remove\_item\_dietary(dietary\_types\_listbox,event\_dietary\_types\_listbox,qty\_entry),

        )

    remove\_button.grid(row = 6, column = 4, pady = 10)

#Defines function that adds an item to a list box

def add\_item\_menu(menu\_listbox,event\_menu\_listbox):

    error\_window\_3 = ErrorWindow(root, "You did not make a selection, please try again.", "N/A")

    #Gets the user selection

    selection = menu\_listbox.curselection()

    #Checks if their is a selection

    if selection:

        #Insert item into the empty listbox and delete it from the original

        item = str(menu\_listbox.get(selection[0])).replace("(","").replace(",","").replace("'","").replace(")","")

        event\_menu\_listbox.insert(END, f"{item}")

        menu\_listbox.delete(selection[0])

    else:

        error\_window\_3.create()

#Defines function that removes an item from a list box

def remove\_item\_menu(menu\_listbox,event\_menu\_listbox):

    #Creates an item for the errorwindow class

    error\_window\_4 = ErrorWindow(root, "You did not make a selection, please try again.", "N/A")

    # Gets the user selection

    selection = event\_menu\_listbox.curselection()

    #Checks if their is a selection

    if selection:

        #Insert item into the original listbox and delete it from the empty one

        item = str(event\_menu\_listbox.get(selection[0])).replace("(","").replace(",","").replace("'","").replace(")","")

        menu\_listbox.insert(END, f"{item}")

        event\_menu\_listbox.delete(selection[0])

    else:

        error\_window\_4.create()

#Defines function that adds an item to a list box

def add\_item\_dietary(dietary\_types\_listbox,event\_dietary\_types\_listbox,qty\_entry):

    #Creates an item for the errorwindow class

    error\_window\_5 = ErrorWindow(root, "You did not make a selection, please try again.", "N/A")

    error\_window\_6 = ErrorWindow(root, "Your qty was not an integer, please try again.", "N/A")

    #Gets the user selection

    selection = dietary\_types\_listbox.curselection()

    #Checks if their is a selection

    if selection:

        validity = True

        #Checks if value is an integer

        try:

            qty = int(qty\_entry.get())

        except ValueError:

            validity = False

            error\_window\_6.create()

        if validity == True:

            #Gets required information

            qty = str(qty\_entry.get())

            item = str(dietary\_types\_listbox.get(selection[0])).replace("(","").replace(",","").replace("'","").replace(")","")

            item = item + " " + qty

        #Inserts it into one list box and deletes it from the other

        event\_dietary\_types\_listbox.insert(END, f"{item}")

        dietary\_types\_listbox.delete(selection[0])

    else:

        error\_window\_5.create()

#Defines function that removes an item from a list box

def remove\_item\_dietary(dietary\_types\_listbox,event\_dietary\_types\_listbox):

    #Creates an item for the errorwindow class

    error\_window\_7 = ErrorWindow(root, "You did not make a selection, please try again.", "N/A")

    #Gets the user selection

    selection = event\_dietary\_types\_listbox.curselection()

    #Checks if their is a selection

    if selection:

        #Gets required information

        item = str(event\_dietary\_types\_listbox.get(selection[0])).replace("(","").replace(",","").replace("'","").replace(")","")

        item = item.split(" ")

        id = item[0]

        item\_name = item[1]

        item = id + " " + item\_name

        #Inserts it into one list box and deletes it from the other

        dietary\_types\_listbox.insert(END, f"{item}")

        event\_dietary\_types\_listbox.delete(selection[0])

    else:

        error\_window\_7.create()

#Defines function that checks all the data validation

def data\_validation(left\_frame,middle\_frame,right\_frame,event\_name\_entry,number\_of\_people\_entry,day\_dropdown,month\_dropdown,year\_dropdown,minute\_dropdown,hour\_dropdown,type\_of\_catering\_dropdown,cattering\_for\_dropdown,cattering\_requirements\_entry,event\_menu\_listbox,event\_dietary\_types\_listbox):

    #Creates an item for the errorwindow class

    error\_window\_8 = ErrorWindow(root, "Value error, please try again.", "N/A")

    #sets the value of the variable to True

    validity = True

    #Checks if value is a string

    try:

        event\_name = str(event\_name\_entry.get())

    except ValueError:

        validity = False

        error\_window\_8.create()

    #Checks if value is an integer

    try:

        number\_of\_people = int(number\_of\_people\_entry.get())

    except ValueError:

        validity = False

        error\_window\_8.create()

    #Checks if a value is a string

    try:

        cattering\_requirements = str(cattering\_requirements\_entry.get())

    except ValueError:

        validity = False

        error\_window\_8.create()

    #does a final check if data is valid

    if validity == True:

        #Formats date and time

        day = str(day\_dropdown.get())

        month = str(month\_dropdown.get())

        year = str(year\_dropdown.get())

        date = year + "-" + month + "-" + day

        minute = str(minute\_dropdown.get())

        hour = str(hour\_dropdown.get())

        time = hour + ":" + minute

        #Calls Function

        are\_you\_sure(left\_frame,middle\_frame,right\_frame,event\_name,number\_of\_people,date,time,type\_of\_catering\_dropdown,cattering\_for\_dropdown,cattering\_requirements,event\_menu\_listbox,event\_dietary\_types\_listbox)

    else:

        #Runs item through class

        error\_window\_8.create()

#Defines function that checks if you are sure you want to proceed

def are\_you\_sure(left\_frame,middle\_frame,right\_frame,event\_name,number\_of\_people,date,time,type\_of\_catering\_dropdown,cattering\_for\_dropdown,cattering\_requirements,event\_menu\_listbox,event\_dietary\_types\_listbox):

    #Clears all widgits

    for widget in left\_frame.winfo\_children():

        widget.destroy()

    for widget in middle\_frame.winfo\_children():

        widget.destroy()

    for widget in right\_frame.winfo\_children():

        widget.destroy()

    #Creates a label

    event\_name\_label = ctk.CTkLabel(left\_frame, text=f"Event Name Is: {event\_name}", font= standard\_font)

    event\_name\_label.grid(row = 0, column = 0, pady = 10)

    #Creates a label

    number\_of\_people\_label = ctk.CTkLabel(left\_frame, text=f"Number Of People Is: {number\_of\_people}", font= standard\_font)

    number\_of\_people\_label.grid(row = 0, column = 0, pady = 10)

    #Creates a label

    date\_label = ctk.CTkLabel(left\_frame, text=f"Event Date Is: {date}", font= standard\_font)

    date\_label.grid(row = 0, column = 0, pady = 10)

    #Creates a label

    time\_label = ctk.CTkLabel(left\_frame, text=f"Event Time Is: {time}", font= standard\_font)

    time\_label.grid(row = 0, column = 0, pady = 10)

    #Creates a label

    type\_of\_catering\_label = ctk.CTkLabel(left\_frame, text=f"Type of catering is: {type\_of\_catering\_dropdown}", font= standard\_font)

    type\_of\_catering\_label.grid(row = 0, column = 0, pady = 10)

    #Creates a label

    catering\_for\_label = ctk.CTkLabel(left\_frame, text=f"Catering For: {cattering\_for\_dropdown}", font= standard\_font)

    catering\_for\_label.grid(row = 0, column = 0, pady = 10)

    #Creates a label

    special\_catering\_requirements\_label = ctk.CTkLabel(left\_frame, text=f"Special Catering Requirements: {cattering\_requirements}", font= standard\_font)

    special\_catering\_requirements\_label.grid(row = 0, column = 0, pady = 10)

    #Creates a label

    menu\_items\_label = ctk.CTkLabel(middle\_frame, text="Menu Items Are:", font= standard\_font)

    menu\_items\_label.grid(row = 0, column = 0, pady = 10)

    #Creating list box frame

    listbox\_frame= ctk.CTkFrame(middle\_frame, fg\_color= "#292929")

    listbox\_frame.grid(row = 1, column = 3)

    #Getting data from old list box

    data = event\_menu\_listbox.get(0, END)

    #Creates a listbox and inserts data from the old listbox into it

    event\_menu\_listbox = Listbox(listbox\_frame, bg= "#292929", fg= "Silver", width= 30, height= 12, font= standard\_font)

    for row in data:

        event\_menu\_listbox.insert(END, row)

    event\_menu\_listbox.pack(side=LEFT)

    #Creates a scroll bar

    listboxScrollbar= ctk.CTkScrollbar(listbox\_frame, command=event\_menu\_listbox.yview)

    listboxScrollbar.pack(side="right", fill=Y)

    event\_menu\_listbox.config(yscrollcommand=listboxScrollbar.set)

    #Creates a label

    dietary\_requirements\_label = ctk.CTkLabel(middle\_frame, text="Dietary Requirements Are:", font= standard\_font)

    dietary\_requirements\_label.grid(row = 0, column = 0, pady = 10)

    #Creating list box frame

    listbox\_frame\_two= ctk.CTkFrame(middle\_frame, fg\_color= "#292929")

    listbox\_frame\_two.grid(row = 1, column = 3)

    #Gets data from old listbox

    data\_two = event\_dietary\_types\_listbox.get(0, END)

    #Creates a listbox and inserts data from the old listbox into it

    event\_dietary\_types\_listbox = Listbox(listbox\_frame\_two, bg= "#292929", fg= "Silver", width= 30, height= 12, font= standard\_font)

    for row in data\_two:

        event\_menu\_listbox.insert(END, row)

    event\_dietary\_types\_listbox.pack(side=LEFT)

    #Creates a scroll bar

    listboxScrollbar= ctk.CTkScrollbar(listbox\_frame\_two, command=event\_menu\_listbox.yview)

    listboxScrollbar.pack(side="right", fill=Y)

    event\_menu\_listbox.config(yscrollcommand=listboxScrollbar.set)

    #Creating a button

    remove\_button = ctk.CTkButton(

        right\_frame,

        text= "Remove Item",

        font= standard\_font,

        width= standard\_width,

        height= standard\_height,

        command= lambda: cattering\_request(left\_frame,middle\_frame,right\_frame),

        )

    remove\_button.grid(row = 6, column = 4, pady = 10)

    #Creating a button

    remove\_button = ctk.CTkButton(

        right\_frame,

        text= "Remove Item",

        font= standard\_font,

        width= standard\_width,

        height= standard\_height,

        command= lambda: confirm\_request(event\_name,number\_of\_people,date,time,type\_of\_catering\_dropdown,cattering\_for\_dropdown,cattering\_requirements,event\_menu\_listbox,event\_dietary\_types\_listbox)

        )

    remove\_button.grid(row = 6, column = 4, pady = 10)

#Defines function that adds the new information to the database

def confirm\_request(event\_name,number\_of\_people,date,time,type\_of\_catering\_dropdown,cattering\_for\_dropdown,cattering\_requirements,event\_menu\_listbox,event\_dietary\_types\_listbox):

    #Gathers all the required information and formats it.

    type\_of\_catering = str(type\_of\_catering\_dropdown.get())

    if type\_of\_catering == "On Sight-Catering":

        type\_of\_catering = 1

    elif type\_of\_catering == "Off Sight-Catering":

        type\_of\_catering = 2

    catering\_for = str(cattering\_for\_dropdown.get())

    if catering\_for == "Breakfast":

        catering\_for = 1

    elif catering\_for == "Morning Tea":

        catering\_for = 2

    elif catering\_for == "Lunch":

        catering\_for = 3

    elif catering\_for == "Afternoon Tea":

        catering\_for = 4

    elif catering\_for == "Dinner":

        catering\_for = 5

    event\_menu = event\_menu\_listbox.get(0, END)

    event\_dietary\_types = event\_dietary\_types\_listbox.get(0, END)

    #Inserts information into the database

    cursor = connection.cursor()

    cursor.execute(f"INSERT INTO Event (EventName,NumberOfPeople,Date,Time) VALUES ('{event\_name}','{number\_of\_people}','{date}','{time}')").fetchall()

    connection.commit()

    #Extracts required information from the database

    cursor = connection.cursor()

    eventID = str(cursor.execute(f"SELECT EventID FROM Event ORDER BY EventID DESC LIMIT 1").fetchone()).replace("(","").replace(",","").replace(")","")

    cursor = connection.cursor()

    userID = str(cursor.execute(f"SELECT UserID FROM User WHERE Username = '{username}'").fetchone()).replace("(","").replace(",","").replace(")","")

    cursor = connection.cursor()

    cardID = str(cursor.execute(f"SELECT CardID FROM CardDetails WHERE UserID = '{userID}'").fetchone()).replace("(","").replace(",","").replace(")","")

    #Inserts information into the database

    cursor = connection.cursor()

    cursor.execute(f"INSERT INTO Catering (UserID,MealID,EventID,CardID,TypesOfCateringID,SpecificCateringRequests) VALUES ('{userID}','{catering\_for}','{eventID}','{cardID}','{type\_of\_catering}','{cattering\_requirements}')").fetchall()

    connection.commit()

    #Extracts information from the database

    cursor = connection.cursor()

    cateringID = str(cursor.execute(f"SELECT CateringID FROM Catering WHERE EventID = '{eventID}'").fetchone()).replace("(","").replace(",","").replace(")","")

    #Adds the menu items requested for the event into the database

    for item in event\_menu:

        item = str(item).split(" ")

        menuID = item[0]

        cursor = connection.cursor()

        cursor.execute(f"INSERT INTO EventMenu (CateringID,MenuID) VALUES ('{cateringID}','{menuID}')").fetchall()

        connection.commit()

    #Adds the dietary requirements for the event into the database

    for item in event\_dietary\_types:

        item = str(item).split(" ")

        dietary\_typeID = item[0]

        dietary\_type\_qty = item[2]

        cursor = connection.cursor()

        cursor.execute(f"INSERT INTO EventDietaryRequirements (DietaryTypesID,CateringID,Qty) VALUES ('{dietary\_typeID}','{cateringID}','{dietary\_type\_qty}')").fetchall()

        connection.commit()

    #Calls the function requester\_gui

    requester\_gui()

#Defines function that creates the deativated account gui

def deactivated\_account():

    #Creates the main frame for the widgits

    main\_frame = ctk.CTkFrame(root)

    main\_frame.pack(fill="both", expand=True)

    #Clears all the widgits

    for widget in main\_frame.winfo\_children():

        widget.destroy()

    #Creating a label

    label = ctk.CTkLabel(main\_frame, text="Your account has been deactivated. \nPlease go talk to a catering staff member about the issue.", fg\_color="transparent", font= standard\_font)

    label.pack(pady = standard\_y\_padding)

    #Creates a button

    close\_button = ctk.CTkButton(

        main\_frame,

        text="Close Window",

        font=standard\_font,

        width=standard\_width,

        height=standard\_height,

        command=root.destroy)

    close\_button.pack(pady=standard\_y\_padding)

#Launches the programs code

if \_\_name\_\_ == "\_\_main\_\_":

    main(button\_type = None)

#Makes the tkinter code work

root.mainloop()

# Part 2 Evaluate

## SQL Queries Used to Manipulate Data

**SELECT** /\*This query returns the users account information\*/

**User**.FirstName ||' '|| **User**.LastName **AS** **User**,

AccountType.AccountTypeName **AS** Account\_Type,

**User**.Username **AS** Username,

**User**.Password **AS** Password

**FROM** **User**

**INNER** **JOIN** AccountType

**ON** AccountType.AccountTypeID = **User**.AccountTypeID;

**SELECT** /\*This query returns the users contact details\*/

**User**.FirstName ||' '|| **User**.LastName **AS** **User**,

**User**.PhoneNumber **AS** Phone,

**User**.Email **AS** Email

**FROM** **User**;

**SELECT** /\*This query returns the name of the meal and the price per head\*/

Meals.MealName **AS** Meal,

Meals.MealPrice **AS** Price

**FROM** Meals;

**SELECT** /\*This query returns the types of catering\*/

TypesOfCatering.TypesOfCateringName

**FROM** TypesOfCatering;

**SELECT** /\*This query returns the card status options\*/

CardStatus.CardStatusName **AS** Card\_Status

**FROM** CardStatus;

**SELECT** /\*This query returns the card details of a user\*/

**User**.FirstName ||' '|| **User**.LastName **AS** **User**,

CardDetails.CardNumber **AS** Card\_Number,

CardDetails.ExpiryDate **AS** Expiry\_Date,

CardDetails.CVV **AS** CVV,

CardStatus.CardStatusName **AS** Card\_Status

**FROM** CardDetails

**INNER** **JOIN** **User**

**ON** **User**.UserID = CardDetails.UserID

**INNER** **JOIN** CardStatus

**ON** CardStatus.CardStatusID = CardDetails.UserID;

**SELECT** /\*This query returns the event details and users contact detail for all events\*/

**User**.FirstName ||' '|| **User**.LastName **AS** **User**,

Event.EventName **AS** Event,

**User**.PhoneNumber **AS** Phone,

**User**.Email **AS** Email,

Event.NumberOfPeople **AS** Number\_of\_People,

Event.**Date** **AS** **Date**,

Event.**Time** **AS** **Time**

**FROM** Catering

**INNER** **JOIN** **User**

**ON** **User**.UserID = Catering.UserID

**INNER** **JOIN** Event

**ON** Event.EventID = Catering.EventID;

**SELECT** /\*This query returns the menu options\*/

Menu.ItemName **AS** Item

**FROM** Menu;

**SELECT** /\*This query returns the event menu\*/

Event.EventName **AS** Event\_Name,

Menu.ItemName **AS** Item

**FROM** Event

**INNER** **JOIN** Catering

**ON** Event.EventID = Catering.EventID

**INNER** **JOIN** EventMenu

**ON** Catering.CateringID = EventMenu.CateringID

**INNER** **JOIN** Menu

**ON** EventMenu.MenuID = Menu.MenuID

**WHERE** Event.EventName = "Book\_Fair"

**ORDER** **BY** Event.EventName, Menu.ItemName;

**SELECT** /\*This query returns the dietary types saved in the database\*/

DietaryTypes.DietaryTypesName **AS** Dietary\_Types

**FROM** DietaryTypes;

**SELECT** /\*This query returns the event dietary types\*/

Event.EventName **AS** Event\_Name,

DietaryTypes.DietaryTypesName **AS** Dietary\_Types,

EventDietaryRequirements.Qty **AS** Qty

**FROM** Event

**JOIN** Catering

**ON** Event.EventID = Catering.EventID

**JOIN** EventDietaryRequirements

**ON** Catering.CateringID = EventDietaryRequirements.CateringID

**JOIN** DietaryTypes

**ON** EventDietaryRequirements.DietaryTypesID = DietaryTypes.DietaryTypesID

**WHERE** Event.EventName = "Sports\_Day"

**ORDER** **BY** Event.EventName, DietaryTypes.DietaryTypesName, EventDietaryRequirements.Qty;

**SELECT** /\*This query returns the catering information for all events\*/

**User**.FirstName ||' '|| **User**.LastName **AS** **User**,

Meals.MealName **AS** Meal,

Event.EventName **AS** Event\_Name,

Catering.CardID **AS** Card\_ID,

TypesOfCatering.TypesOfCateringName **AS** Type\_Of\_Catering,

Catering.SpecificCateringRequests **AS** Specific\_Catering\_Requests

**FROM** Catering

**JOIN** "User"

**ON** **User**.UserID = Catering.UserID

**JOIN** Meals

**ON** Meals.MealID = Catering.MealID

**JOIN** Event

**ON** Event.EventID = Catering.EventID

**JOIN** TypesOfCatering

**ON** TypesOfCatering.TypesOfCateringID = Catering.TypesOfCateringID;

## Reflection

As I look back at my solution to the catering problem, I feel I have succeeded at creating a program that allows users to create catering requests for events and it allows catering staff to see all the key information they need in order to cater for these events. However, it is in a very basic form and likely filled with bugs. So, although it does meet the requirements set out by the customer it does this barely and many improvements can be made to improve the code.

The database had some major changes from the original plan in the ER-Diagram. The entity relationships stayed the same in the database however a new table was added. this new table was called Card status and was made so that if a card expired and was no longer valid. So instead of deleting it, it would be set to inactive and wouldn’t show up. Some of the columns in the tables was changed to accommodate more information or new tables. Such as adding card status id to the card details table.

I did not add many extra features as it is a very simple and basic program however, I did add a GUI that makes it easier to look at and navigate. I also added a long in feature that shows different information for normal users who can only see the catering requests and caters who can see all the event information and catering information. There is also a deactivated account setting in case someone account needs to be removed.

There are no bugs that I have found in the program however there are some limitations. Users can select up to 31 days in every month including the ones that only have 30 days. This could cause some confusion and some errors further on down the track. If the user does not have custom tkinter installed the program will not work and will error out. If the user does not have the database in the same directory as the python file, the program will not be able to connect to the database and error out. If the user in the catering staff gui does not select in some of the menus it will create an error however this error does not break the program or stop it from running. If the user, then selects something after this mistake the program will continue as if nothing happened. For the requester gui the program does make sure that the correct values are being used however if no values are entered in some of the entries, then the program will not recognize this and continue as if nothing happened leading to issues when inserting data into the database. It also does not make sure a user has selected something in the dropdown menu also leading to some errors when inserting data into the database. The request can also go through if no dietary requirements where selected and if no menu items were selected meaning caters won’t have any information on what food they are supposed to be catering with. There is no changing your request or cancelling it before it has been completed.

## Developer retrospect

The data base works well and stores all required information without any issues. It handles when data should be deactivated instead of deleting it and it doesn’t duplicate information. The GUI achieves its goal of making it easy to navigate through the program and makes it easy to use.

The actual logic of the program works but barely and is very limited in what it could do now compared to what I wanted it to do. There as a are you sure window that displayed all the user information for them to look at after entering a request before they did I final confirmation and it got added to the database. However, it was causing lots of headaches and due to the time from I decided to remove this aspect of the code.

Next time I would work on my time management a bit better and spend longer on certain aspects and less time on others. I would also do more of the project at home as time was a major factor. Next time I would also go without the gui and get the user to interact with the program through the terminal. This is because the gui aspect of the program took up the most time leaving little time for the program’s logic.

## Sources

*Chat GPT*.

*One Note*.

Staff, Developer com. “Python: How to Use Tkinter’s Grid Manager.” *Developer.Com*, 24 Mar. 2022, <https://www.developer.com/languages/python/tkinter-grid-manager/>.

## AI Conversation

A screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated A screen shot of a computer code

Description automatically generated A screenshot of a computer

Description automatically generated A screen shot of a computer

Description automatically generated A screen shot of a computer

Description automatically generated A screenshot of a computer

Description automatically generated A screenshot of a computer

Description automatically generated A screen shot of a computer

Description automatically generated A screenshot of a computer

Description automatically generated A screenshot of a computer screen

Description automatically generated A screenshot of a computer

Description automatically generated