David O’Donnell
Ellen Tunison
Ryan Sadowski
Nick Furlo

Restaurant Management Database - Phase 3

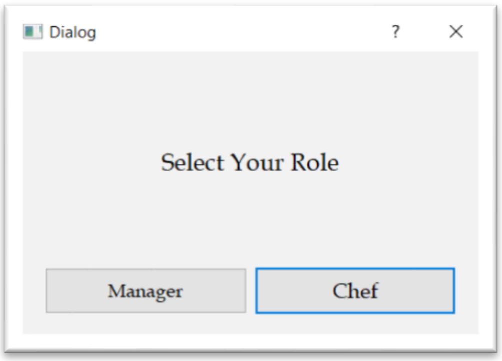
Table of Contents

User Manual

The application will be stored and ran as an executable file that is stored locally on the chef/manager's computer. Alternatively, install Python and PyQt and run logWindow.py with a python supporting IDE. (Pip install PyQt5)

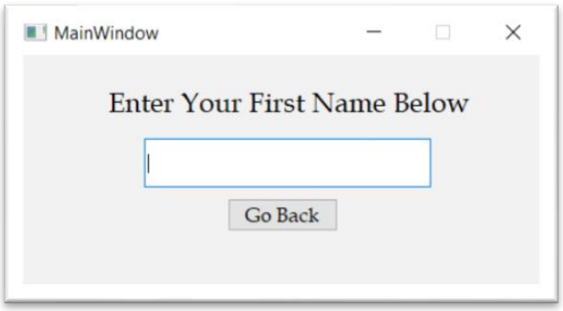
Logging In

The Restaurant Management Database System consists of a Graphical User Interface (GUI) where users can login via selecting two roles: Manager and Chef



Chef

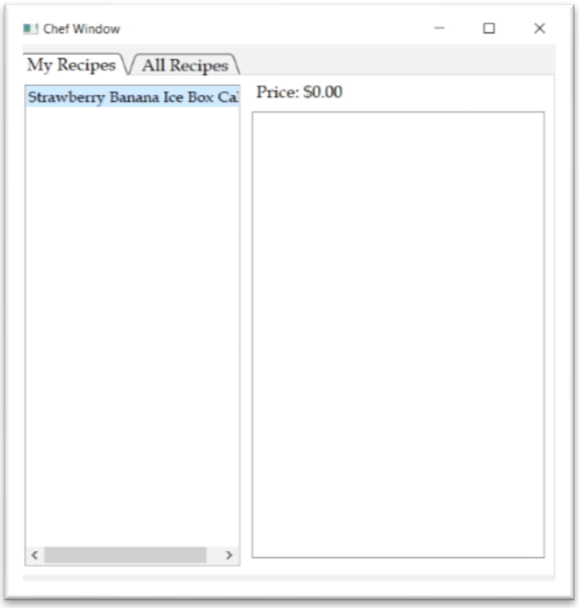
The functionality between the Manager and Chef are different, the chef can view their known recipes, as well as all the recipes in the database. After selecting "chef" from the login window, you will be brought to the following screen:



Enter the chef's first name you are logging in as or create a new chef by logging in as a Manager. Once logged in as a Chef, the user will be brought to a new window with the following features:

- Under the "My Recipes" tab
    - View their recipes in the left-pane under "My recipes".

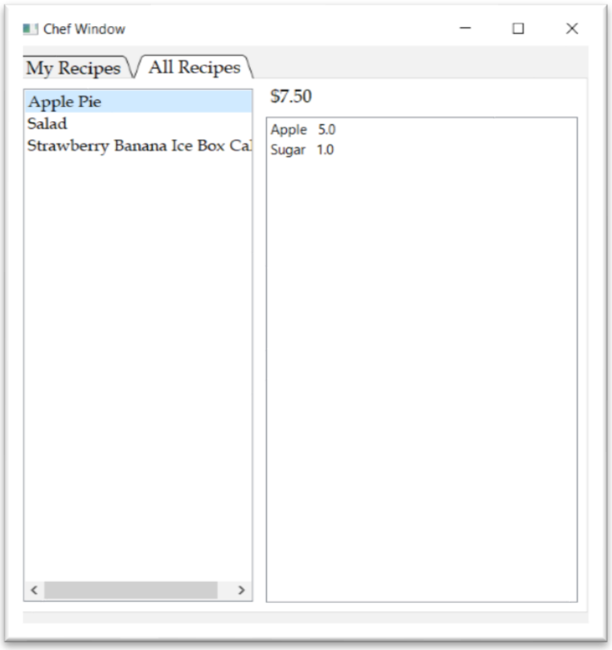
        - Clicking on a recipe will show the ingredients and calculated price in the right-pane.



Strawberry Banana Ice Box Ca

Price: $0.00

- Under the “All Recipes” tab
    - View all recipes of the database in the left-pane.
        - Clicking on a recipe will show the ingredients and calculated price in the right-pane.



Apple Pie
Salad
Strawberry Banana Ice Box Ca

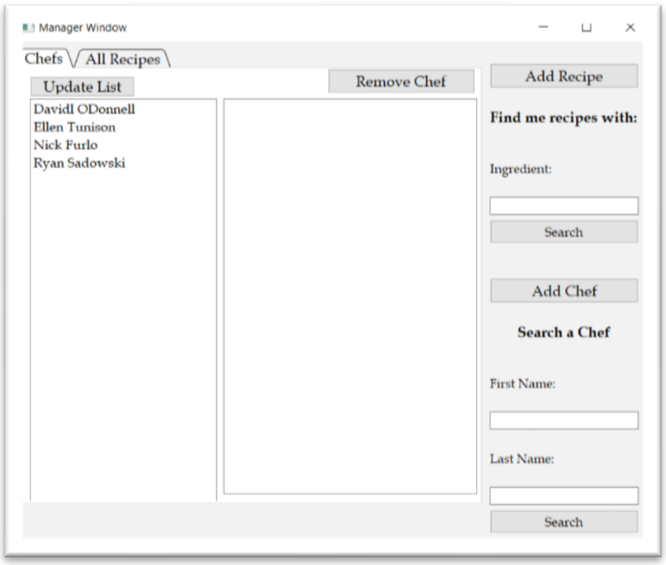
$7.50

```
Apple   5.0
Sugar   1.0
```

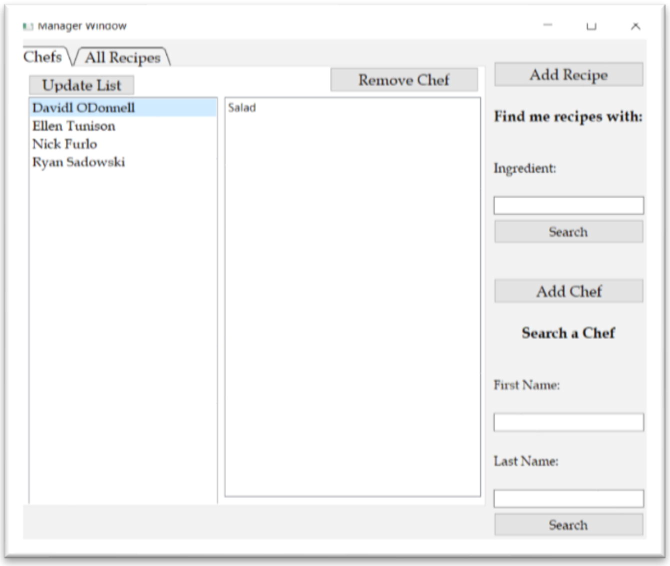
Manager

Once logged in as a Manager, the user will be brought to a new window with the following features:

- Under the “Chefs” tab
    - View every chef in the database in the left-plane
        - Clicking on a chef in the left-pane will display the recipes that the chef knows in the right-pane, this may help a manager make a schedule for the week or update the menu for the night.



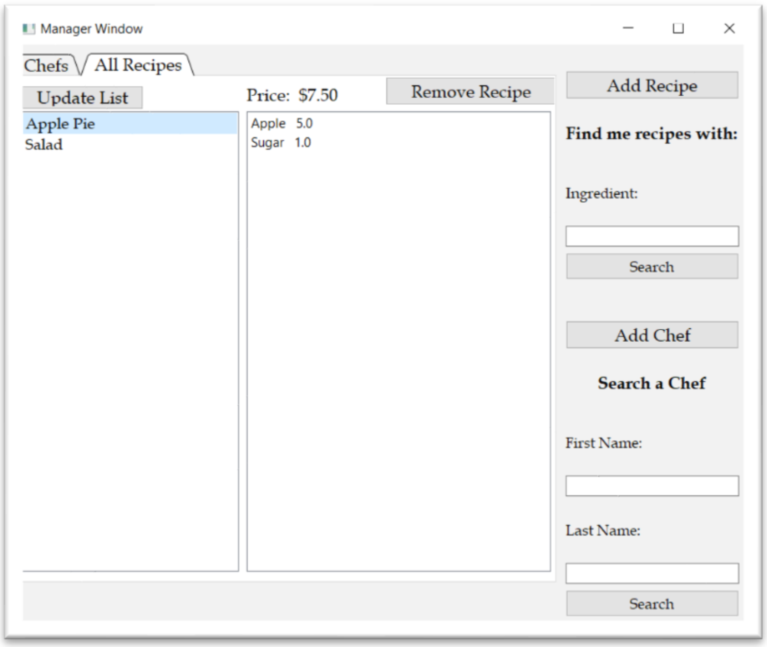
Davidl ODonnell
Ellen Tunison
Nick Furlo
Ryan Sadowski



Davidl ODonnell
Ellen Tunison
Nick Furlo
Ryan Sadowski

Salad

- Under the “All Recipes” tab
    - View all recipes of the database in the left-pane.
        - Clicking on a recipe will show the ingredients and calculated price in the right-pane.



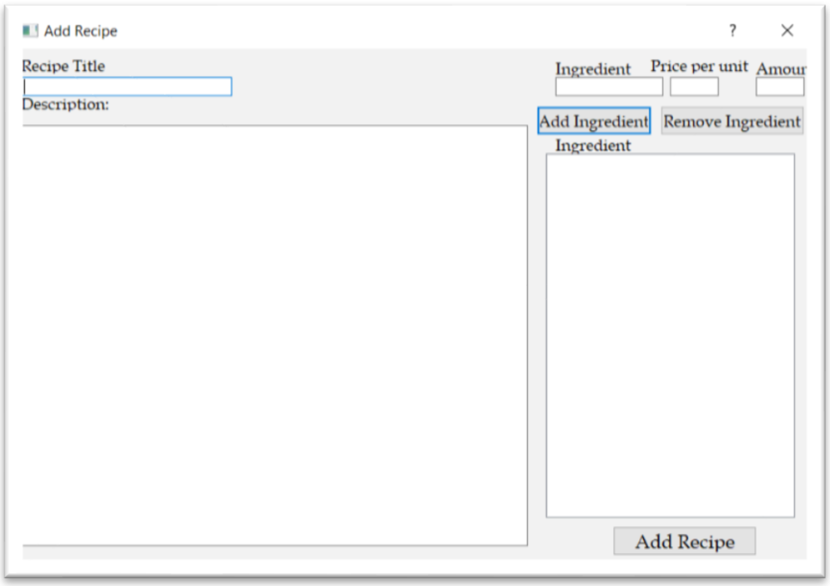
Price: $7.50

Apple Pie
Salad

Apple 5.0
Sugar 1.0

- Add Recipe button
    - A new window will pop-up

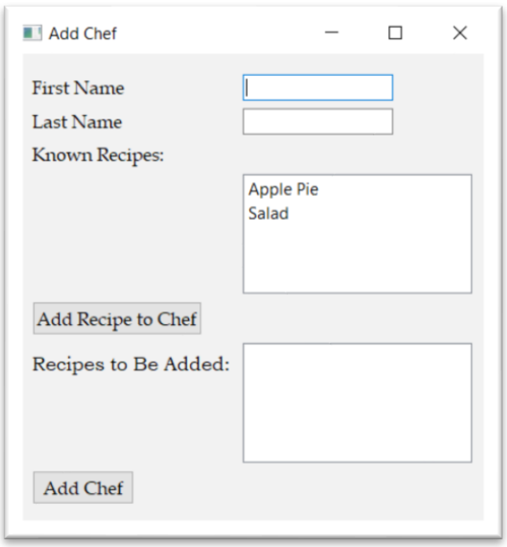
        - The “Recipe Title” should reflect a proper name for your new recipe.
        - “Description” should be a detailed summary of your recipe.
        - Ingredients can be added to the recipe by inputting proper values for Ingredient, Price per unit and Amount.
        - Once added the Ingredient list will be populated with the newly created ingredient.
        - Click “Add Recipe” once all fields are properly satisfied.
        - See the “Sample Output” section for more details.



- Remove Recipe button
    - Highlight the desired recipe you wish to delete and select the button to remove it from the database and the user interface.
        - See example under the “Sample Output” section.

- Searching for a Recipe button
    - Input ingredient names (up-to-three at a time) and click search to search all recipes with the search criteria.
        - See “Sample Output”.

- Add a Chef
    - A new window will pop-up
        - Provide a first and last name for the new Chef in the proper fields.
        - Choose from a drop-down of recipes already incorporated into the database system to specify recipes these Chef’s already know. Once Chosen, they can be seen in the box directly below “Known Recipes”.
        - Click “Add” once all fields are properly satisfied.
        - See “Sample Output”.



- Searching for a Chef
    - Input the Chef's first and last name and click search to search all chefs with the search criteria.
        - See "Sample Output".

<u>Implementation Manual</u>

Problem Statement

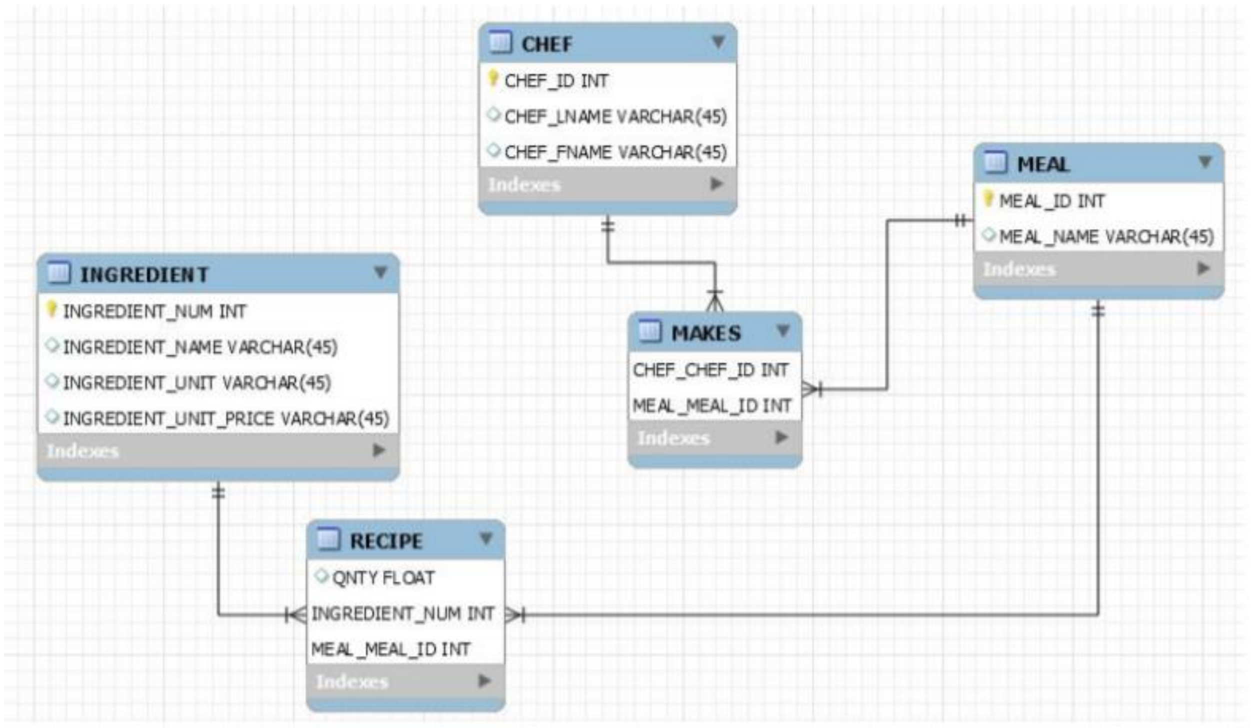
Our application will help restaurants manage their recipes as well as what recipes each chef is assigned to. This system will help managers and chefs quickly lookup recipes, their ingredients, costs, and which chef will be making that dish. A database must be used in order to implement the relationships between recipes and chefs.

System Requirements

This application will be used by both managers and chefs. Managers and chefs will be able to add and edit recipes, and change ingredient prices. Managers will also be able to assign recipes to chefs. The program will be able to calculate how much each specific recipe will cost to make

Conceptual Database Design

The relationship in our database can be described by the following business rules. "Each meal may use many ingredients ingredients." "Each ingredient may be used in many meals, but may not be used in any meals". "Each chef can make many meals." "Each meal can be made by many chefs." The following diagram is an ERD for our conceptual database design.



Functional Requirements

Adding a new recipe:
-Make sure the recipe doesn't already exist, etc.

Removing a recipe:
-Remove the recipe from the database without removing the ingredients that are in the recipe.

Adding Ingredients:
-Make sure the ingredient doesn't already exist etc.

Searching for a recipe:
-Search for the recipe by name

Calculating price of a recipe:
-Find the sum of the recipes based on the amount of units of each ingredient and the unit price of each ingredient.

Assign recipes to chefs:
-Manager can assign chefs to make a certain recipe dependent on the types of recipe a chef knows.

Logical Database Design

Relation: CHEF(CHEF_ID, CHEF_FNAME, CHEF_LNAME)
Primary keys: CHEF_ID
Foreign keys: N/A
Functional dependencies: CHEF_FNAME and CHEF_LNAME are functionally dependent on CHEF_ID because CHEF_ID uniquely determines CHEF_FNAME and CHEF_LNAME.

Relation: MAKES(CHEF_ID, MEAL_ID)
Primary keys: CHEF_ID, MEAL_ID
Foreign keys: CHEF_ID, MEAL_ID
Functional dependencies: N/A

Relation: MEAL(MEAL_ID, MEAL_NAME)
Primary keys: MEAL_ID
Foreign keys: N/A
Functional dependencies: MEAL_NAME is functionally dependent of MEAL_ID because MEAL_ID uniquely determines MEAL_NAME

Relation: RECIPE(INGREDIENT_NUM, MEAL_ID, QNTY)
Primary keys: INGREDIENT_NUM, MEAL_ID
Foreign keys: INGREDIENT_NUM, MEAL_ID
Functional dependencies: N/A

Relation: INGREDIENT(INGREDIENT_NUM, INGREDIENT_NAME, INGREDIENT_UNIT, INGREDIENT_UNIT_PRICE)
Primary keys: INGREDIENT_NUM
Foreign keys: N/A
Functional dependencies: INGREDIENT_NAME, INGREDIENT_UNIT, and INGREDIENT_UNIT_PRICE are functionally dependent on INGREDIENT_NUM because INGREDIENT_NUM uniquely determines INGREDIENT_NAME, INGREDIENT_UNIT, and INGREDIENT_UNIT_PRICE.

Application Programs

Adding a new recipe:

Input: Recipe name
Output: A window saying "Recipe successfully added" or "Recipe already exists"
Other functions referenced: Adding ingredients
Psuedocode:

```
add_rec(name)
    Make a query into the MEAL table to see if there is already a recipe with attribute
    MEAL_NAME = name.
    If the query returns an attribute, return a window saying "Recipe already exists"
```

```
        If the query does not return an attribute, prompt user to either choose ingredients
        from the existing ingredients(along with their quantities), or add a new
        ingredient(s)(and ask for its quantity.
        Add this meal to the MEAL table
        Add associating entities in the RECIPE table
        Return a window saying "Recipe successfully added"
```

SQL queries:

```
SELECT *  FROM MEAL WHERE MEAL_NAME = name;
INSERT INTO MEAL VALUES(name); (This assumes that MEAL_ID is auto-
incrementing)
INSERT INTO RECIPE VALUES(SELECT MEAL_ID FROM MEAL WHERE
MEAL_NAME = name, SELECT INGREDIENT_ID FROM INGREDIENT WHERE
INGREDIENT_NAME = ingredient); (This needs to be done for each ingredient in the
meal.)
```

Removing a recipe:

Input : Name of recipe to be removed.
Output: A window saying "Recipe has been removed" or "Recipe already did not exist"
Psuedocode:

```
rem_rec(name)
    Query the MEAL table for MEAL_NAME = name
    If the query does not return anything, return a window saying "Recipe already did
    not exist"
    otherwise
    For record in RECIPE where MEAL_ID is associated with the MEAL_NAME
        Remove the record
    Remove the entity from the MEAL table where MEAL_NAME = name
```

SQL queries:

```
SELECT * FROM MEAL WHERE MEAL_NAME = name;
DELETE FROM RECIPE WHERE MEAL_ID = SELECT MEAL_ID FROM MEAL
WHERE MEAL_NAME = name;
DELETE FROM MEAL WHERE MEAL_NAME = name;
```

Adding Ingredients:

Input: ingredient name, ingredient price, ingredient unit
Output: A window saying "Ingredient was added" or "Ingredient was already here"
Psuedocode:

```
add_ing(name, price, unit)
    Query the INGREDIENT table for INGREDIENT_NAME = name
```

```
        If the query returns something, return a window saying "Ingredient was already
        here"
        Otherwise
        Add a new entity into the INGREDIENT table and return a window saying
        "Ingredient added"
```

SQL queries:

```
SELECT * FROM INGREDIENT WHERE INGREDIENT_NAME = name;
INSERT INTO INGREDIENT VALUES(name, price, unit); (This assumes that
ingredient_id is auto-incrementing)
```

Searching for a recipe:

Input: Recipe name
Output: A window displaying the recipe
Pseudocode:

```
find_rec(name)
        Query the MEAL table for MEAL_NAME = name
        If the query does not return anything, return a window saying "Recipe not found"
        Otherwise
        Display the recipe
```

SQL queries:

```
SELECT * FROM MEAL WHERE MEAL_NAME = name
```

Calculating price of a recipe:

Input: array of number of ingredient units, array of ingredient unit prices
Output: A window showing the proper recipe price based on the current inputs.
Pseudocode:

```
calc_recipe_price(recipe_name)
    Query the RECIPE table for INGREDIENT_NUM = ingredient_num
    Query the RECIPE table for QUANTITY = recipe_quantity
    Query the INGREDIENT table for INGREDIENT_UNIT_PRICE = unit_price
    Query the INGREDIENT table for INGREDIENT_UNIT = unit
        If unit_price is null, display window "Ingredient does not exist in the
        table, please add the ingredient before continuing."
    Otherwise
    price = quantity * unit_price * unit
    Display a window saying "total price: " + price
```

SQL Queries:

```
SELECT * FROM INGREDIENT WHERE INGREDIENT_NUM =
ingredient_num
```

Assign recipes to chefs:

Input: recipe, chef's name
Output: window saying "Recipe assigned to [chef's name]" or "Recipe does not exist" or "Chef does not exist"
Pseudocode:

```
assign_rec(rec, chef)
    Query the CHEF table for CHEF_NAME = chef
    If the query does not return anything, return a window saying "Chef does not exist"
    Otherwise
    Query the MEAL table for MEAL_NAME = rec
    If the query does not return anything, return a window saying "Recipe does not exist"
    Otherwise
    Query the MAKES table for an association between chef and recipe
    If the query returns something, return a window saying "This chef is already
    assigned this recipe"
    otherwise
    Create an entity in the MAKES table that associates chef and recipe
```

SQL query:

```
SELECT * FROM CHEF WHERE CHEF_NAME = chef;
SELECT* FROM MEAL WHERE MEAL_NAME = rec;
    SELECT * FROM MAKES WHERE (MEAL_ID = SELECT MEAL_ID
    FROM MEAL WHERE MEAL_NAME = rec) AND (CHEF_ID =
    SELECT CHEF_ID FROM CHEF WHERE CHEF_NAME = chef);
```

Generate an ID:

Input: an array of existing ID's
Output: a new array of existing ID's
Pseudocode:

```
idGen(existingIds)
    Assign new id to random int between 1 and 1000.
    If the id is already in the existing ID:
        Recursive call
    Otherwise
        Return the array of new ID's
```

Check if chef's name already exists:

```
Input: name
Output: Boolean
Pseudocode:

checkChefName(name)
    Query the Chef table for all names matching the input name
    Store in array
    If array is empty:
        Assume the name doesn't exist already: return False
    Otherwise
        Return True

SQL query:
    SELECT * FROM CHEF WHERE CHEF_FNAME = name
```

Get all the chefs from the Chef table

```
Input: none
Output: array of chef names
Pseudocode:

getChefs()
    Query the Chef table for all names
    Store in array
    Return array

SQL query:
    SELECT CHEF_FNAME, CHEF_LNAME FROM CHEF
```

Get all the recipes that a chef knows

```
Input: Chef name
Output: array of meals the chef knows
Pseudocode:

getRecByChefName(name)
    Query the Chef table for the chef ID matching the name input
    Store the ID
    Query the Makes table for the meal IDs matching the Chef's ID
    Store the ids in an array
    Create array to hold all meal names
    Loop through mealIDs
        Query the Meal table for meal names matching the meal IDs
    Return array of meal names

SQL query:
    SELECT CHEF_ID FROM CHEF WHERE CHEF_FNAME = name
    SELECT MEAL_ID FROM MAKES WHERE CHEF_ID = chefid
    SELECT MEAL_NAME FROM MEAL WHERE MEAL_ID = mealid
```

Get ingredients belonging to a recipe

Input: recipe name
Output: arrays of ingredient names and ingredient quantities
Pseudocode:

```
getIngredients(recipe_name)
    Query the Meal table for Meal IDs matching the recipe name
    Store in an array
    Query the Recipe table for Ingredient number and quantity where Meal ID
    matches the meal id from above
    Store in an array
    Loop through array and return all ingredient numbers and quantities as arrays
    Loop through ingredients to get the ingredients name
        Query the Ingredient table for ingredient name where the numbers match
```

SQL query:

```
    SELECT MEAL_ID FROM MEAL WHERE MEAL_NAME = recipe_name
    SELECT INGREDIENT_NUM, QNTY FROM RECIPE WHERE MEAL_ID =
    meal_id
    SELECT INGREDIENT_NAME FROM INGREDIENT WHERE
    INGREDIENT_NUM = num
```

Check if ingredient already exists

Input: ingredient name, price per unit
Output: Boolean True if matching record found
Pseudocode:

```
checking(name, ppu)
    Query Ingredient table for ingredient name matching input
    Logic to handle different cases: return boolean
```

SQL query:

```
    SELECT INGREDIENT_NAME FROM INGREDIENT WHERE
    INGREDIENT_NAME = name
    SELECT FROM INGREDIENT WHERE INGREDIENT_NAME = name AND
    INGREDIENT_UNIT_PRICE = ppu
```

Get all recipes from the Meal table

Input: none
Output: array of recipe names
Pseudocode:

```
getAllRecipes()
    Query and return all recipes
```

SQL query:

```
    SELECT MEAL_NAME FROM MEAL
```

Add chef

Input: first name, last name, array of recipes
Output: none
Pseudocode:

```
addChef(fname, lname, recipes[])
    Query Chef table for CHEF_ID
    Generate new ids
    Insert into Chef table the new chef parameters
    Loop through recipes to add them accordingly
    Insert into Makes table to the recipes the chef knows
```

SQL query:

```
    SELECT CHEF_ID FROM CHEF
    INSERT INTO CHEF VALUES (new_ids[], lname, fname)
    SELECT MEAL_ID FROM MEAL WHERE MEAL_NAME = recipe_name
    INSERT INTO MAKES VALUES(recipe_ids[], new id)
```

Remove Chef

Input: first name, last name
Output: none
Pseudocode:

```
removeChef(fname, lname)
    Delete chef from Chef table where names match
```

SQL query:

```
    DELETE FROM CHEF WHERE CHEF_FNAME = fname AND
    CHEF_LNAME = lname
```

Get recipe description

Input: recipe name
Output: description cell from database
Pseudocode:

```
getDescription(recipe_name)
    Query Meal table for description where meal_name and recipe_name match
```

SQL query:

```
    SELECT DESCR FROM MEAL WHERE MEAL_NAME = recipe_name
```

Search for chef

Input: first name, last name
Output: chef
Pseudocode:

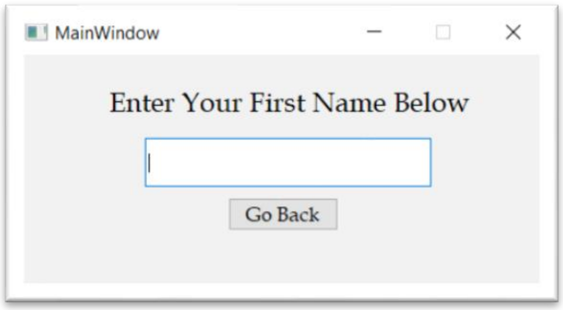
```
searchChef(first_name, last_name)
    Query Chef table for first and last name of chef where first and last name match
    Store matching records in an array as objects
    Return all matching
```

SQL query:

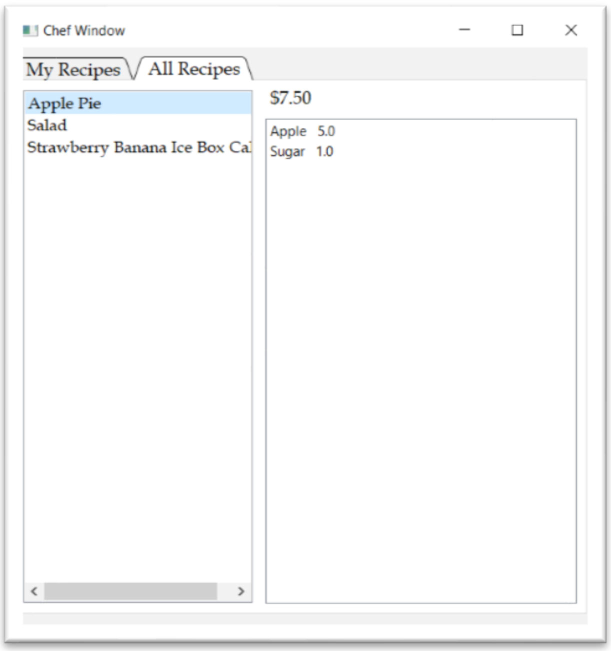
```
    SELECT CHEF_FNAME, CHEF_LNAME FROM CHEF WHERE
    CHEF_LNAME = last_name AND CHEF_FNAME = first_name
```

User Interface Design

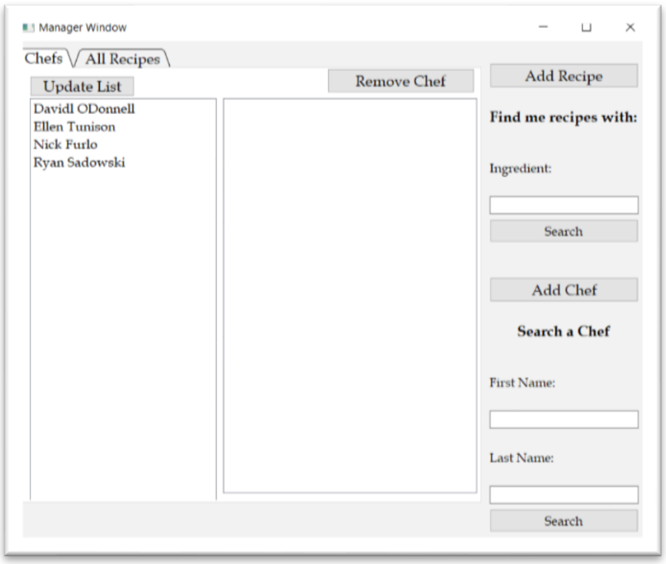
For the user interface, the user will begin with the role window, asking what their role in the restaurant is.



Upon choosing user role, one of two windows will appear. The first we will discuss is the chef view. The chef view will allow a particular chef to keep track of which recipes they know and view how much it will cost to make them. In the left box will be a list of recipes, and when a recipe is clicked, its contents will be shown in the right box, as well as how much it will cost to make that recipe.



In the Manager view, the manager can do everything a chef can do, except there is also a chef list that allows a manager to choose an existing chef and view which recipes they know. From the manager view, the manager can add a new chef and search for chefs. A manager can also add and delete recipes. In the Chef tab of the Manager view, the right left box will contain the list of chefs and when a chef is chosen, all of the recipes in their repertoire will be shown in the right box.



When the add recipe button is clicked, a new window opens. In this window, the user must enter the title and details of the recipe, each ingredient, how much is needed, and how much each ingredient costs per unit. For each ingredient entered, a list will be made indicating each ingredient and how much it will cost for that particular recipe.

The final window is the Add Chef window which can be accessed by clicking the "Add Chef" button in the manager view. This window will allow a manager to add a chef to the database. He or she will also be able to choose which recipes the new chef already knows by choosing from existing recipes located in the combo box. Each recipe chosen will be shown in a list that can be reviewed before adding the chef.

The User Manual section of this report goes into complete detail of the application functionality. See also the Sample Output section for examples of the application working.

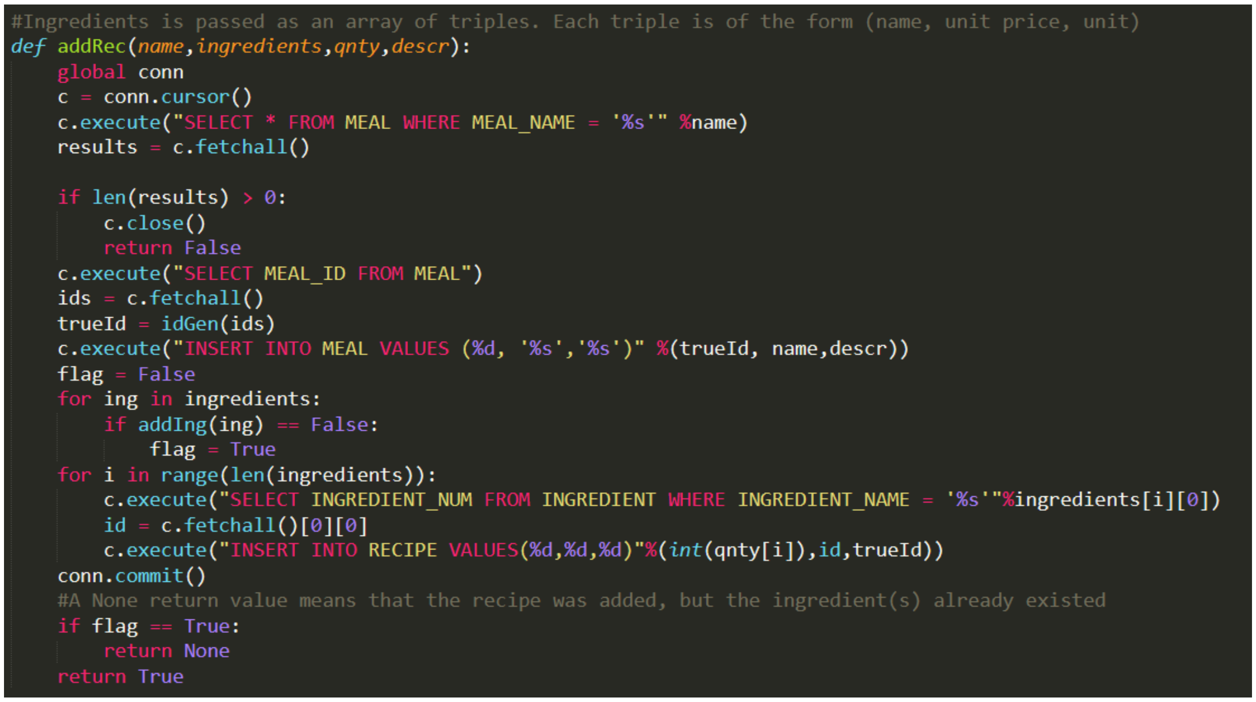
Implementation Plan

Adding a new recipe: David
Removing a recipe: Ellen
Adding Ingredients: Ryan
Searching for a recipe: Nick
Calculating price of a recipe: David & Ellen
Assign recipes to chefs: Ryan & Nick

Code Listing

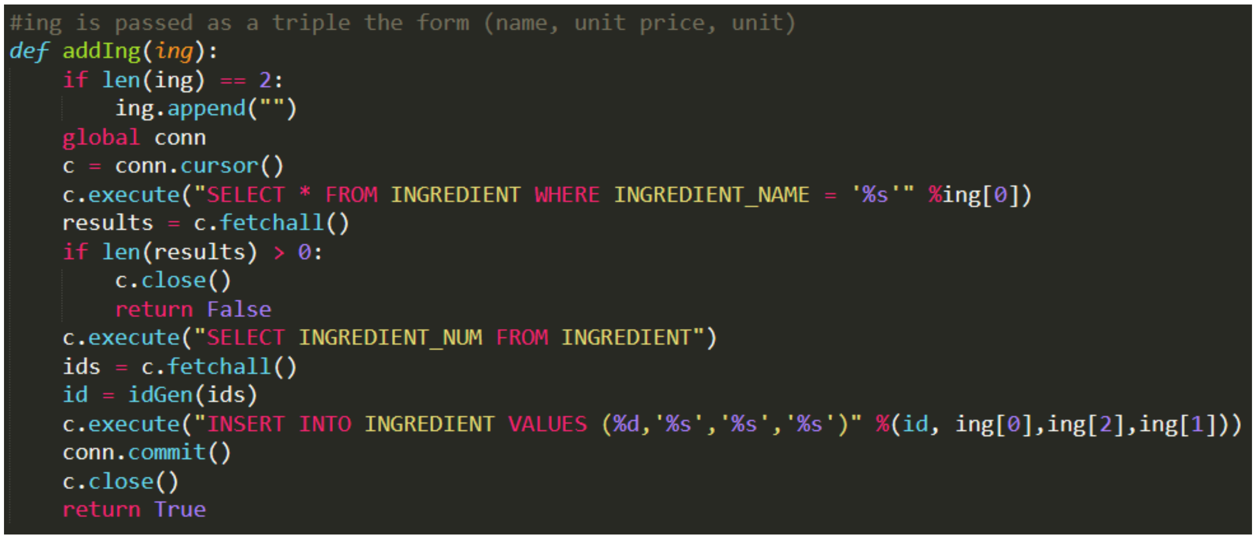
Listed is the code used to generate the SQL queries. For the complete code, see the contents of the attached zip file.

Adding a recipe



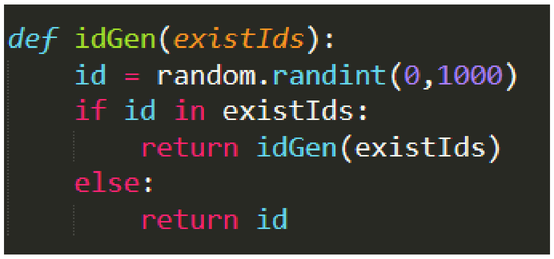
```python
#Ingredients is passed as an array of triples. Each triple is of the form (name, unit price, unit)
def addRec(name,ingredients,qnty,descr):
    global conn
    c = conn.cursor()
    c.execute("SELECT * FROM MEAL WHERE MEAL_NAME = '%s'" %name)
    results = c.fetchall()

    if len(results) > 0:
        c.close()
        return False
    c.execute("SELECT MEAL_ID FROM MEAL")
    ids = c.fetchall()
    trueId = idGen(ids)
    c.execute("INSERT INTO MEAL VALUES (%d, '%s','%s')" %(trueId, name,descr))
    flag = False
    for ing in ingredients:
        if addIng(ing) == False:
            flag = True
    for i in range(len(ingredients)):
        c.execute("SELECT INGREDIENT_NUM FROM INGREDIENT WHERE INGREDIENT_NAME = '%s'"%ingredients[i][0])
        id = c.fetchall()[0][0]
        c.execute("INSERT INTO RECIPE VALUES(%d,%d,%d)"%(int(qnty[i]),id,trueId))
    conn.commit()
    #A None return value means that the recipe was added, but the ingredient(s) already existed
    if flag == True:
        return None
    return True
```

Adding an ingredient



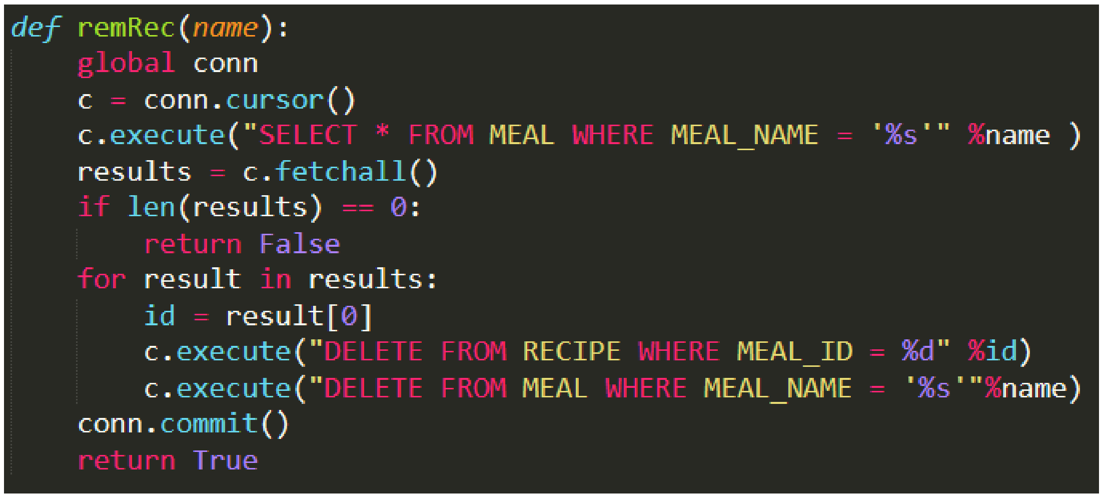
```python
#ing is passed as a triple the form (name, unit price, unit)
def addIng(ing):
    if len(ing) == 2:
        ing.append("")
    global conn
    c = conn.cursor()
    c.execute("SELECT * FROM INGREDIENT WHERE INGREDIENT_NAME = '%s'" %ing[0])
    results = c.fetchall()
    if len(results) > 0:
        c.close()
        return False
    c.execute("SELECT INGREDIENT_NUM FROM INGREDIENT")
    ids = c.fetchall()
    id = idGen(ids)
    c.execute("INSERT INTO INGREDIENT VALUES (%d,'%s','%s','%s')" %(id, ing[0],ing[2],ing[1]))
    conn.commit()
    c.close()
    return True
```

Generating ID's



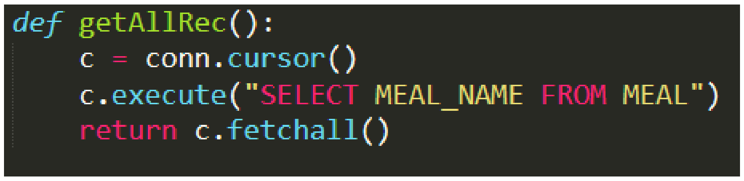
```
def idGen(existIds):
    id = random.randint(0,1000)
    if id in existIds:
        return idGen(existIds)
    else:
        return id
```

Removing a recipe



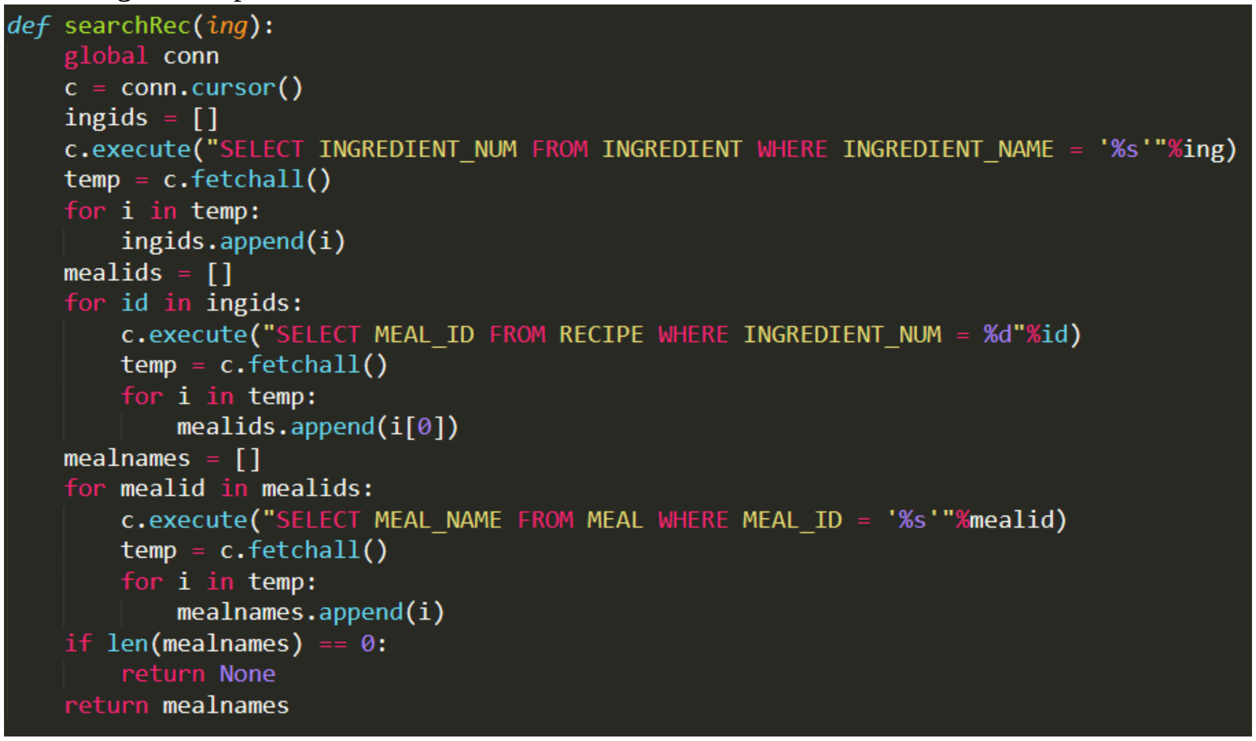
```
def remRec(name):
    global conn
    c = conn.cursor()
    c.execute("SELECT * FROM MEAL WHERE MEAL_NAME = '%s'" %name )
    results = c.fetchall()
    if len(results) == 0:
        return False
    for result in results:
        id = result[0]
        c.execute("DELETE FROM RECIPE WHERE MEAL_ID = %d" %id)
        c.execute("DELETE FROM MEAL WHERE MEAL_NAME = '%s'"%name)
    conn.commit()
    return True
```

Getting a recipe



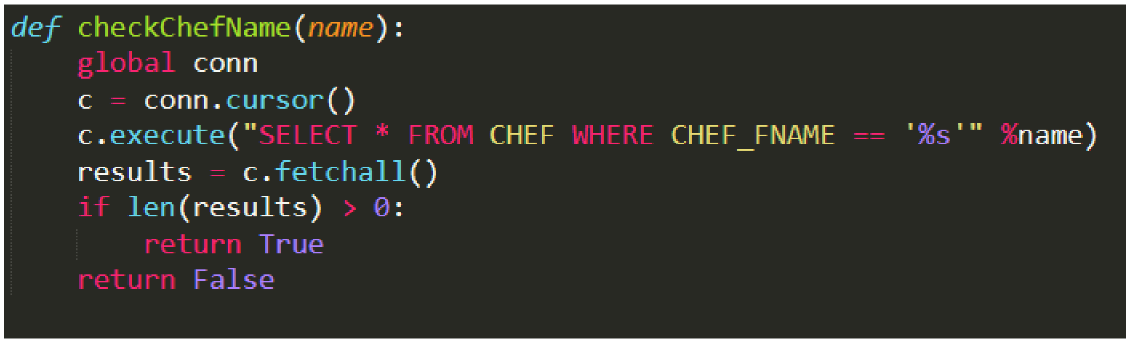
```
def getAllRec():
    c = conn.cursor()
    c.execute("SELECT MEAL_NAME FROM MEAL")
    return c.fetchall()
```

Searching for recipes



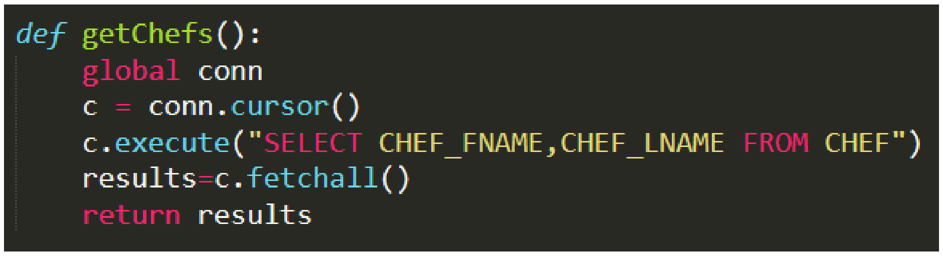
```
def searchRec(ing):
    global conn
    c = conn.cursor()
    ingids = []
    c.execute("SELECT INGREDIENT_NUM FROM INGREDIENT WHERE INGREDIENT_NAME = '%s'"%ing)
    temp = c.fetchall()
    for i in temp:
        ingids.append(i)
    mealids = []
    for id in ingids:
        c.execute("SELECT MEAL_ID FROM RECIPE WHERE INGREDIENT_NUM = %d"%id)
        temp = c.fetchall()
        for i in temp:
            mealids.append(i[0])
    mealnames = []
    for mealid in mealids:
        c.execute("SELECT MEAL_NAME FROM MEAL WHERE MEAL_ID = '%s'"%mealid)
        temp = c.fetchall()
        for i in temp:
            mealnames.append(i)
    if len(mealnames) == 0:
        return None
    return mealnames
```

Checking Chef's name for redundancies in database



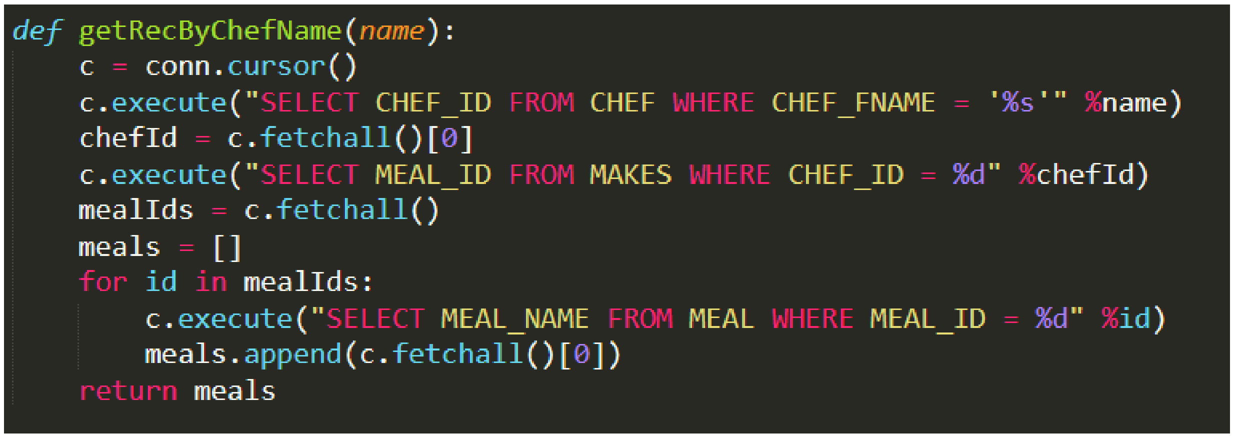
```
def checkChefName(name):
    global conn
    c = conn.cursor()
    c.execute("SELECT * FROM CHEF WHERE CHEF_FNAME == '%s'" %name)
    results = c.fetchall()
    if len(results) > 0:
        return True
    return False
```

Getting all the chefs



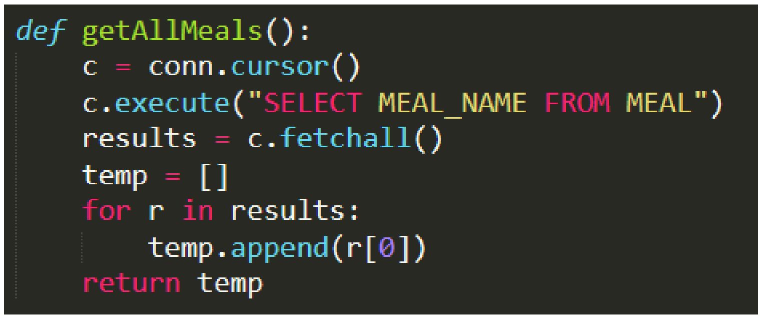
```
def getChefs():
    global conn
    c = conn.cursor()
    c.execute("SELECT CHEF_FNAME,CHEF_LNAME FROM CHEF")
    results=c.fetchall()
    return results
```

Get all recipes that a certain chef knows



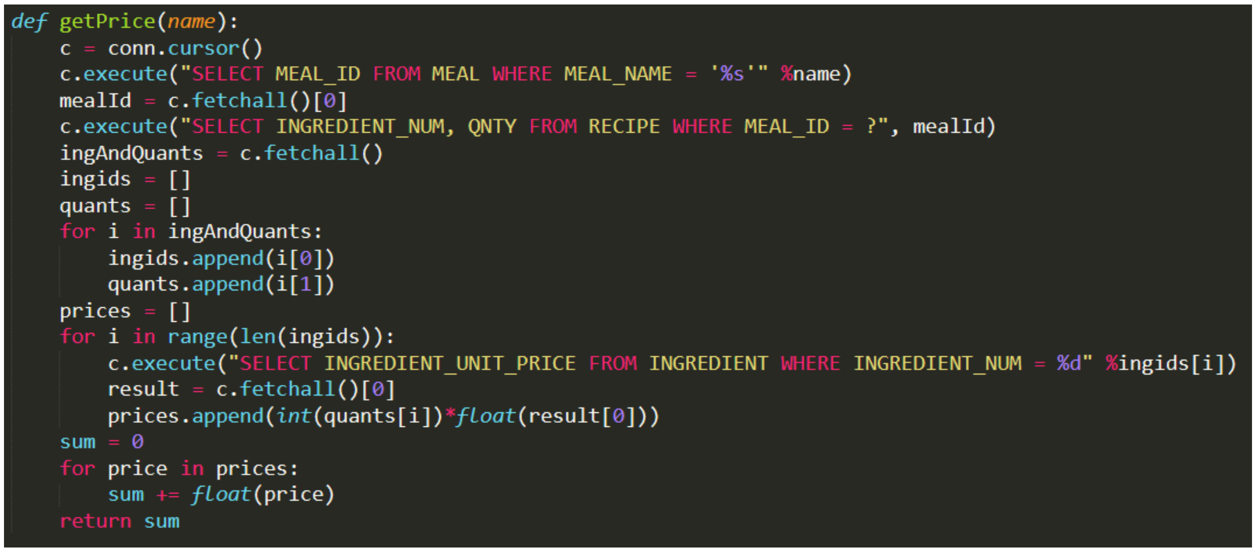
```
def getRecByChefName(name):
    c = conn.cursor()
    c.execute("SELECT CHEF_ID FROM CHEF WHERE CHEF_FNAME = '%s'" %name)
    chefId = c.fetchall()[0]
    c.execute("SELECT MEAL_ID FROM MAKES WHERE CHEF_ID = %d" %chefId)
    mealIds = c.fetchall()
    meals = []
    for id in mealIds:
        c.execute("SELECT MEAL_NAME FROM MEAL WHERE MEAL_ID = %d" %id)
        meals.append(c.fetchall()[0])
    return meals
```

Get all meals in the meal table



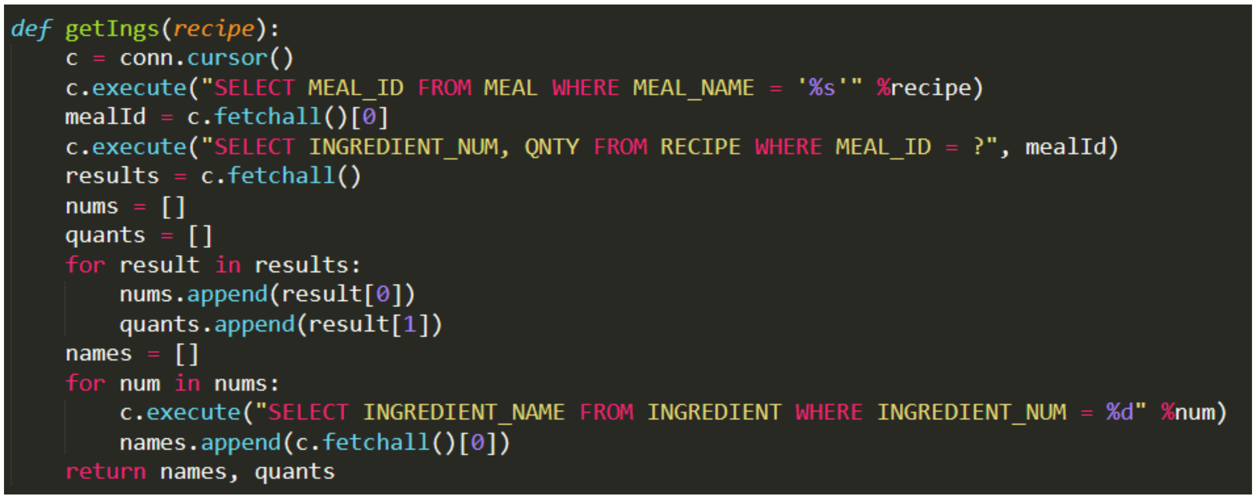
```
def getAllMeals():
    c = conn.cursor()
    c.execute("SELECT MEAL_NAME FROM MEAL")
    results = c.fetchall()
    temp = []
    for r in results:
        temp.append(r[0])
    return temp
```

Get price of a certain meal



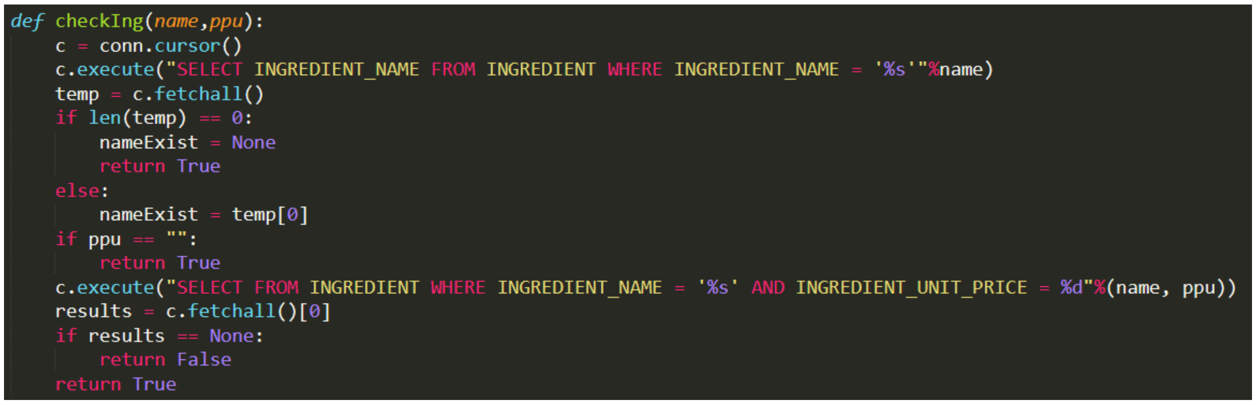
```
def getPrice(name):
    c = conn.cursor()
    c.execute("SELECT MEAL_ID FROM MEAL WHERE MEAL_NAME = '%s'" %name)
    mealId = c.fetchall()[0]
    c.execute("SELECT INGREDIENT_NUM, QNTY FROM RECIPE WHERE MEAL_ID = ?", mealId)
    ingAndQuants = c.fetchall()
    ingids = []
    quants = []
    for i in ingAndQuants:
        ingids.append(i[0])
        quants.append(i[1])
    prices = []
    for i in range(len(ingids)):
        c.execute("SELECT INGREDIENT_UNIT_PRICE FROM INGREDIENT WHERE INGREDIENT_NUM = %d" %ingids[i])
        result = c.fetchall()[0]
        prices.append(int(quants[i])*float(result[0]))
    sum = 0
    for price in prices:
        sum += float(price)
    return sum
```

Get ingredients of a certain meal



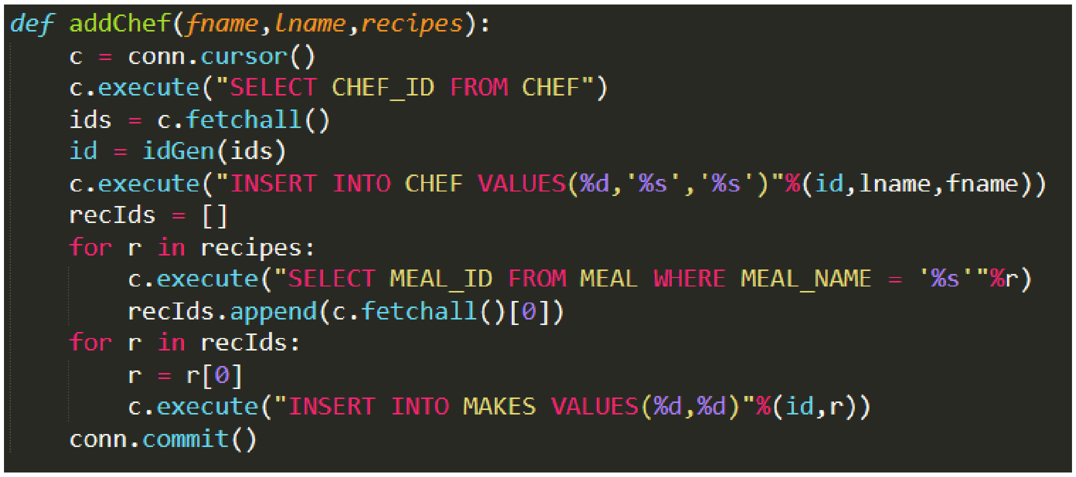
```python
def getIngs(recipe):
    c = conn.cursor()
    c.execute("SELECT MEAL_ID FROM MEAL WHERE MEAL_NAME = '%s'" %recipe)
    mealId = c.fetchall()[0]
    c.execute("SELECT INGREDIENT_NUM, QNTY FROM RECIPE WHERE MEAL_ID = ?", mealId)
    results = c.fetchall()
    nums = []
    quants = []
    for result in results:
        nums.append(result[0])
        quants.append(result[1])
    names = []
    for num in nums:
        c.execute("SELECT INGREDIENT_NAME FROM INGREDIENT WHERE INGREDIENT_NUM = %d" %num)
        names.append(c.fetchall()[0])
    return names, quants
```

Check if ingredient already exists



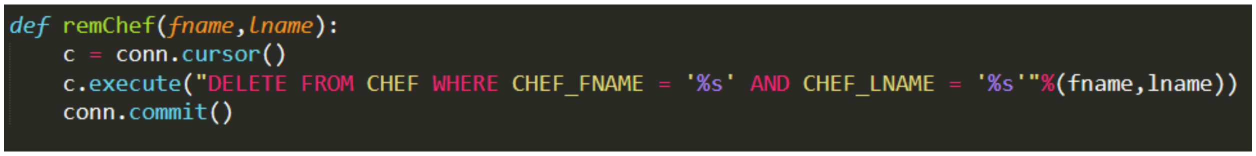
```python
def checkIng(name,ppu):
    c = conn.cursor()
    c.execute("SELECT INGREDIENT_NAME FROM INGREDIENT WHERE INGREDIENT_NAME = '%s'"%name)
    temp = c.fetchall()
    if len(temp) == 0:
        nameExist = None
        return True
    else:
        nameExist = temp[0]
    if ppu == "":
        return True
    c.execute("SELECT FROM INGREDIENT WHERE INGREDIENT_NAME = '%s' AND INGREDIENT_UNIT_PRICE = %d"%(name, ppu))
    results = c.fetchall()[0]
    if results == None:
        return False
    return True
```

Adding a chef



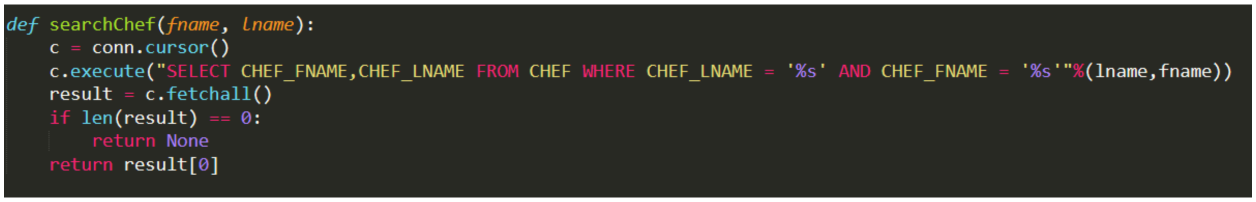
```python
def addChef(fname,lname,recipes):
    c = conn.cursor()
    c.execute("SELECT CHEF_ID FROM CHEF")
    ids = c.fetchall()
    id = idGen(ids)
    c.execute("INSERT INTO CHEF VALUES(%d,'%s','%s')"%(id,lname,fname))
    recIds = []
    for r in recipes:
        c.execute("SELECT MEAL_ID FROM MEAL WHERE MEAL_NAME = '%s'"%r)
        recIds.append(c.fetchall()[0])
    for r in recIds:
        r = r[0]
        c.execute("INSERT INTO MAKES VALUES(%d,%d)"%(id,r))
    conn.commit()
```

Remove a chef



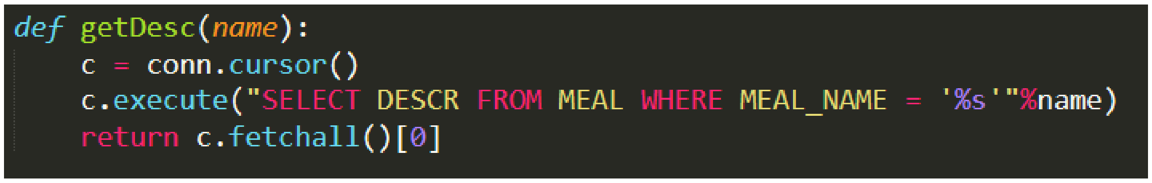
```
def remChef(fname,lname):
    c = conn.cursor()
    c.execute("DELETE FROM CHEF WHERE CHEF_FNAME = '%s' AND CHEF_LNAME = '%s'"%(fname,lname))
    conn.commit()
```

Search for chef



```
def searchChef(fname, lname):
    c = conn.cursor()
    c.execute("SELECT CHEF_FNAME,CHEF_LNAME FROM CHEF WHERE CHEF_LNAME = '%s' AND CHEF_FNAME = '%s'"%(lname,fname))
    result = c.fetchall()
    if len(result) == 0:
        return None
    return result[0]
```

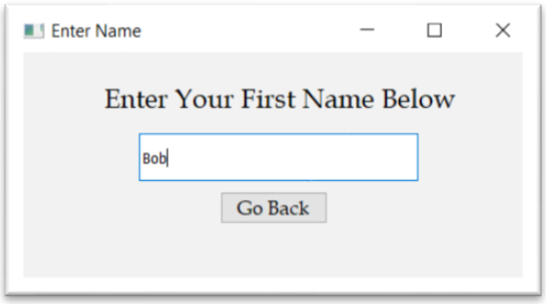
Get Recipe Description

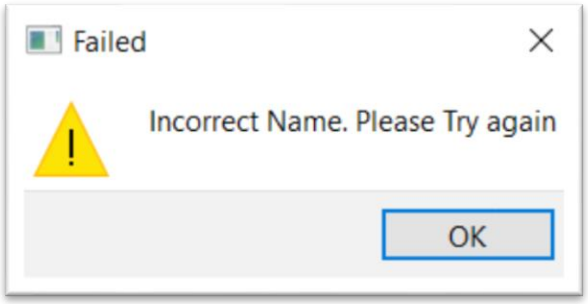


```
def getDesc(name):
    c = conn.cursor()
    c.execute("SELECT DESCR FROM MEAL WHERE MEAL_NAME = '%s'"%name)
    return c.fetchall()[0]
```

Sample Output

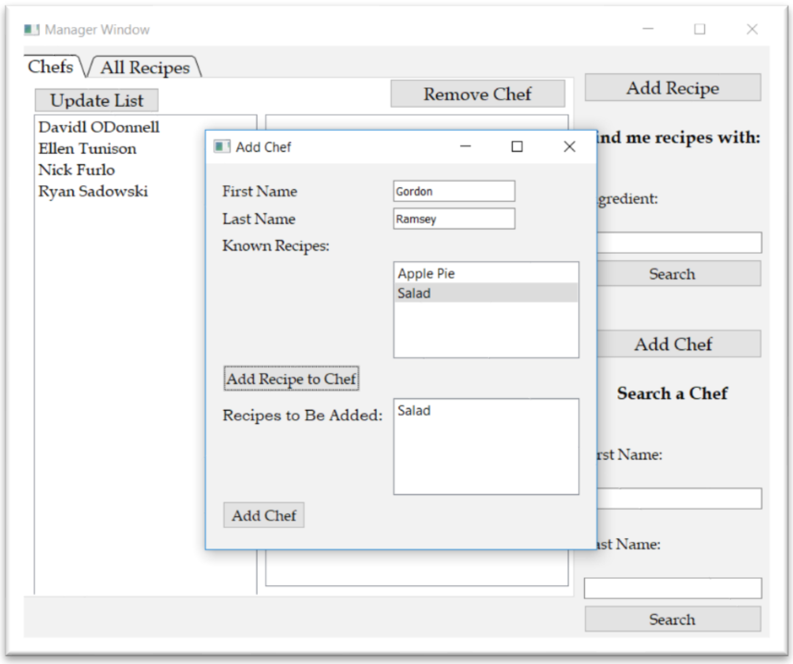
Logging in as a chef that hasn't been created yet.



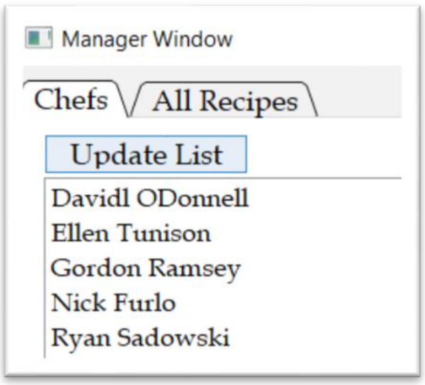


Incorrect Name. Please Try again

Adding a chef

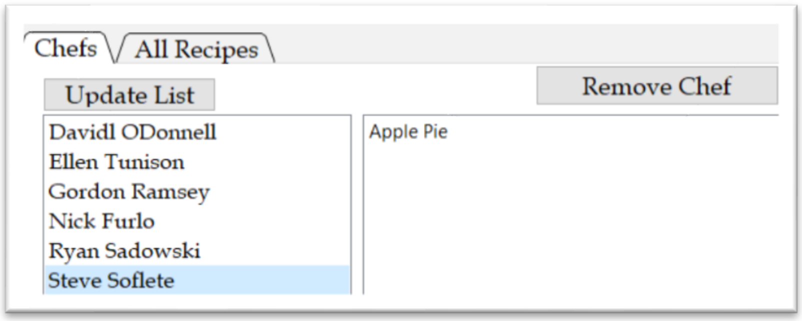


First Name: Gordon
Last Name: Ramsey
Known Recipes: Apple Pie, Salad
Recipes to Be Added: Salad
Davidl ODonnell
Ellen Tunison
Nick Furlo
Ryan Sadowski



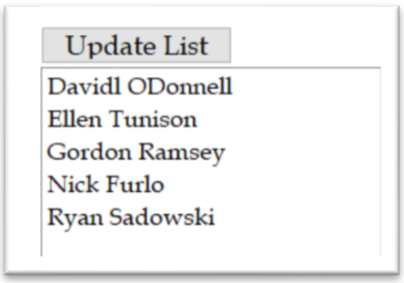
Davidl ODonnell
Ellen Tunison
Gordon Ramsey
Nick Furlo
Ryan Sadowski

Removing a chef



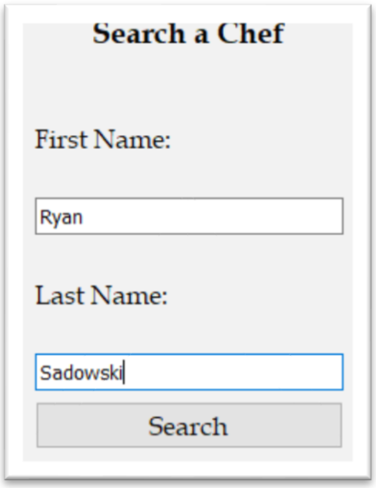
Davidl ODonnell
Ellen Tunison
Gordon Ramsey
Nick Furlo
Ryan Sadowski
Steve Soflete

Apple Pie



Davidl ODonnell
Ellen Tunison
Gordon Ramsey
Nick Furlo
Ryan Sadowski

Search for a chef

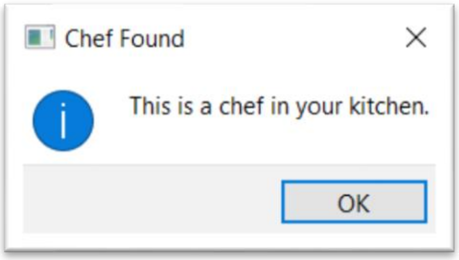

Search a Chef

First Name:

Ryan

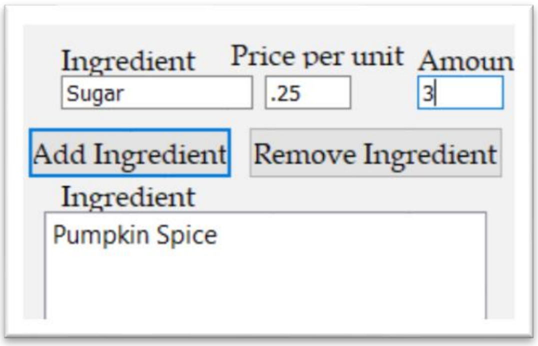
Last Name:

Sadowski


Chef Found

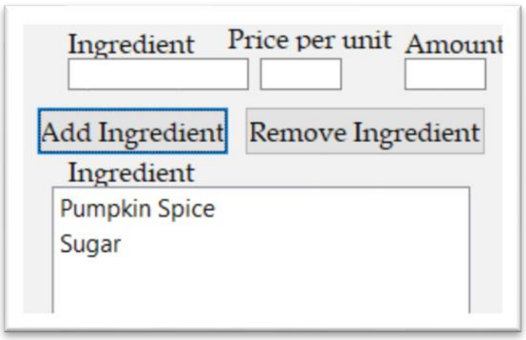
This is a chef in your kitchen.

Add Ingredient to recipe


Ingredient	Price per unit	Amoun
Sugar	.25	3

Ingredient

Pumpkin Spice

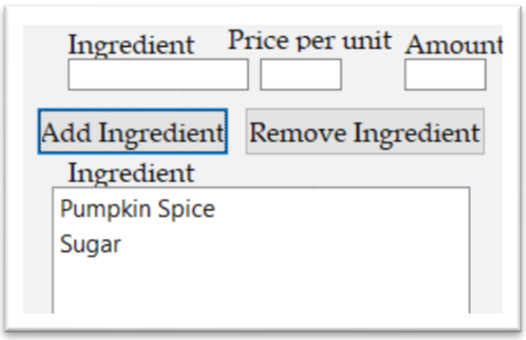


Ingredient

Pumpkin Spice

Sugar

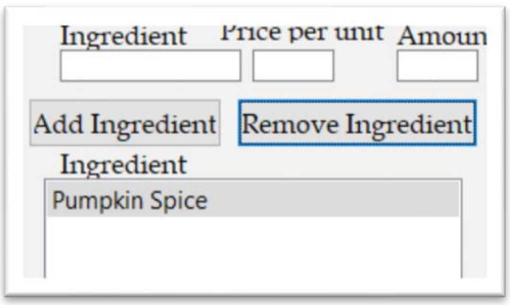
Remove Ingredient



Ingredient

Pumpkin Spice

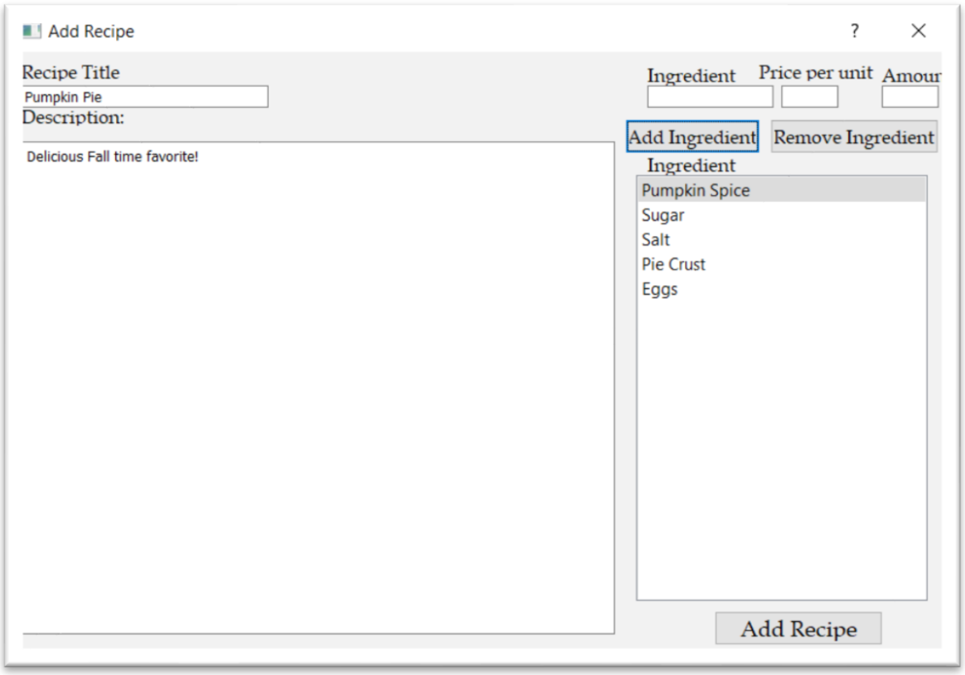
Sugar



Ingredient

Pumpkin Spice

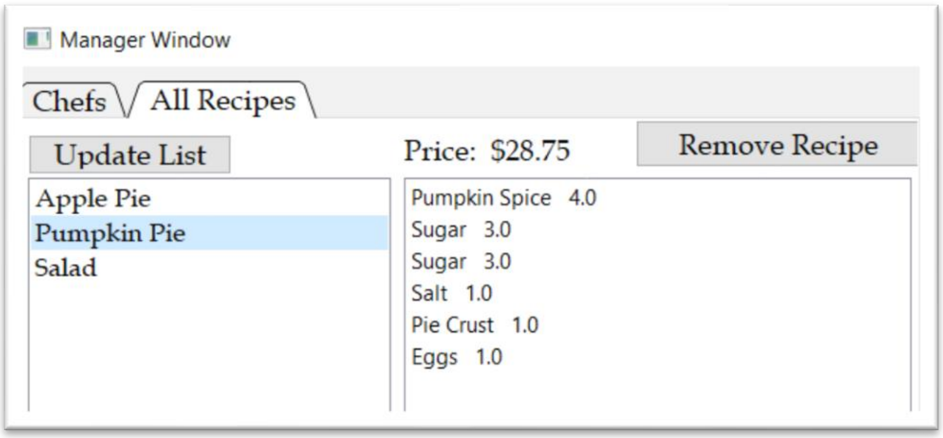
Adding a recipe



Pumpkin Pie

Delicious Fall time favorite!

Pumpkin Spice
Sugar
Salt
Pie Crust
Eggs

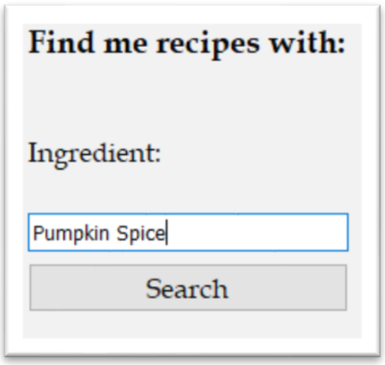


Price: $28.75

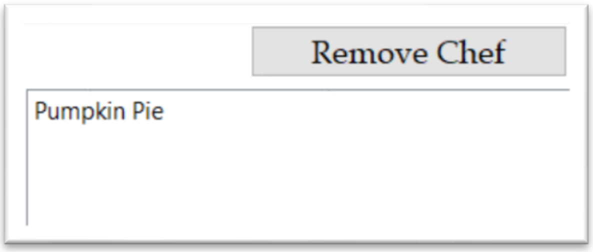
Apple Pie
Pumpkin Pie
Salad

```
Pumpkin Spice   4.0
Sugar   3.0
Sugar   3.0
Salt   1.0
Pie Crust   1.0
Eggs   1.0
```

Search for recipe containing an ingredient

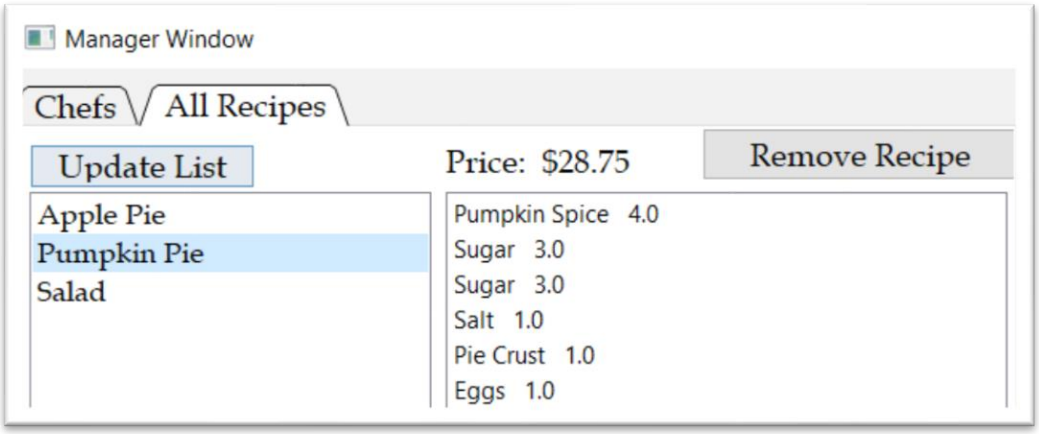


Pumpkin Spice



Pumpkin Pie

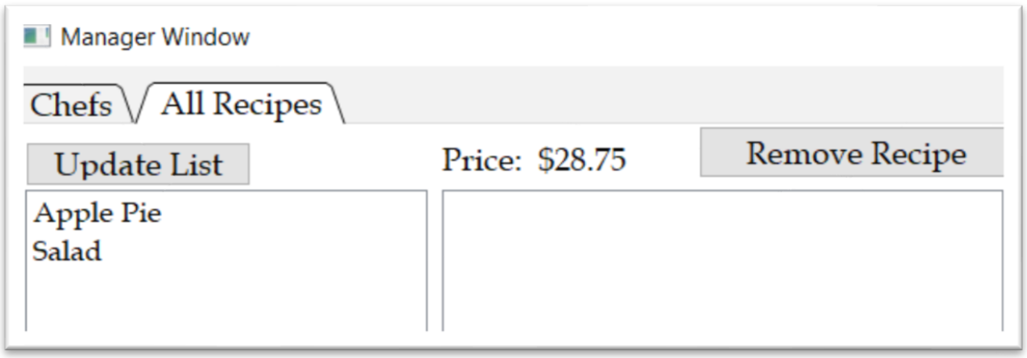
Remove recipe



Price: $28.75

Apple Pie
Pumpkin Pie
Salad

```
Pumpkin Spice   4.0
Sugar   3.0
Sugar   3.0
Salt   1.0
Pie Crust   1.0
Eggs   1.0
```



Price: $28.75

Apple Pie
Salad