ReciPlan

Project Design

Version 4

CMSC 495

For: Dr. Hung Dao

Due: 6 March 2022

Authors:

Team 6 – Josh Coe, Danita Hodges, Scott McClure

Revision History

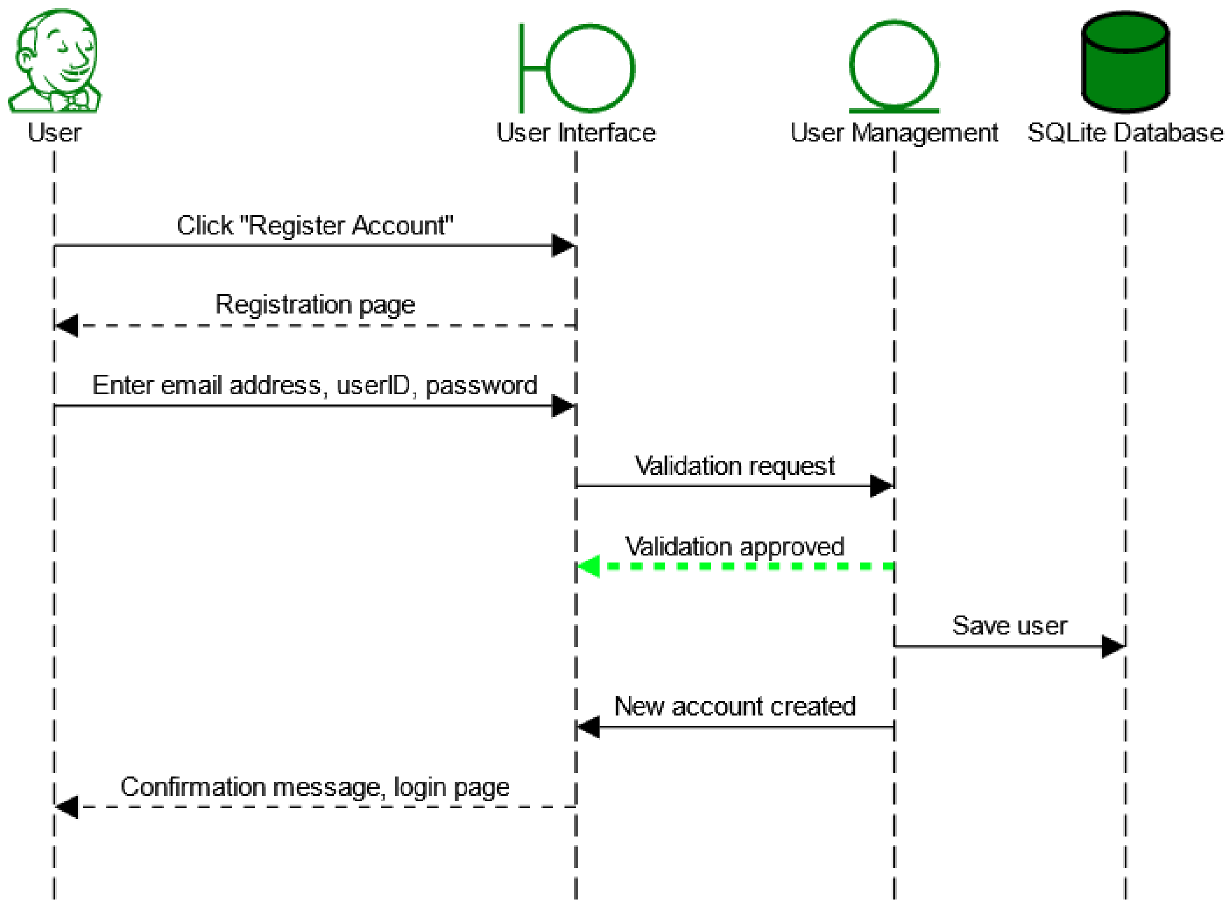
Revision	Date	Author	Changes
1	1 FEB 22	Josh	Initial creation of document
2	5 FEB 22	Josh	Added pseudocode for 5 classes
3	6 FEB 22	Danita	Added event-trace diagrams for all scenarios and potential errors, formatting, header, footer, reviewed pseudocode
4	6 FEB 22	Scott	Added pseudocode for 4 classes & possible enhancements & mitigations

Event-Trace Diagrams:

Scenario 1: Create new user

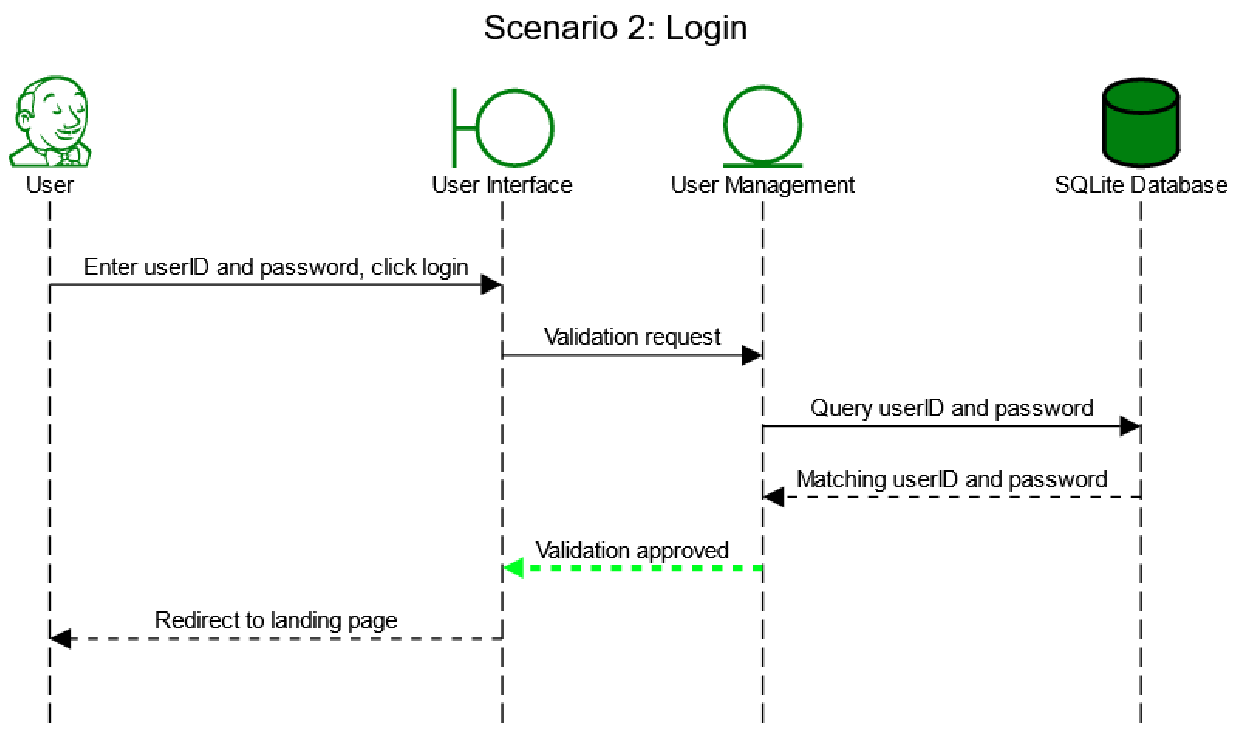
Description:	New users navigate to the website and create a new user profile consisting of a unique email address, unique user ID and a password.
Precondition:	User has an internet connection and is on the home page
Postcondition:	User is redirected to the login page

Scenario 1: Create New User



Scenario 2: **Login**

Description:	Registered users navigate to website and gain access by entering a valid user ID and password combination
Precondition:	User already created and stored in database
Postcondition:	User redirected to landing page



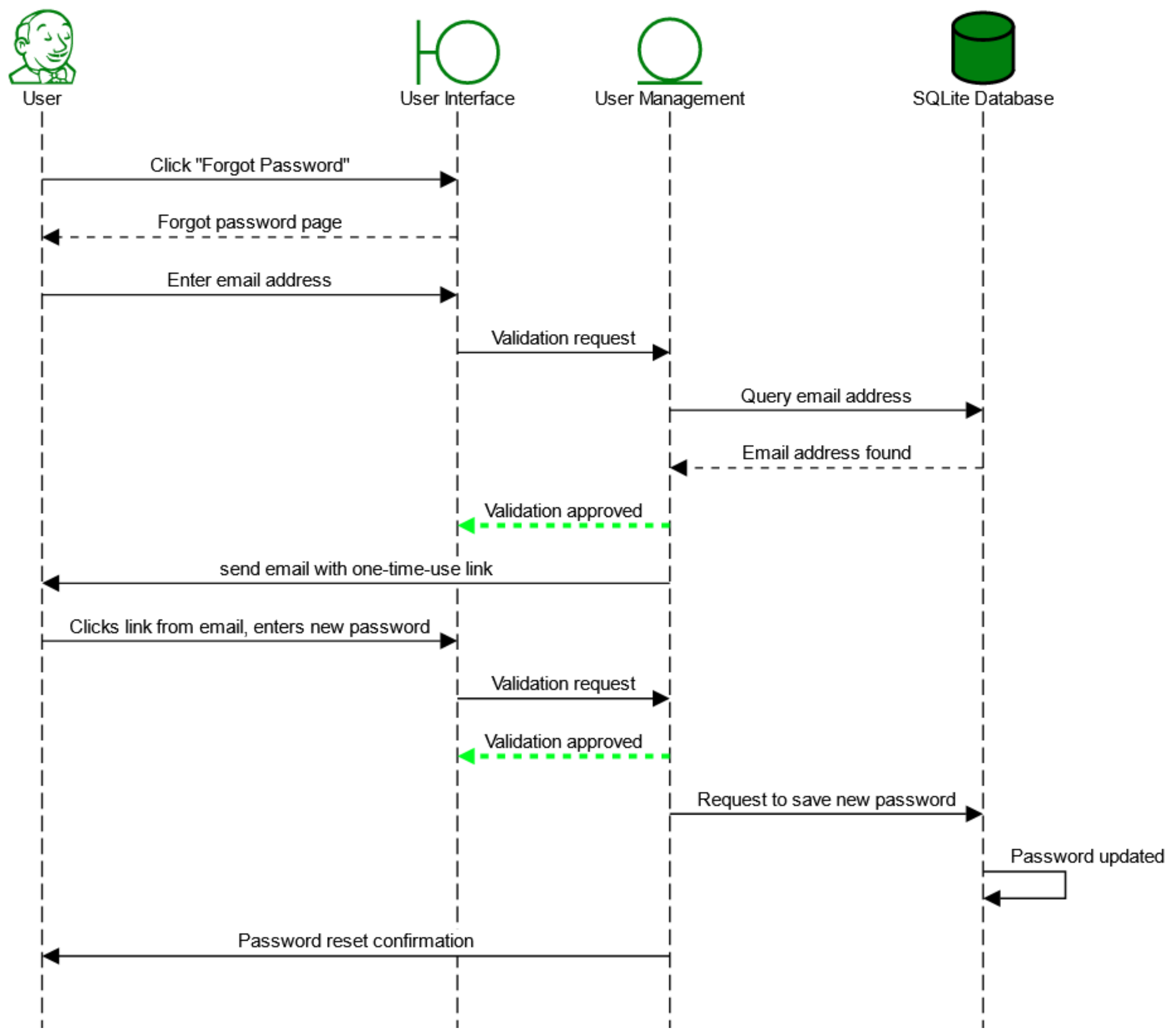
Scenario 3: **Reset password**

Description: Registered users reset their password via email confirmation

Precondition: User already created and stored in database

Postcondition: User changes password successfully

Scenario 3: Reset Password



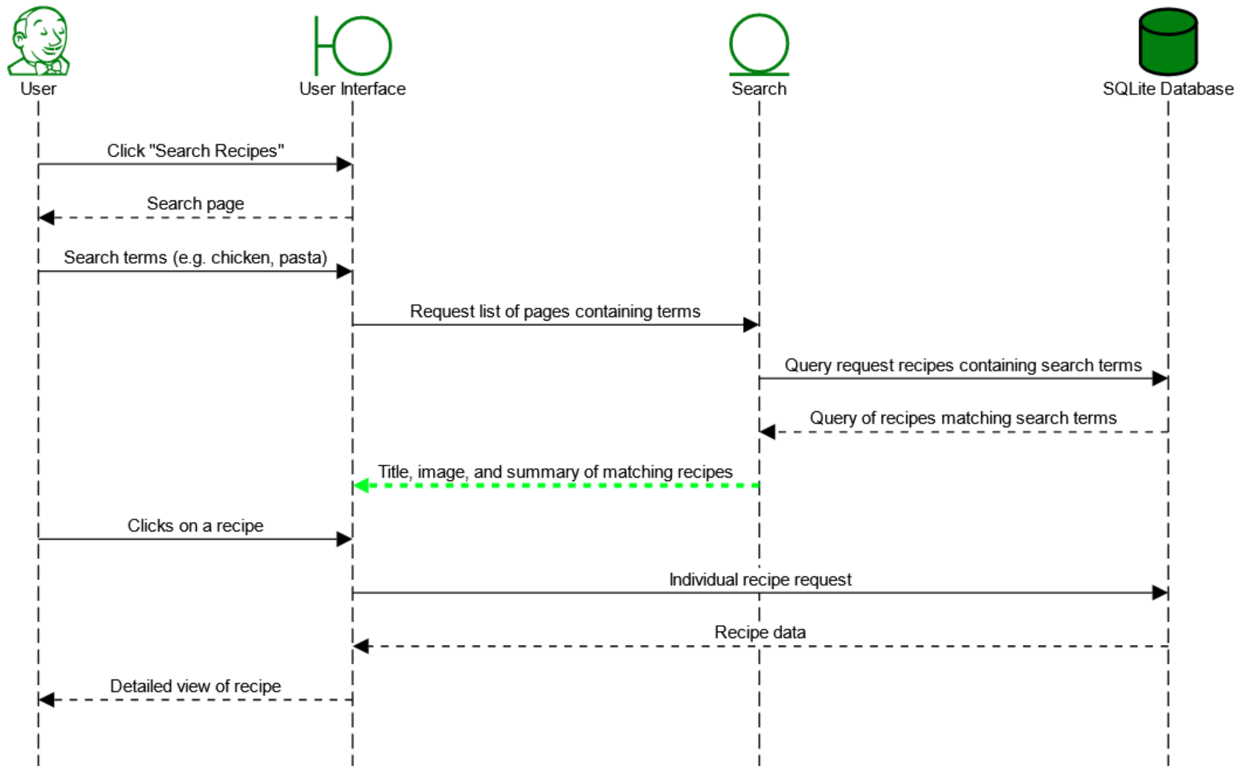
Scenario 4: **Search and view recipe**

Description: User searches the recipe library by recipe name using a search box.

Precondition: Recipes stored in the database

Postcondition: User viewing recipe details

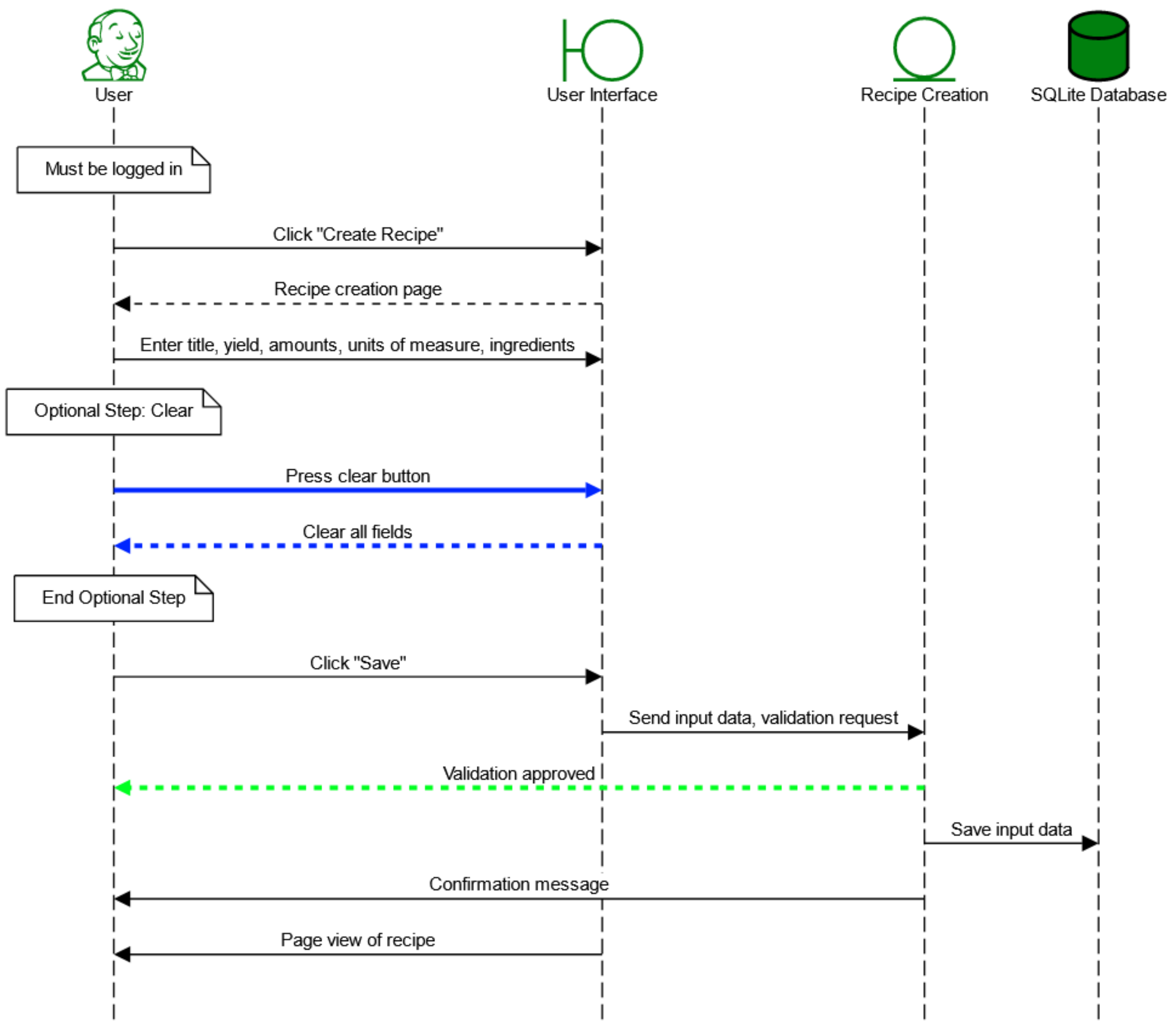
Scenario 4: Search and View Recipe


User
User Interface
Search
SQLite Database
Click "Search Recipes"
Search page
Search terms (e.g. chicken, pasta)
Request list of pages containing terms
Query request recipes containing search terms
Query of recipes matching search terms
Title, image, and summary of matching recipes
Clicks on a recipe
Individual recipe request
Recipe data
Detailed view of recipe

Scenario 5: Create recipe

Description:	Registered users create their own recipe and add it to the library. While entering fields, they can clear them at any time and start over
Precondition:	User must be logged in
Postcondition:	User views recipe detail page

Scenario 5: Create Recipe



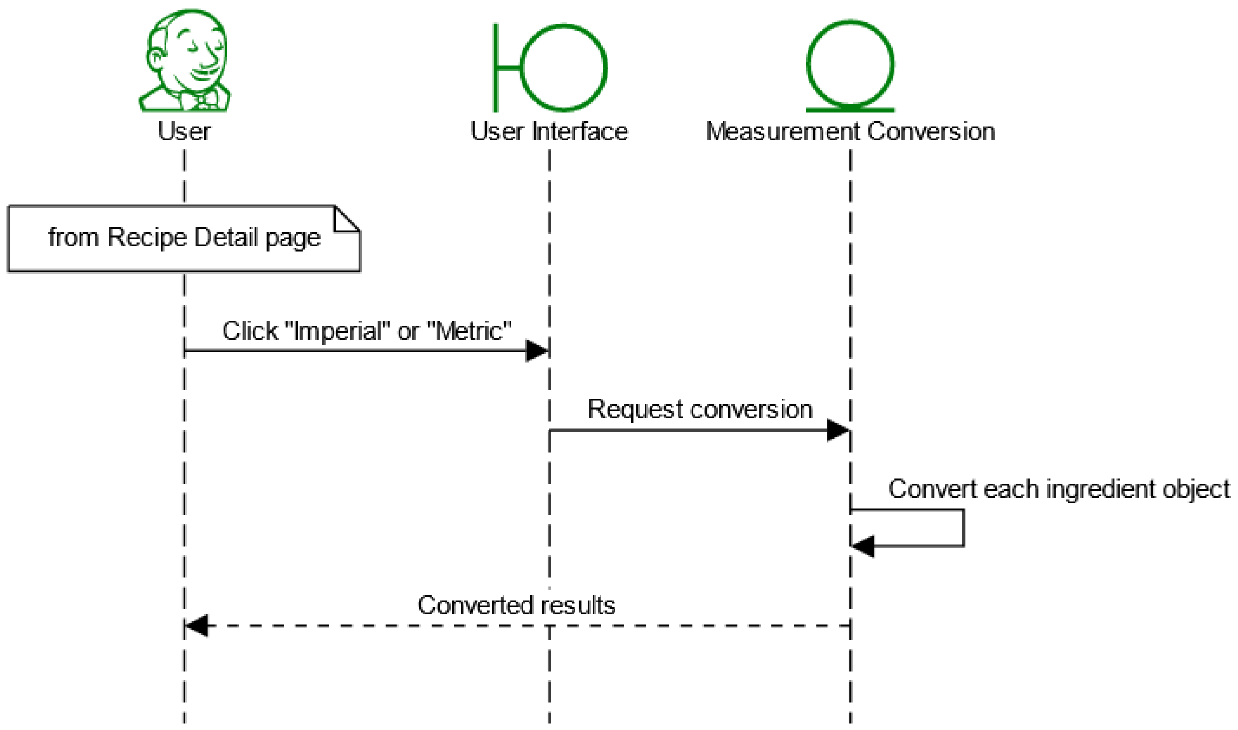
Scenario 6: **Convert measurements**

Description: User clicks a button that will convert ingredient amounts between imperial and metric in the recipe detail view.

Precondition: Recipes stored in the database, user on recipe detail page

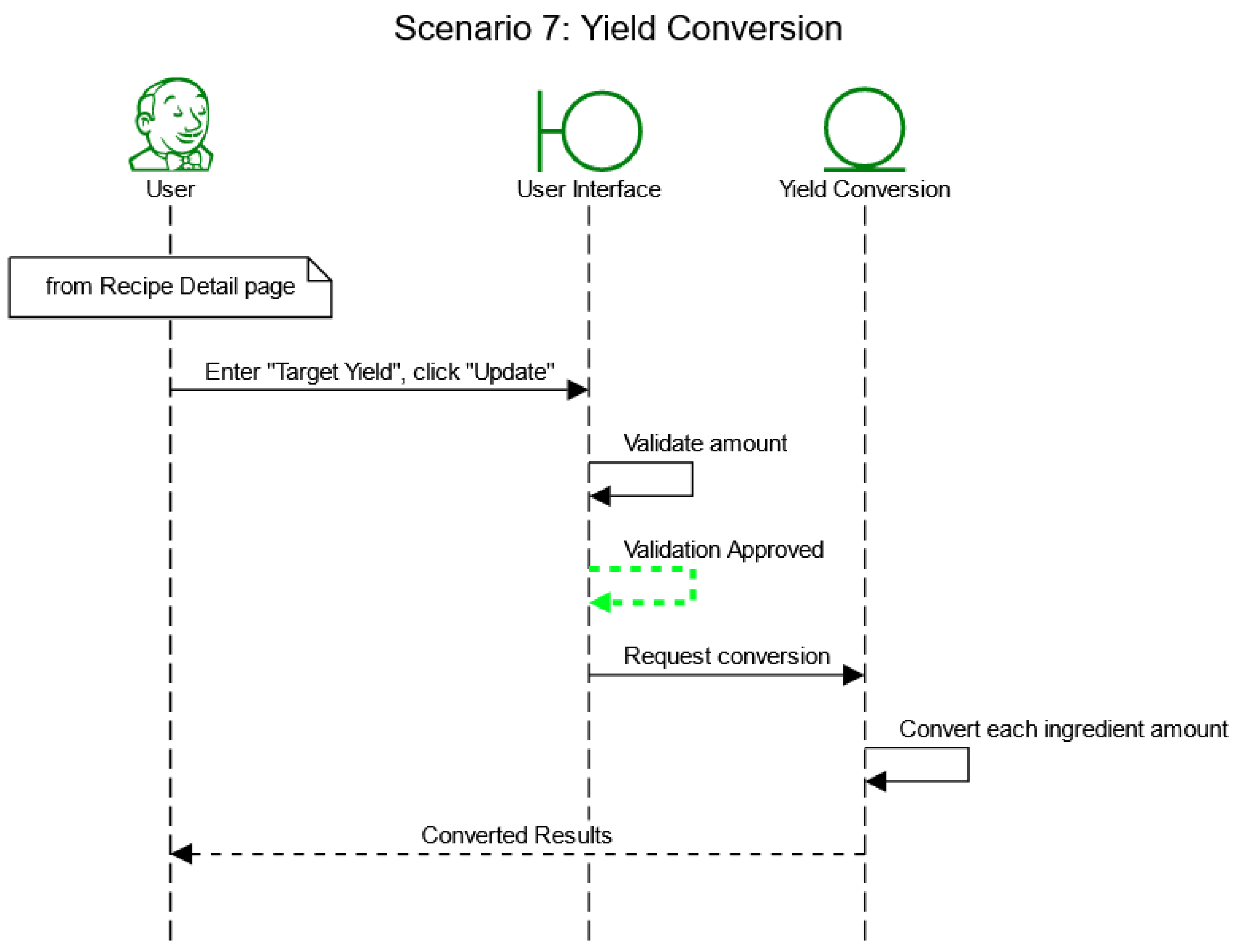
Postcondition: Recipe converted to desired conversion

Scenario 6: Convert Measurements



Scenario 7: Yield conversion

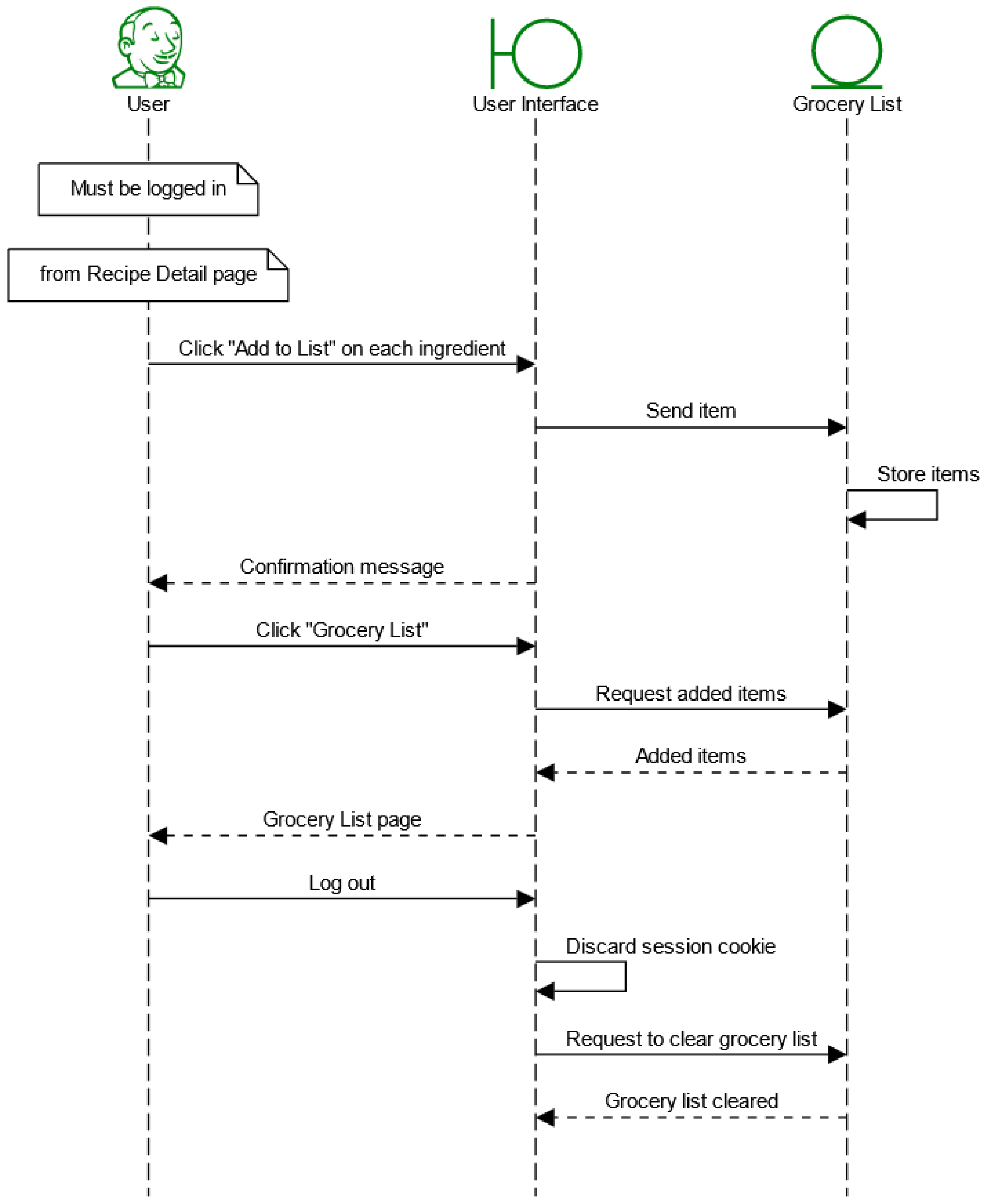
Description:	User changes the target yield on any recipe to accommodate for the amount of portions required in the recipe detail view.
Precondition:	Recipes stored in the database, user on recipe detail page
Postcondition:	Recipe converted to desired conversion



Scenario 8: Add to and view grocery list

Description:	Registered users add ingredients to a shopping list directly from the recipe detail view.
Precondition:	User must be logged in, recipe must be stored in database
Postcondition:	Grocery list cleared when user logs out

Scenario 8: Add to and View Grocery List

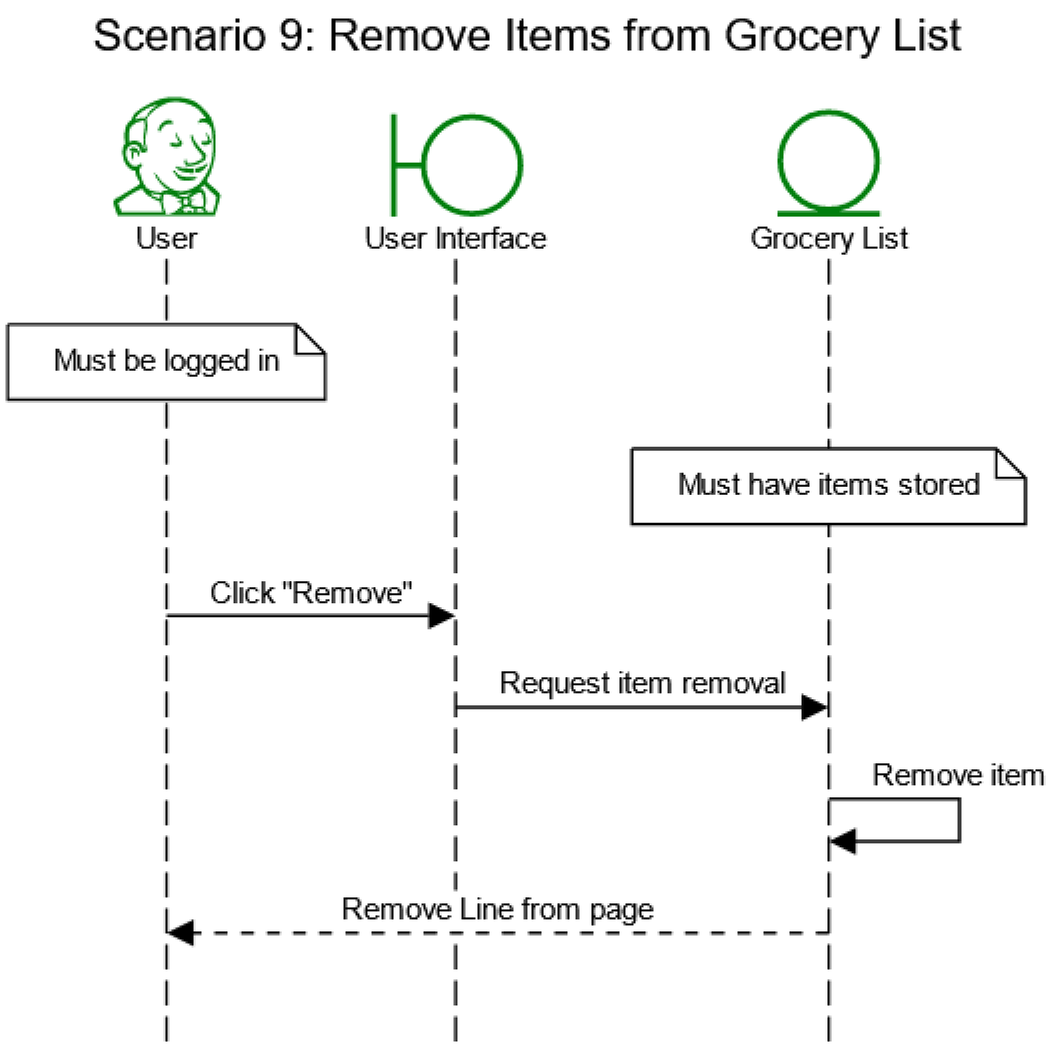


Scenario 9: **Remove items from grocery list**

Description: Registered users remove items from their shopping list.

Precondition: Grocery list has items, user logged in

Postcondition: Grocery list cleared if no items are left



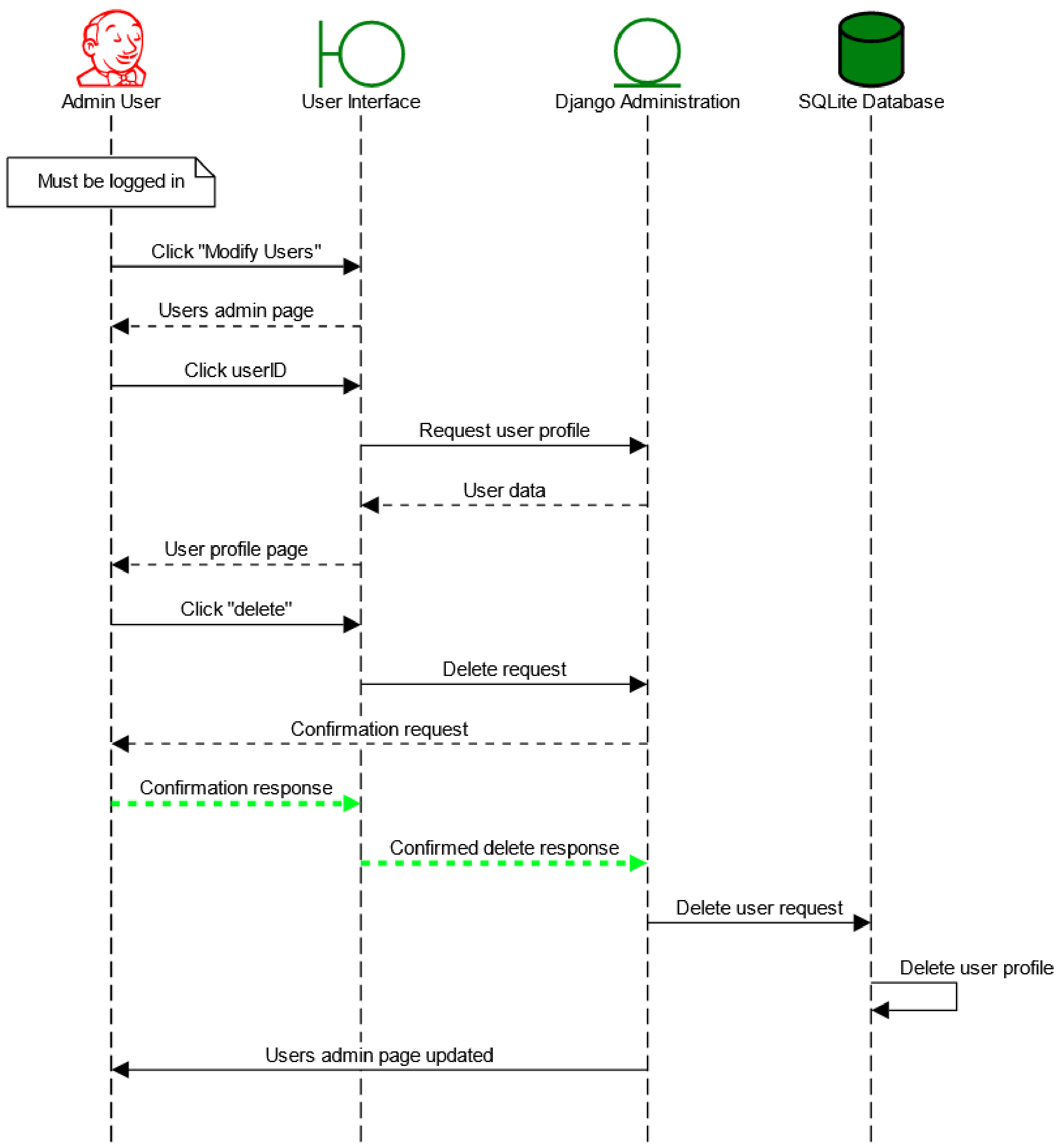
Scenario 10: Delete user profiles

Description: Administrators delete user profiles as needed.

Precondition: User to delete exists in database, admin must be logged in

Postcondition: User deleted

Scenario 10: Delete User Profiles



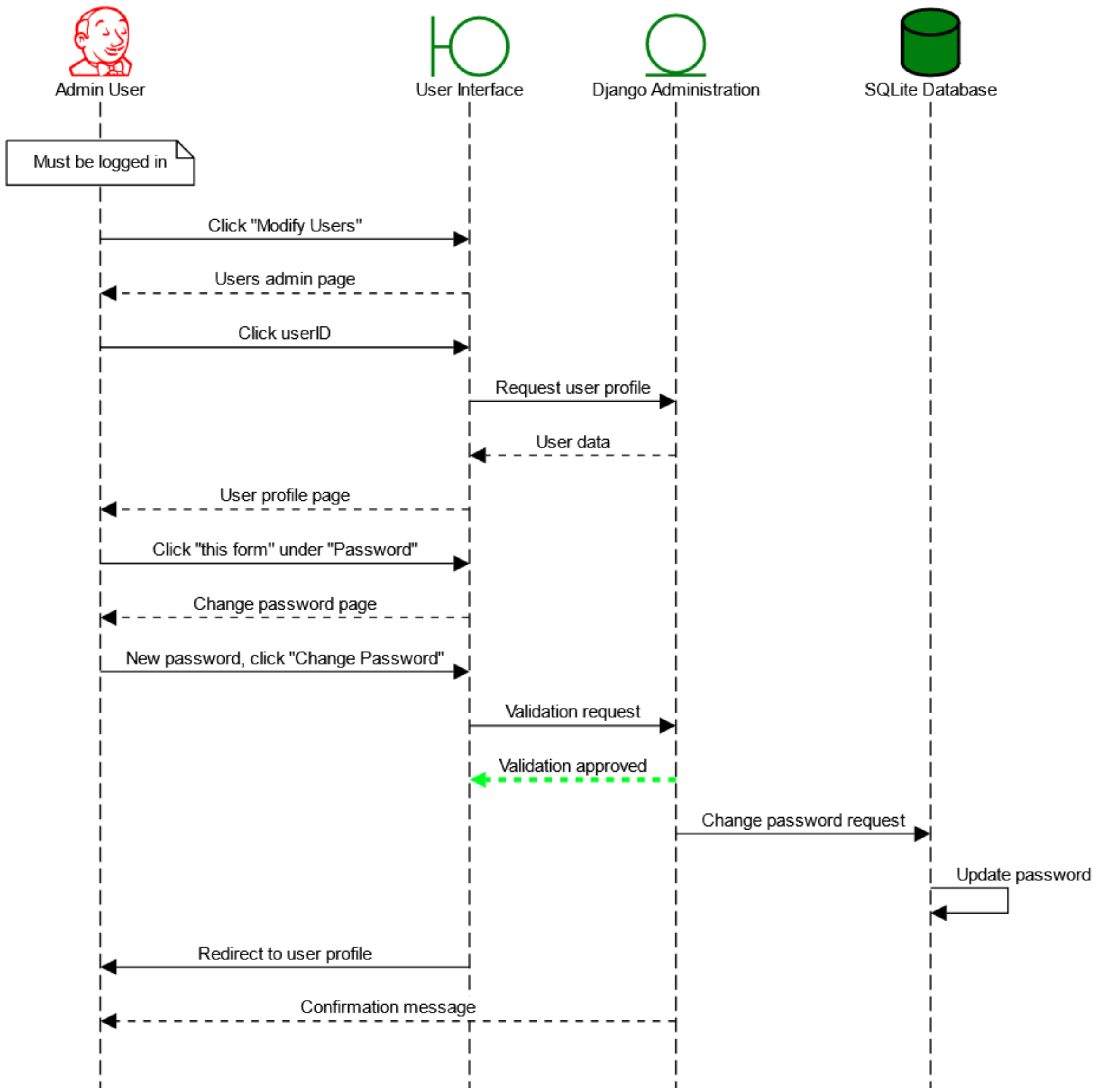
Scenario 11: Admin resets password

Description: Administrators reset user passwords as needed.

Precondition: Admin must be logged in, user to modify must exist in database

Postcondition: User password change successfully

Scenario 11: Reset Password



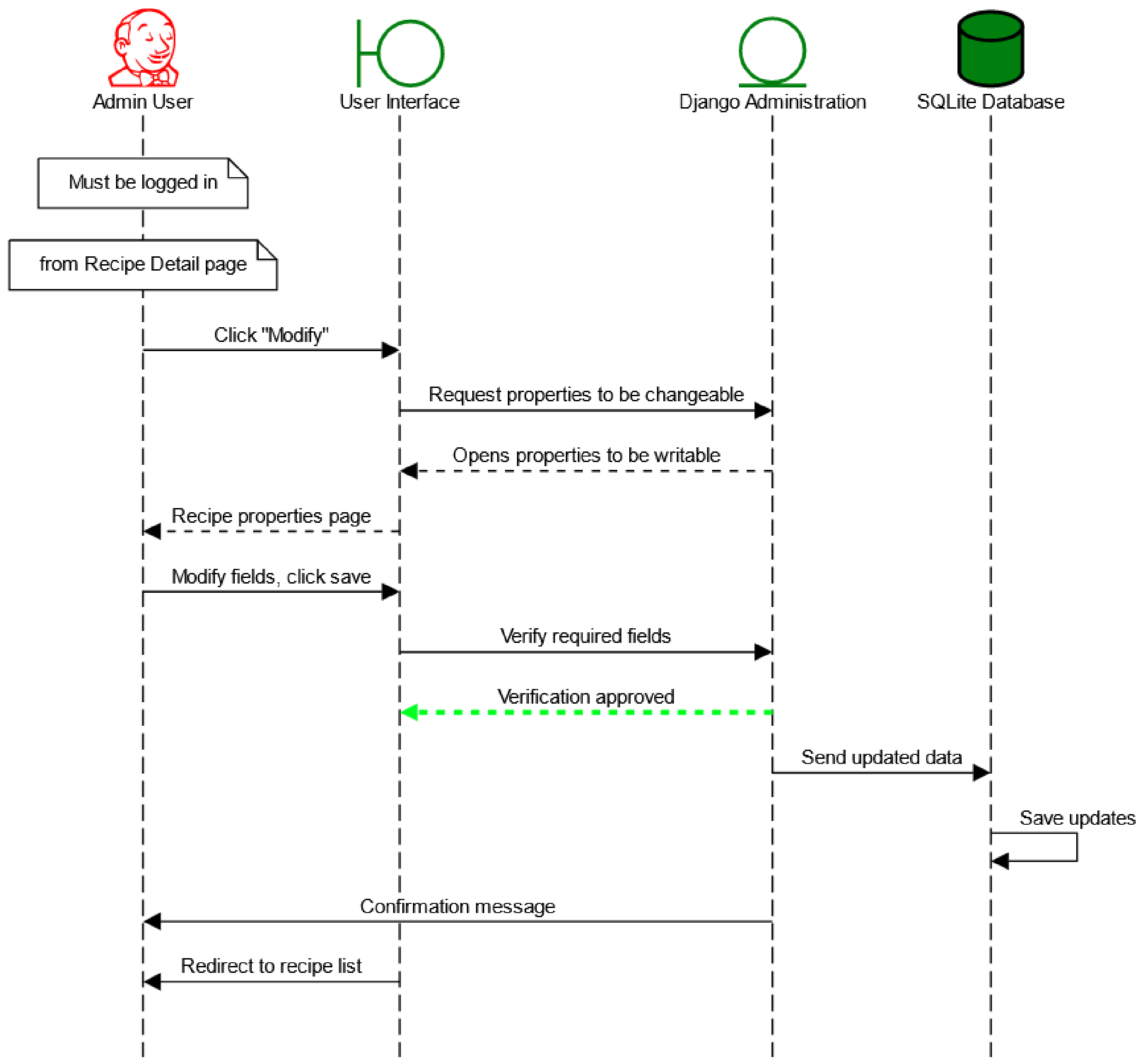
Scenario 12: Modify recipe

Description: Administrators modify any recipe in the library.

Precondition: Recipe stored in database, admin logged in

Postcondition: Recipe modified successfully

Scenario 12: Modify Recipe

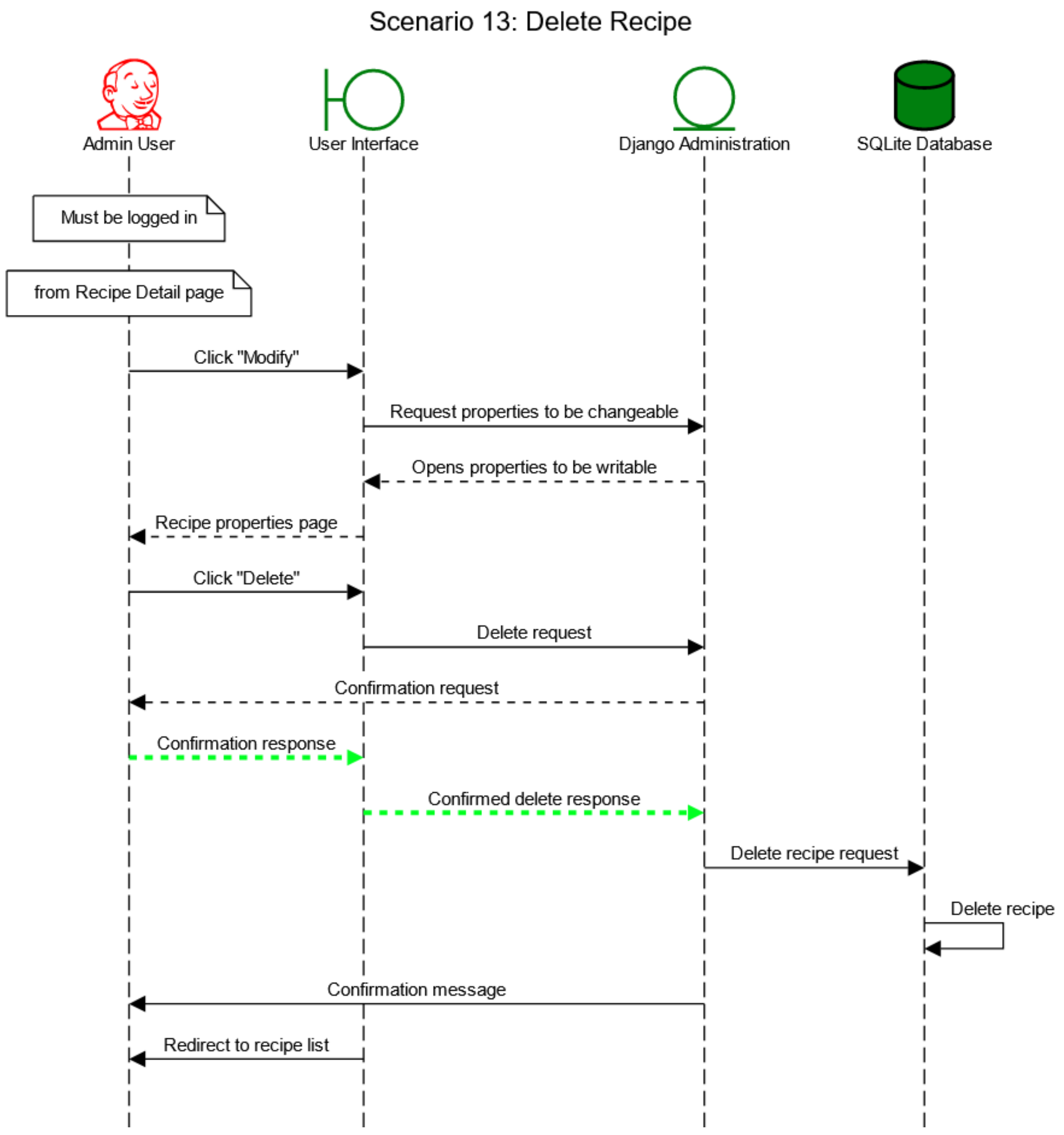


Scenario 13: Delete recipe

Description: Administrators delete any recipe in the library.

Precondition: Recipe stored in database, admin logged in

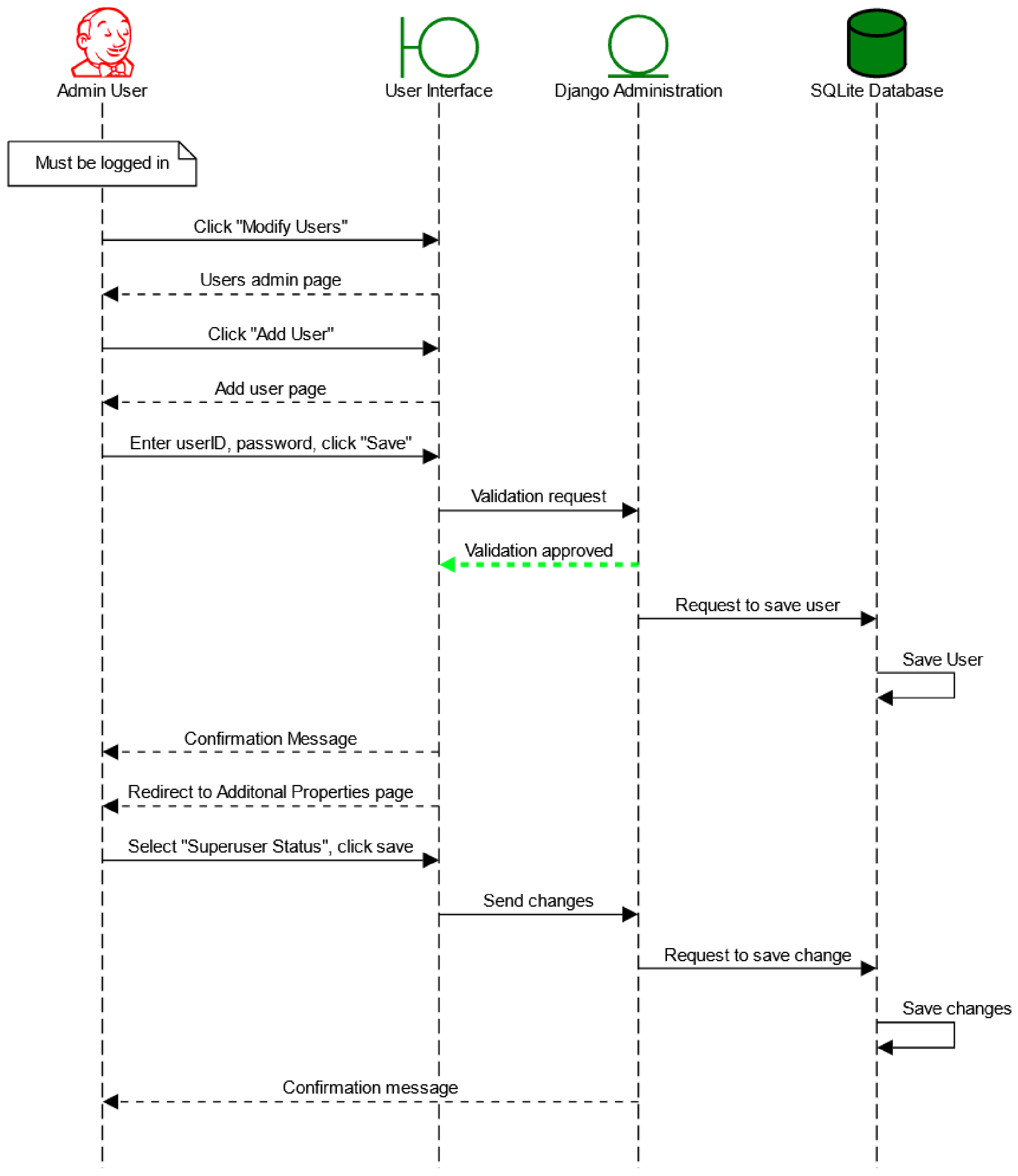
Postcondition: Recipe deleted successfully



Scenario 14: Create new admin users

Description:	Administrators create new admin accounts
Precondition:	New admin user does not exist in database or user exists with no admin privileges
Postcondition:	User modified to admin user successfully

Scenario 14: Create New Admin Users



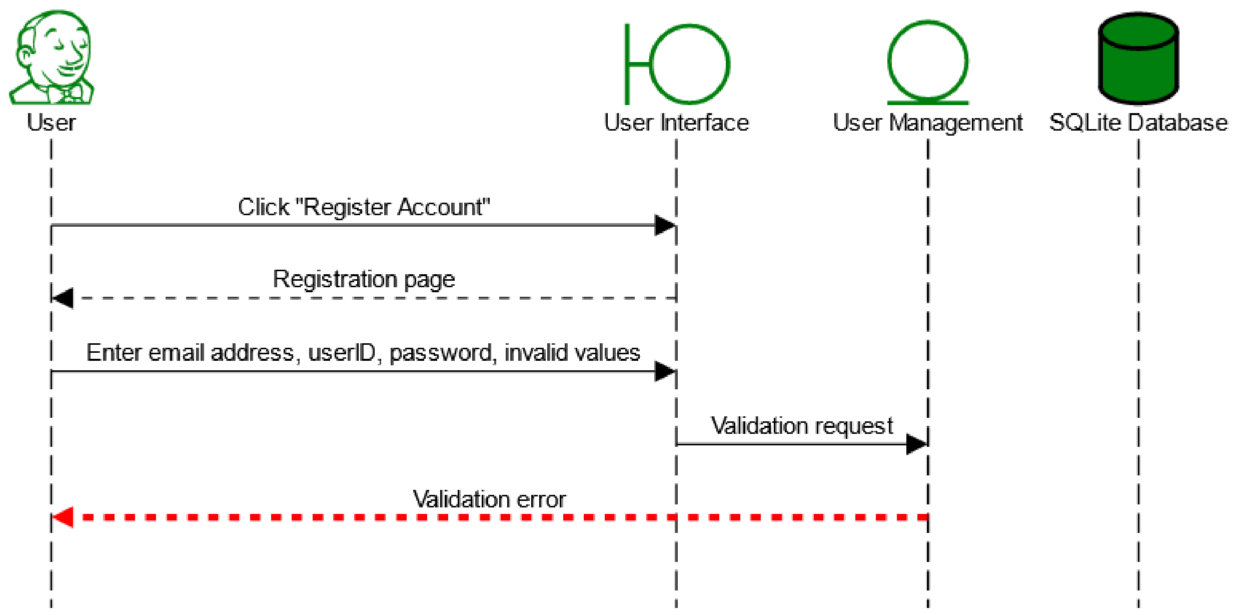
Error 1: **Create new user**

Description: User enters unvalid values when registering a new account

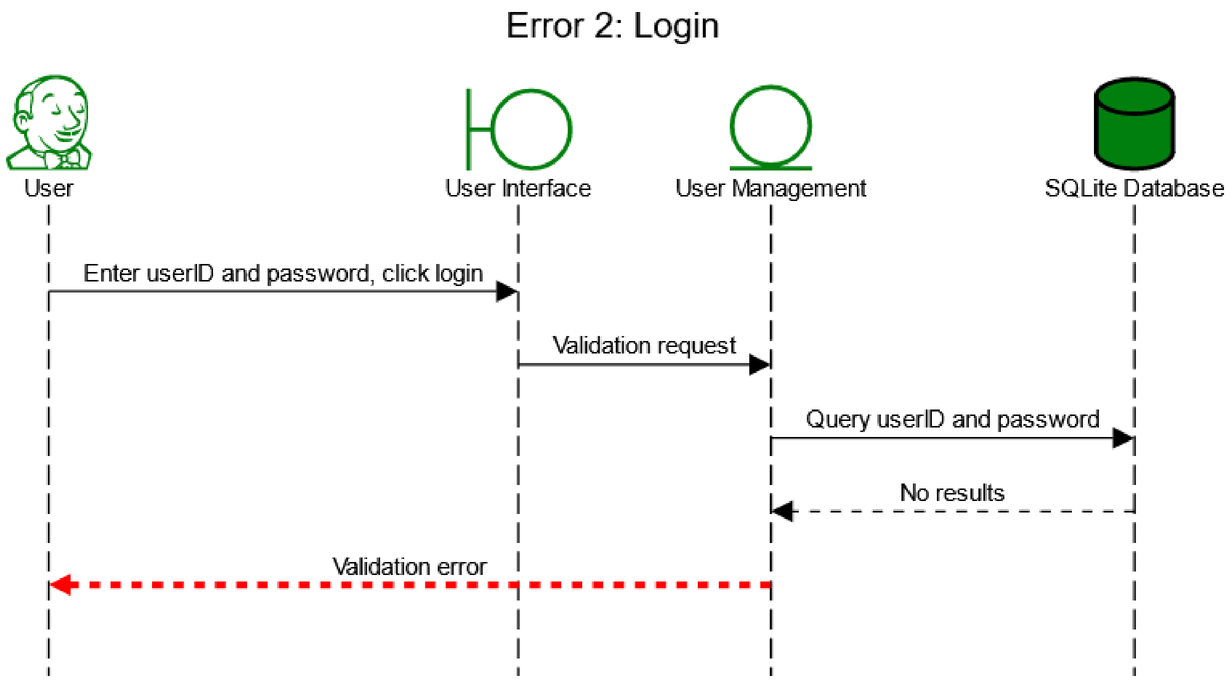
Precondition: User has an internet connection and is on the home page

Postcondition: User receives an error message until valid fields are entered

Error 1: Create New User



Error 2:	**Login**
Description:	User enters invalid username or password
Precondition:	User not created or fields do not match what is stored in database
Postcondition:	User receives an error message until valid fields are entered



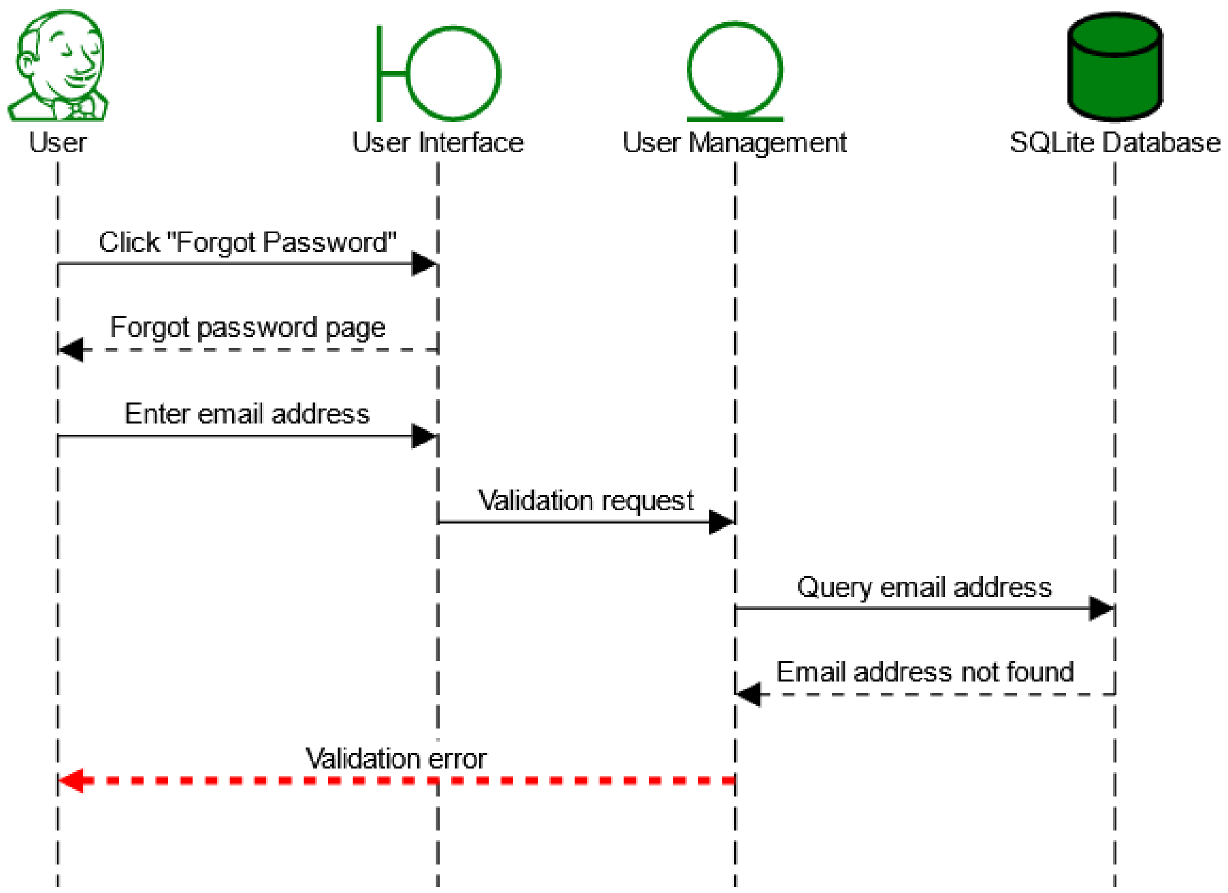
Error 3: **Reset password**

Description: User tries to reset password with invalid email address

Precondition: Email address does not exist in database

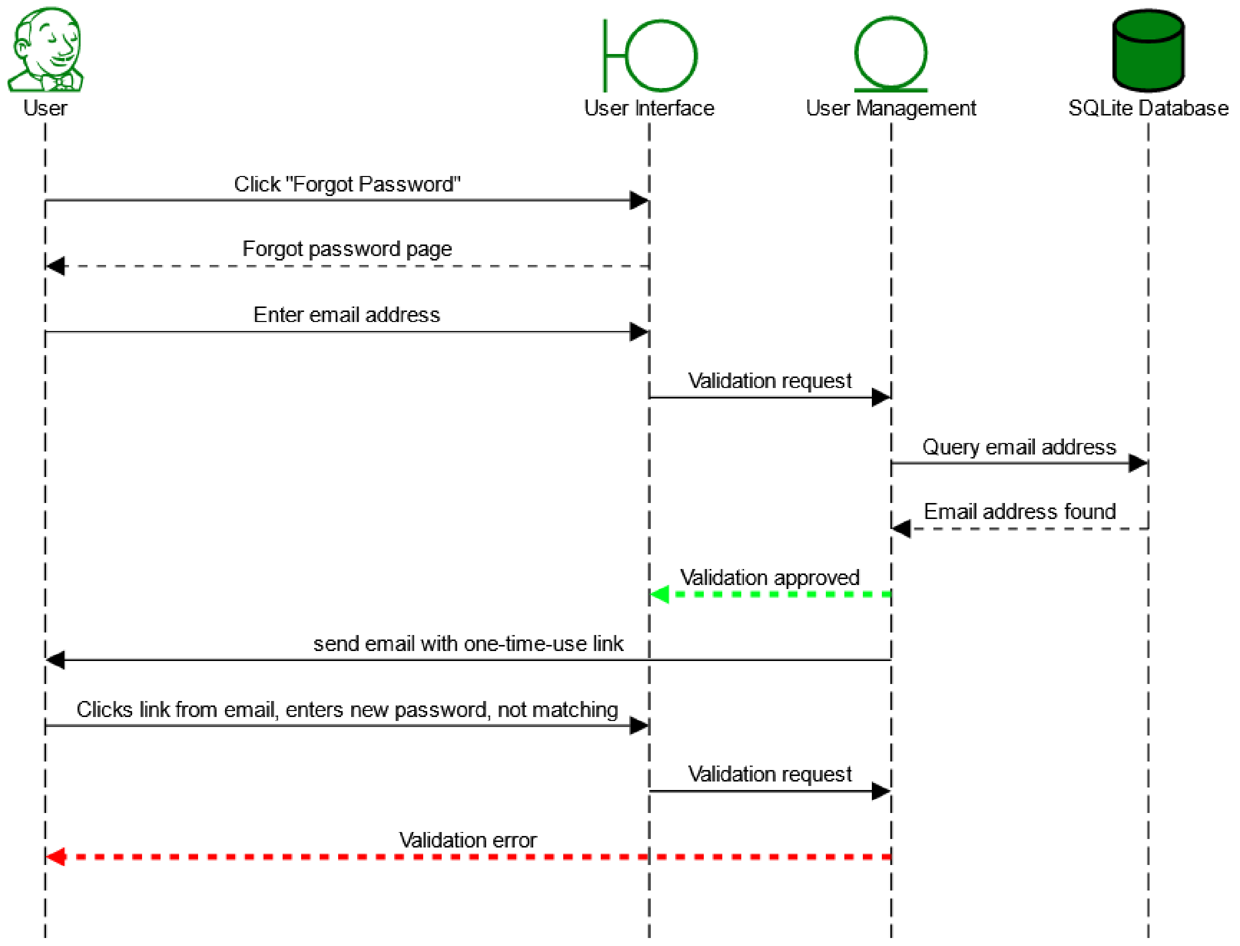
Postcondition: User receives an error message until valid email is entered

Error 3: Reset Password



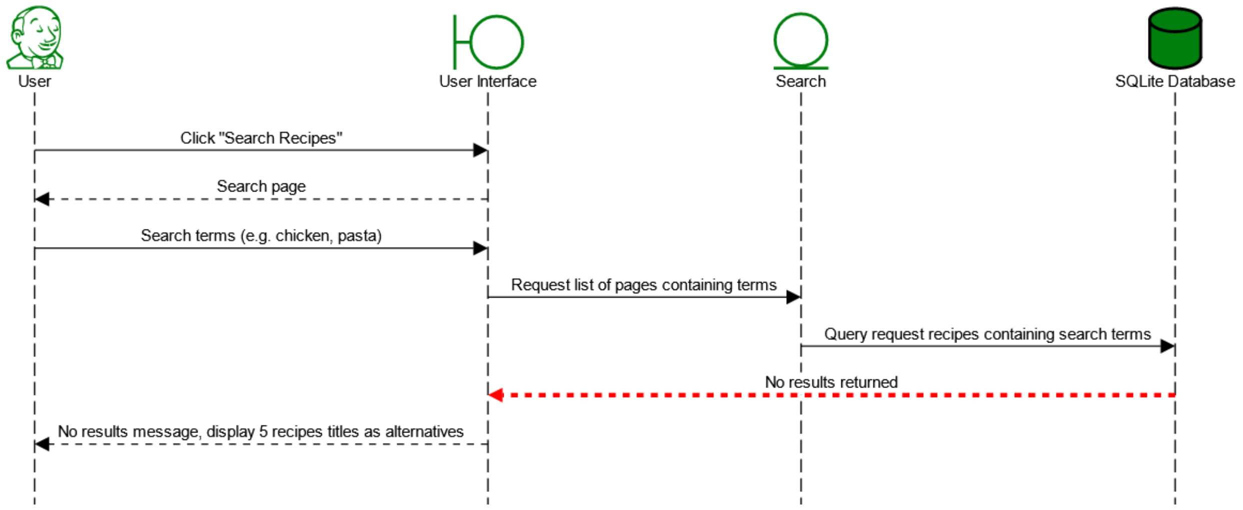
Error 4:	**Reset password**
Description:	User tries to reset password but new passwords do not match
Precondition:	User exists in database
Postcondition:	User receives an error message until valid fields are entered

Error 4: Reset Password



Error 5:	**Search recipes**
Description:	User searches for recipe term but does not match any in database
Precondition:	Search term does not match any recipe title in database
Postcondition:	No results message and five most recent recipes created

Error 5: Search and No Results


User
User Interface
Search
SQLite Database
Click "Search Recipes"
Search page
Search terms (e.g. chicken, pasta)
Request list of pages containing terms
Query request recipes containing search terms
No results returned
No results message, display 5 recipes titles as alternatives

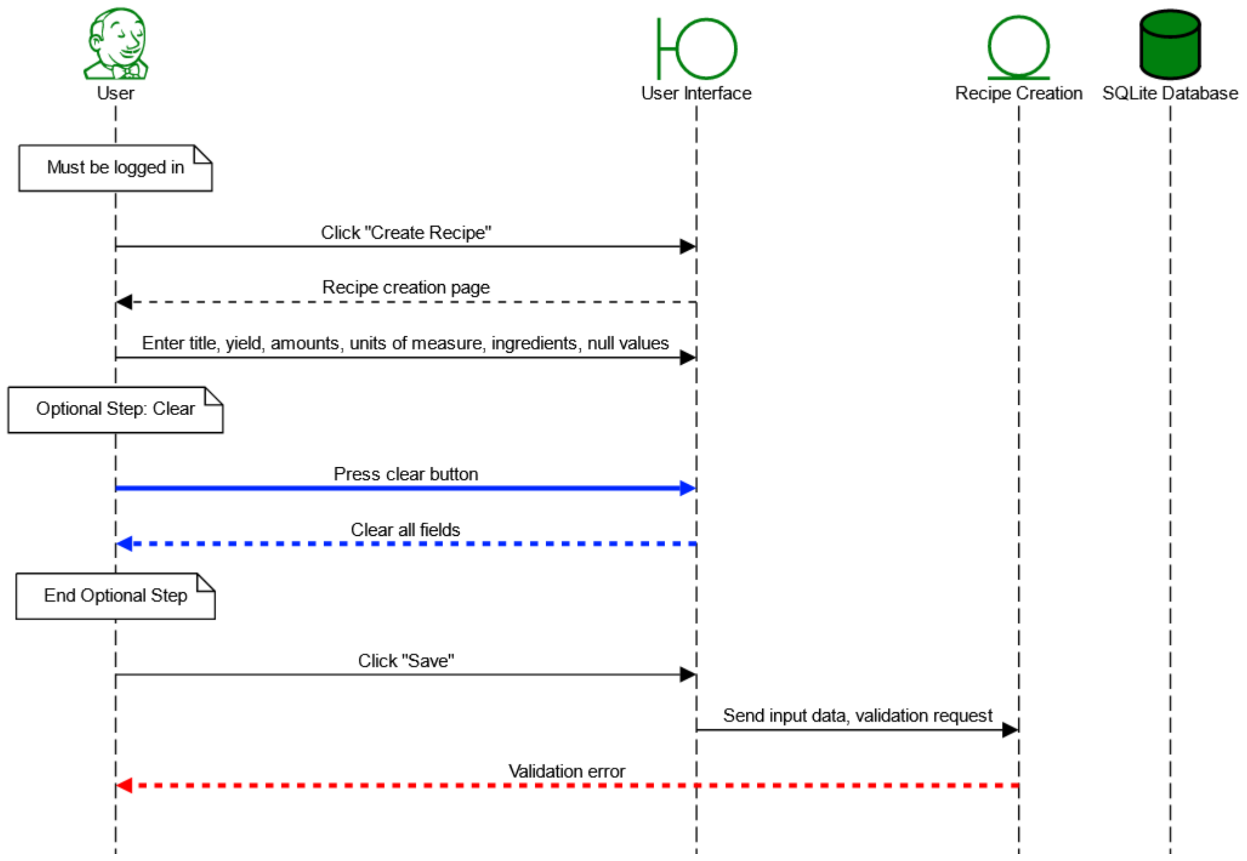
Error 6: **Create recipe**

Description: User tries to create a recipe but enters null values in required fields

Precondition: User must be logged in

Postcondition: User receives an error message until valid fields are entered

Error 6: Create Recipe



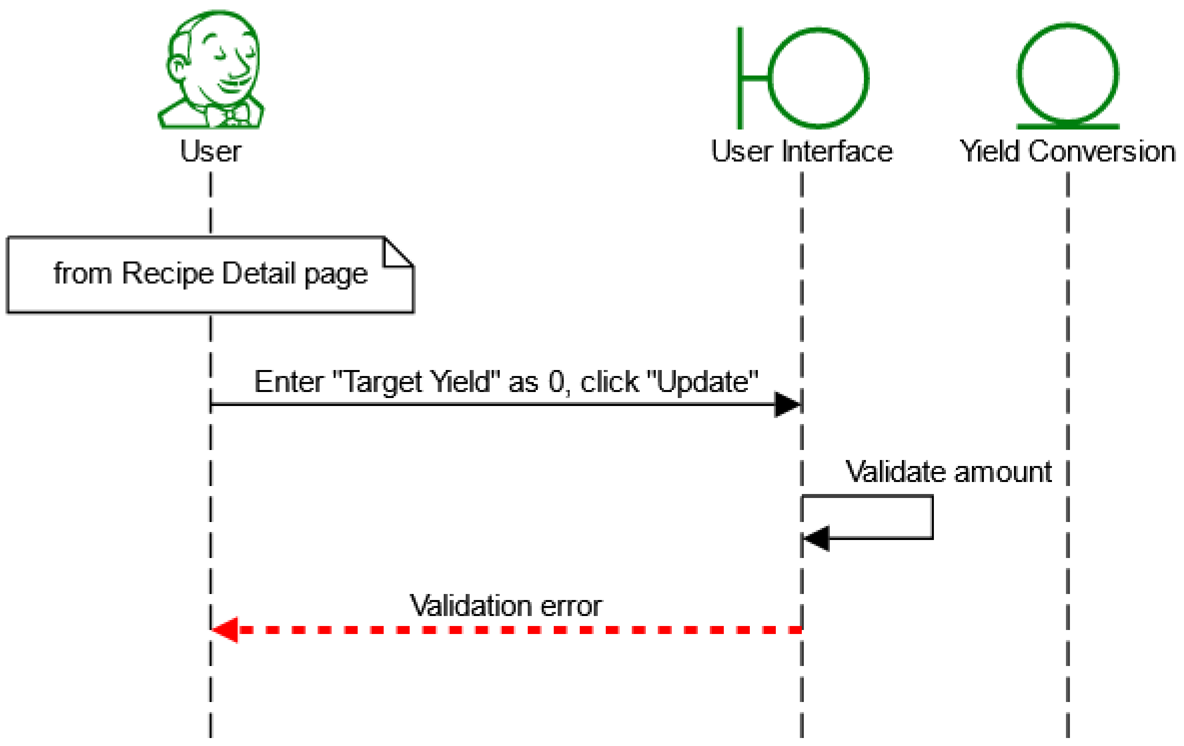
Error 7: **Yield conversion**

Description: User enters 0 for target yield and tries to convert recipe

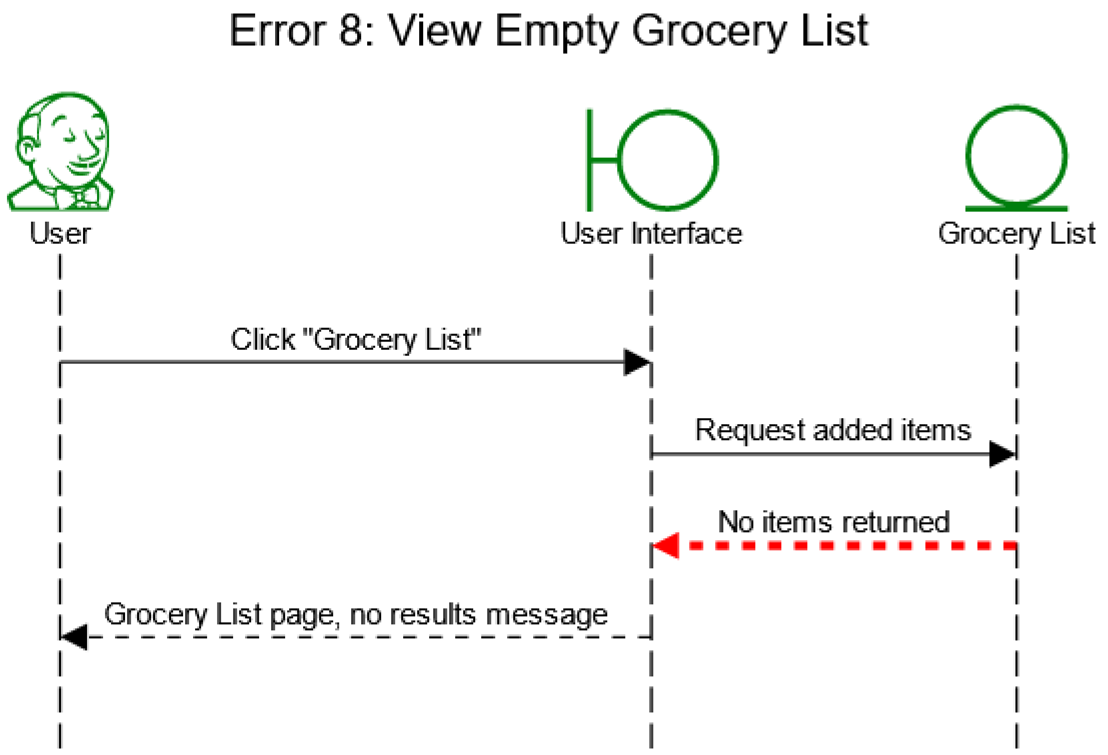
Precondition: Recipe exists in database

Postcondition: User receives an error message until valid fields are entered

Error 7: Yield Conversion



Error 8:	**Empty grocery list**
Description:	User tries to view grocery list with no items
Precondition:	User must be logged in
Postcondition:	Grocery list page displays no results message



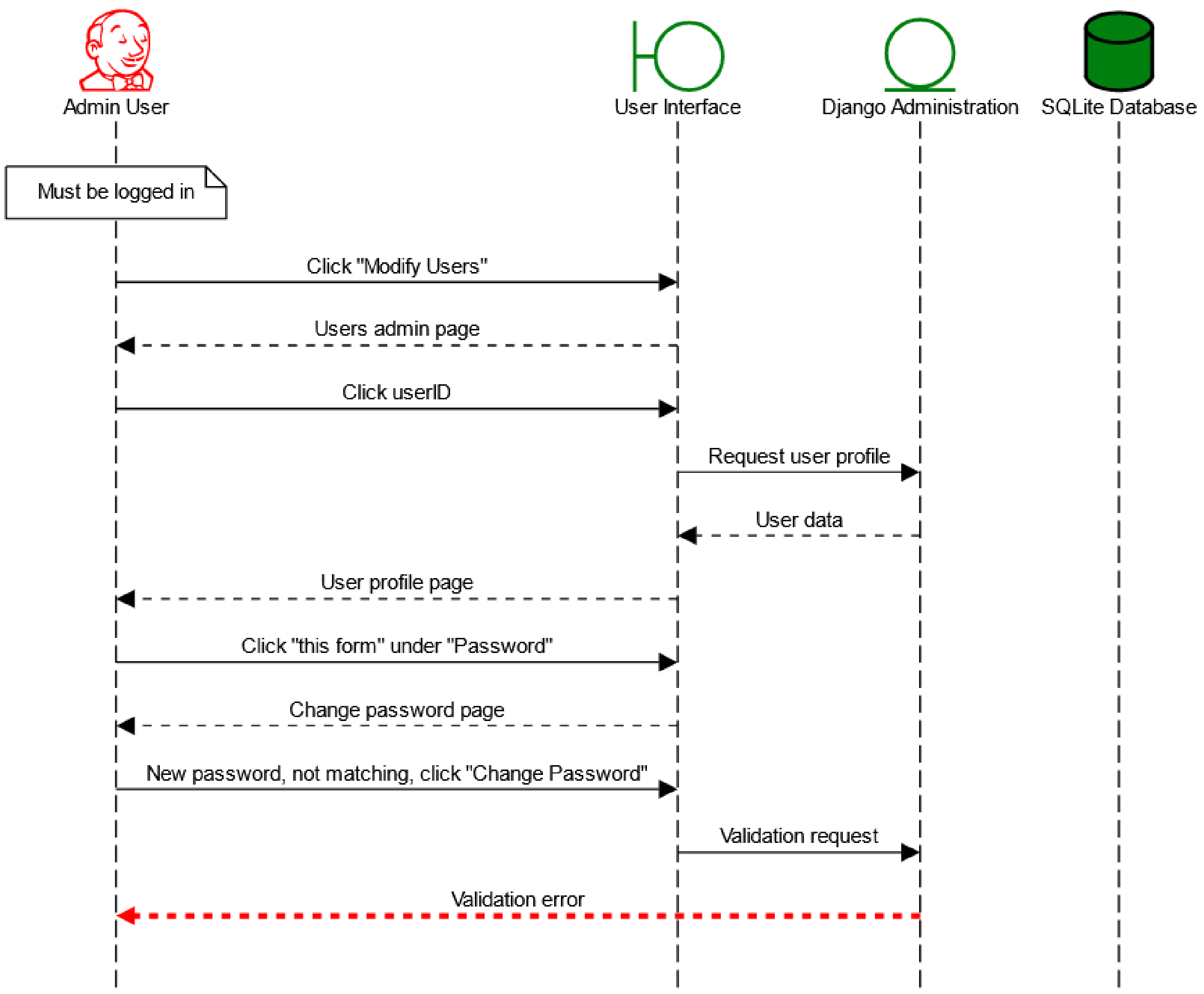
Error 9: **Admin reset password**

Description: Admin tries to reset password for a user but new password does not match validation

Precondition: User to modify exists in database, admin must be logged in

Postcondition: User receives an error message until valid fields are entered

Error 9: Reset Password



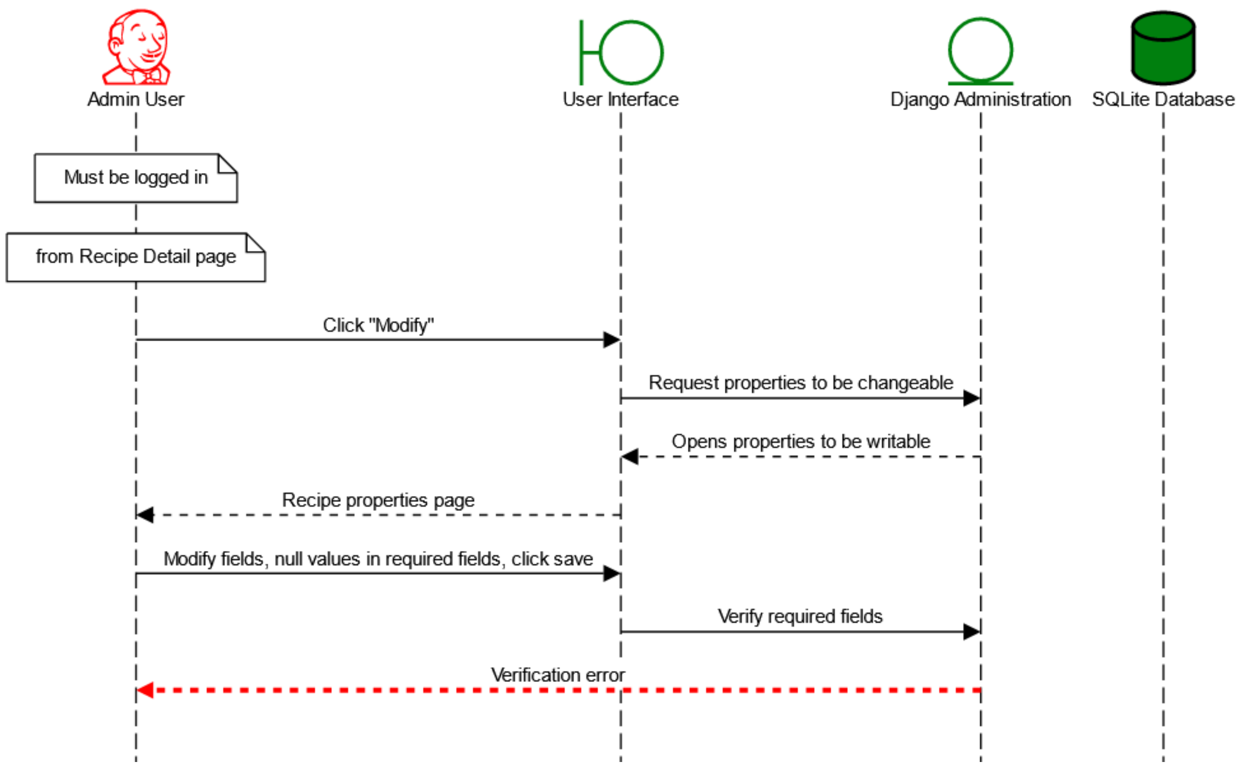
Error 10: **Modify recipe**

Description: Admin tries to modify recipe but has null values in required fields

Precondition: Admin must be logged in, recipe exists in database

Postcondition: User receives an error message until valid fields are entered

Error 10: Modify Recipe



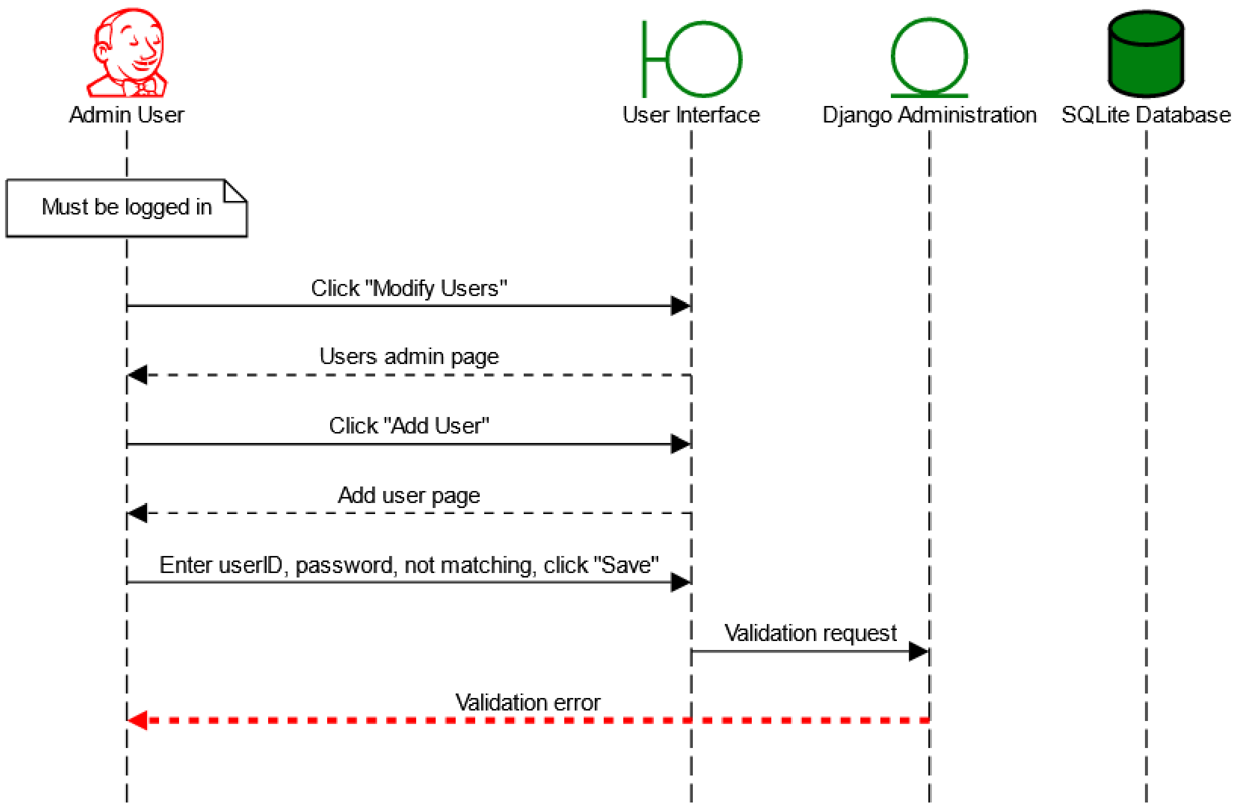
Error 11: **Create new admin users**

Description: Admin tries to create a new admin user but enters null values in required fields

Precondition: Admin must be logged in, recipient user does not already exist in database

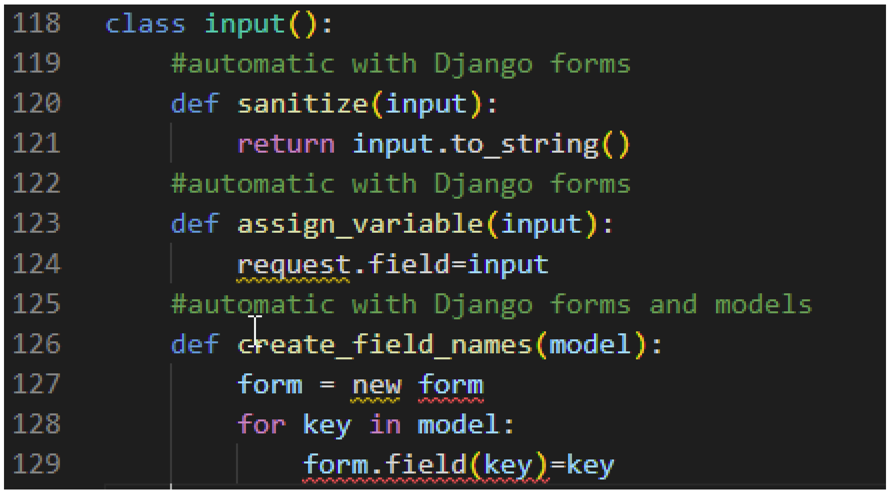
Postcondition: User receives an error message until valid fields are entered

Error 11: Create new Admin Users



Class Design:

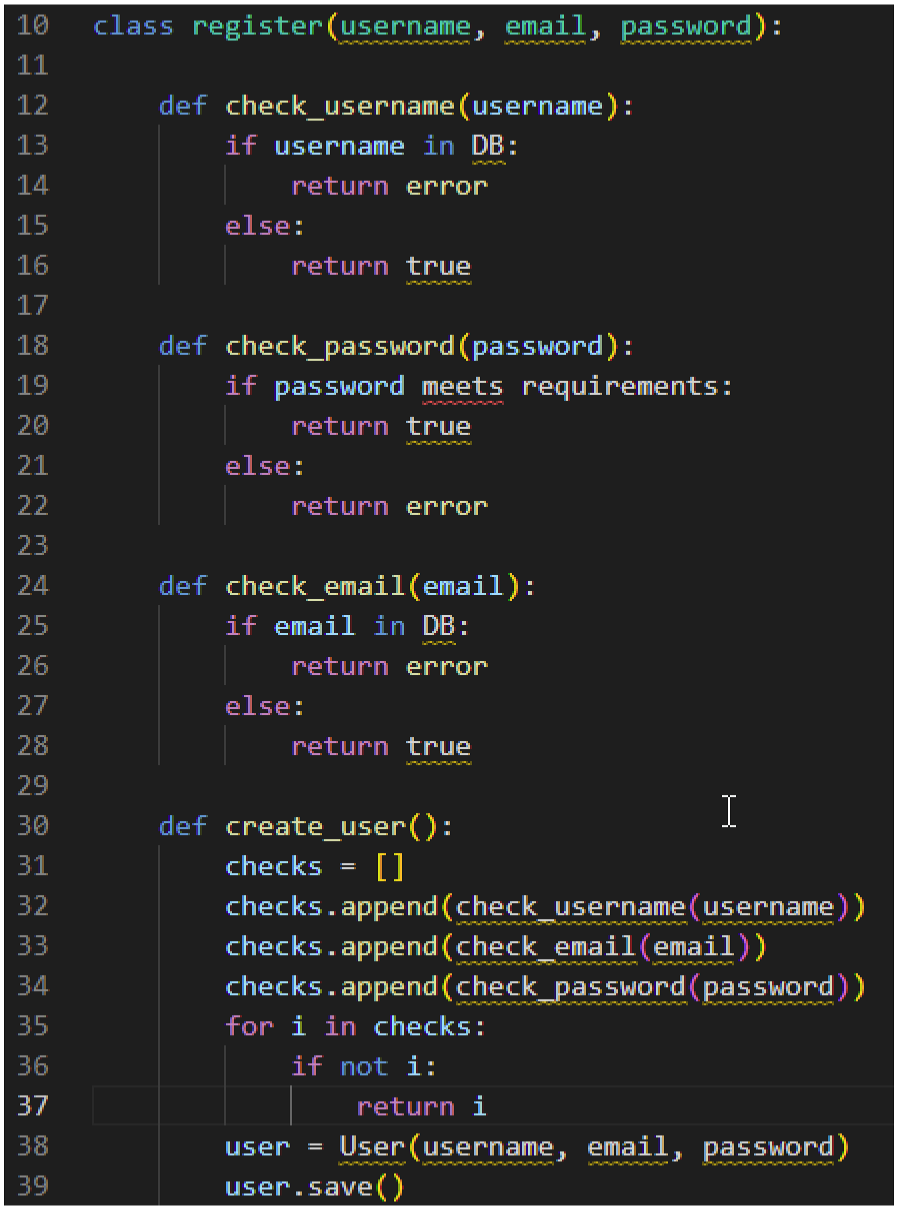
Input Sub-System



```
class input():
    #automatic with Django forms
    def sanitize(input):
        return input.to_string()
    #automatic with Django forms
    def assign_variable(input):
        request.field=input
    #automatic with Django forms and models
    def create_field_names(model):
        form = new form
        for key in model:
            form.field(key)=key
```

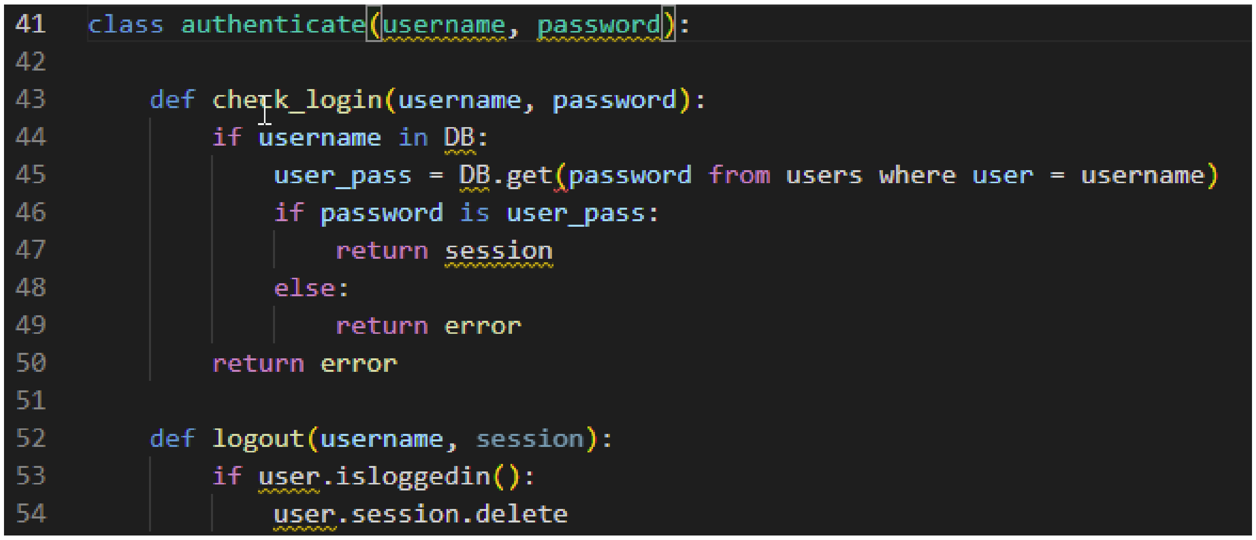
User Management:

1. Class user register



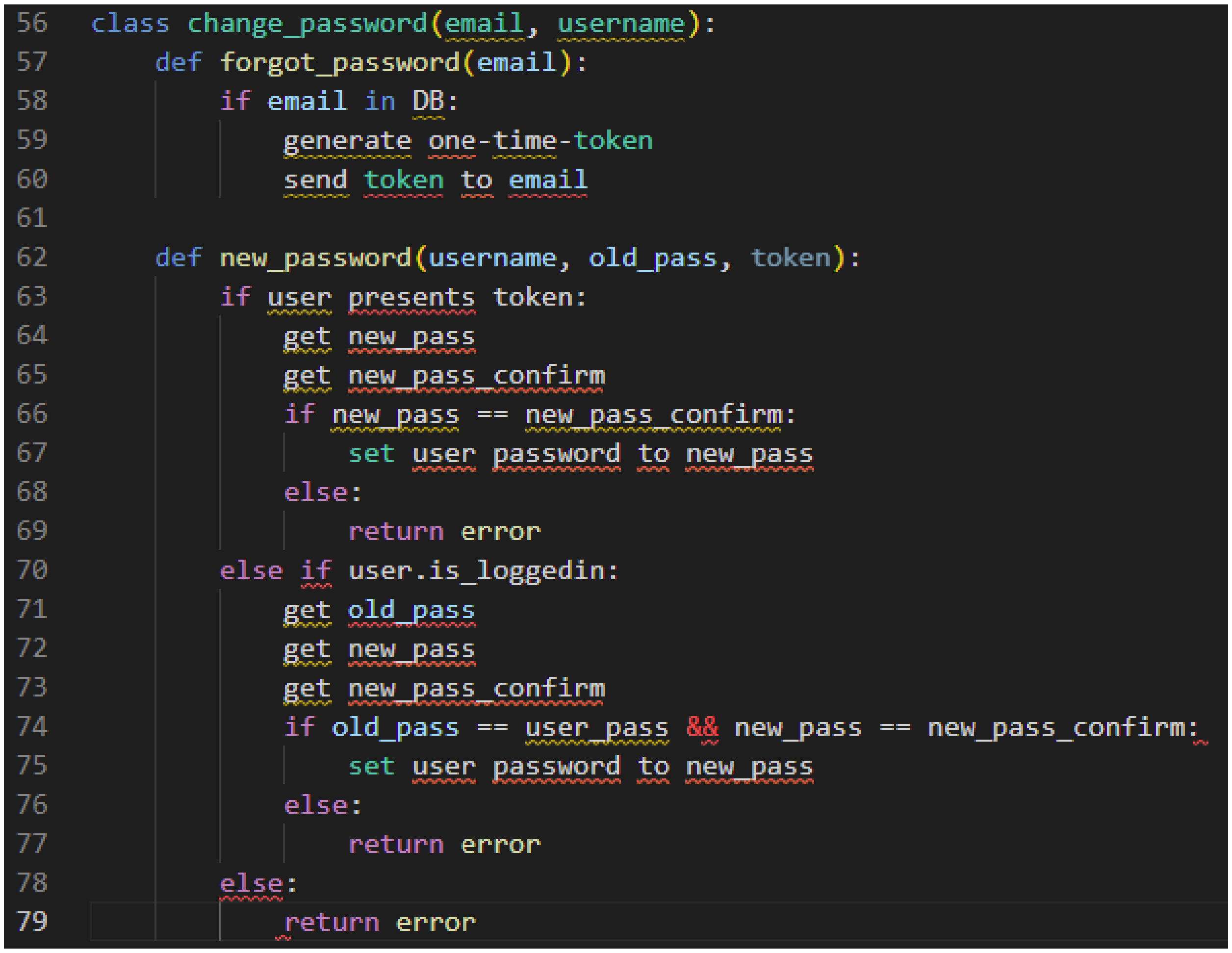
```
class register(username, email, password):

    def check_username(username):
        if username in DB:
            return error
        else:
            return true

    def check_password(password):
        if password meets requirements:
            return true
        else:
            return error

    def check_email(email):
        if email in DB:
            return error
        else:
            return true

    def create_user():
        checks = []
        checks.append(check_username(username))
        checks.append(check_email(email))
        checks.append(check_password(password))
        for i in checks:
            if not i:
                return i
        user = User(username, email, password)
        user.save()
```

2. Class Authenticate



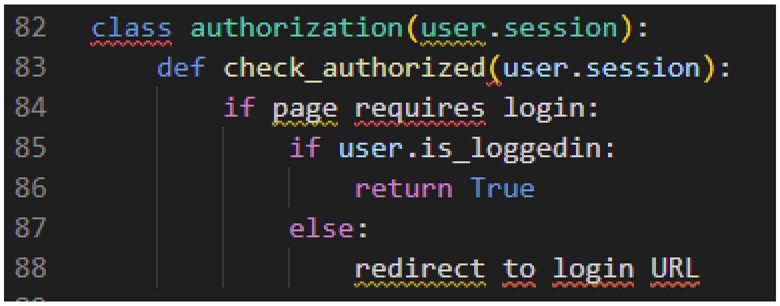
```
class authenticate(username, password):

    def check_login(username, password):
        if username in DB:
            user_pass = DB.get(password from users where user = username)
            if password is user_pass:
                return session
            else:
                return error
        return error

    def logout(username, session):
        if user.isloggedin():
            user.session.delete
```

3. Class change_password



```
class change_password(email, username):
    def forgot_password(email):
        if email in DB:
            generate one-time-token
            send token to email

    def new_password(username, old_pass, token):
        if user presents token:
            get new_pass
            get new_pass_confirm
            if new_pass == new_pass_confirm:
                set user password to new_pass
            else:
                return error
        else if user.is_loggedin:
            get old_pass
            get new_pass
            get new_pass_confirm
            if old_pass == user_pass && new_pass == new_pass_confirm:
                set user password to new_pass
            else:
                return error
        else:
            return error
```

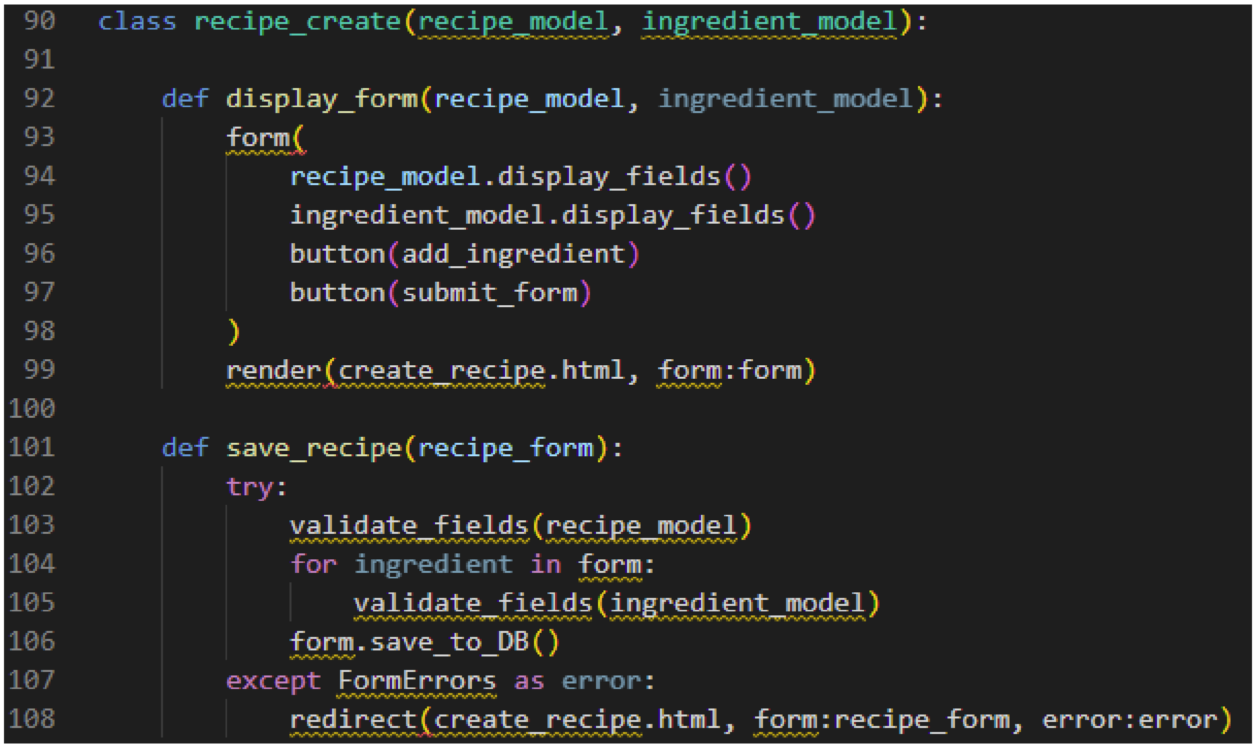
4. Class Authorization



```
class authorization(user.session):
    def check_authorized(user.session):
        if page requires login:
            if user.is_loggedin:
                return True
            else:
                redirect to login URL
```

Recipe Create:

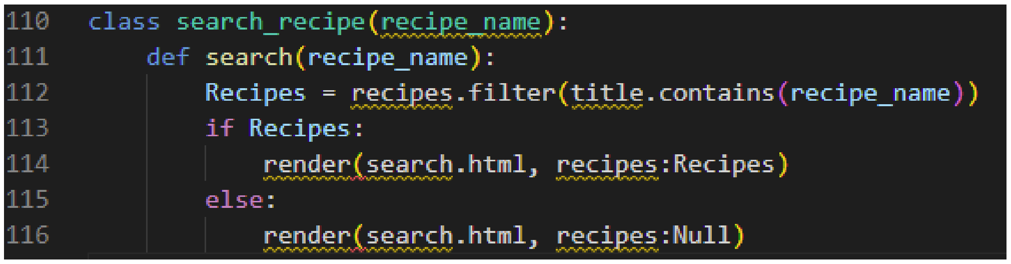
1. Class Recipe_Create



```
class recipe_create(recipe_model, ingredient_model):

    def display_form(recipe_model, ingredient_model):
        form(
            recipe_model.display_fields()
            ingredient_model.display_fields()
            button(add_ingredient)
            button(submit_form)
        )
        render(create_recipe.html, form:form)

    def save_recipe(recipe_form):
        try:
            validate_fields(recipe_model)
            for ingredient in form:
                validate_fields(ingredient_model)
            form.save_to_DB()
        except FormErrors as error:
            redirect(create_recipe.html, form:recipe_form, error:error)
```

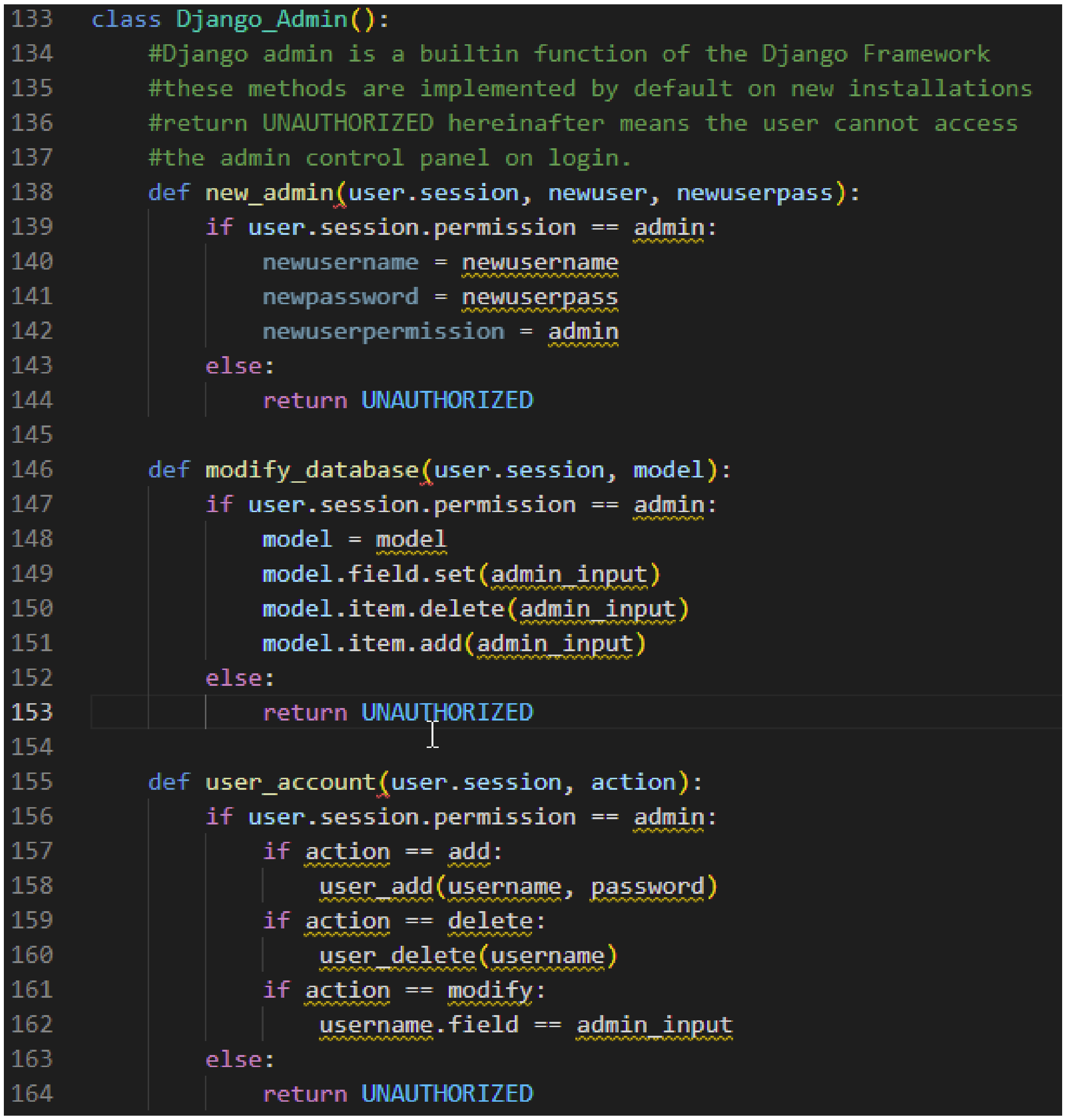
<u>Search_recipe:</u>

1. Class search



```
class search_recipe(recipe_name):
    def search(recipe_name):
        Recipes = recipes.filter(title.contains(recipe_name))
        if Recipes:
            render(search.html, recipes:Recipes)
        else:
            render(search.html, recipes:Null)
```

Django Admin:

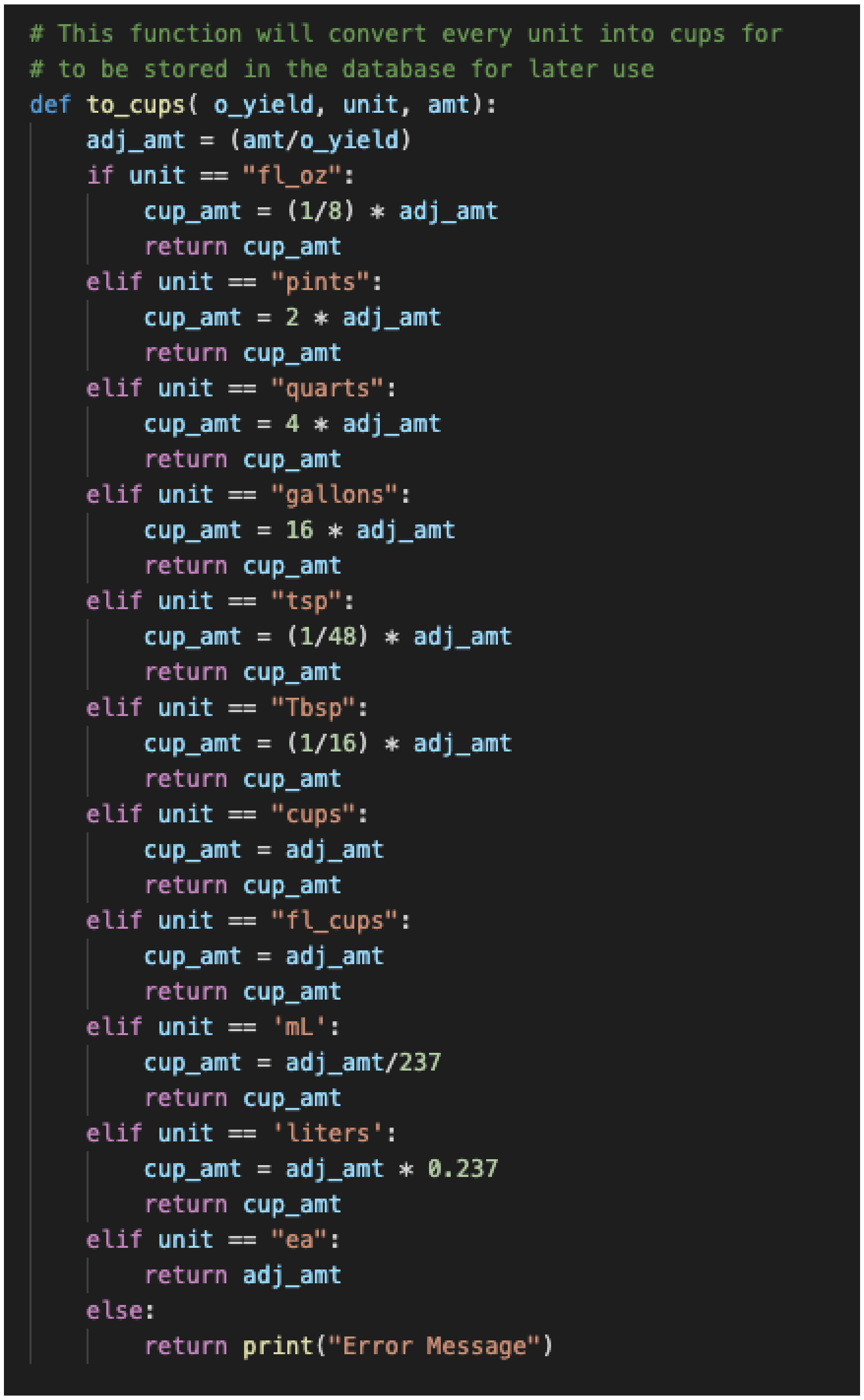


```
class Django_Admin():
    #Django admin is a builtin function of the Django Framework
    #these methods are implemented by default on new installations
    #return UNAUTHORIZED hereinafter means the user cannot access
    #the admin control panel on login.
    def new_admin(user.session, newuser, newuserpass):
        if user.session.permission == admin:
            newusername = newusername
            newpassword = newuserpass
            newuserpermission = admin
        else:
            return UNAUTHORIZED

    def modify_database(user.session, model):
        if user.session.permission == admin:
            model = model
            model.field.set(admin_input)
            model.item.delete(admin_input)
            model.item.add(admin_input)
        else:
            return UNAUTHORIZED

    def user_account(user.session, action):
        if user.session.permission == admin:
            if action == add:
                user_add(username, password)
            if action == delete:
                user_delete(username)
            if action == modify:
                username.field == admin_input
        else:
            return UNAUTHORIZED
```

Convert yield:

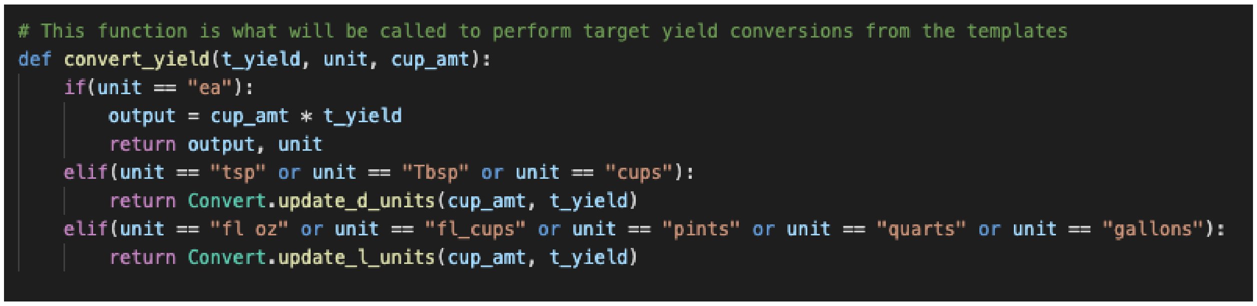
1. Class Convert

   a. Function to_cups



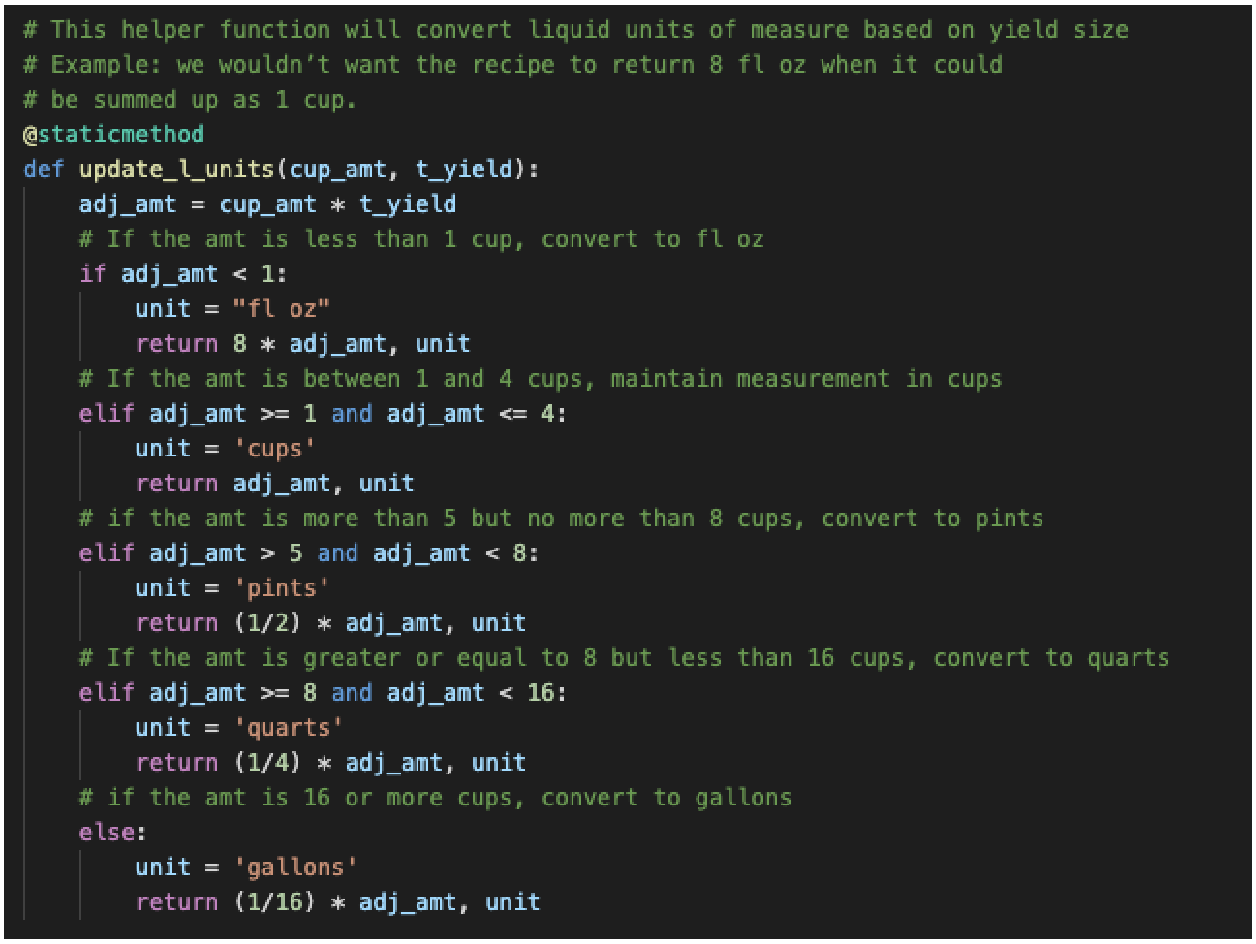
```python
# This function will convert every unit into cups for
# to be stored in the database for later use
def to_cups( o_yield, unit, amt):
    adj_amt = (amt/o_yield)
    if unit == "fl_oz":
        cup_amt = (1/8) * adj_amt
        return cup_amt
    elif unit == "pints":
        cup_amt = 2 * adj_amt
        return cup_amt
    elif unit == "quarts":
        cup_amt = 4 * adj_amt
        return cup_amt
    elif unit == "gallons":
        cup_amt = 16 * adj_amt
        return cup_amt
    elif unit == "tsp":
        cup_amt = (1/48) * adj_amt
        return cup_amt
    elif unit == "Tbsp":
        cup_amt = (1/16) * adj_amt
        return cup_amt
    elif unit == "cups":
        cup_amt = adj_amt
        return cup_amt
    elif unit == "fl_cups":
        cup_amt = adj_amt
        return cup_amt
    elif unit == 'mL':
        cup_amt = adj_amt/237
        return cup_amt
    elif unit == 'liters':
        cup_amt = adj_amt * 0.237
        return cup_amt
    elif unit == "ea":
        return adj_amt
    else:
        return print("Error Message")
```

b. Function convert_yield



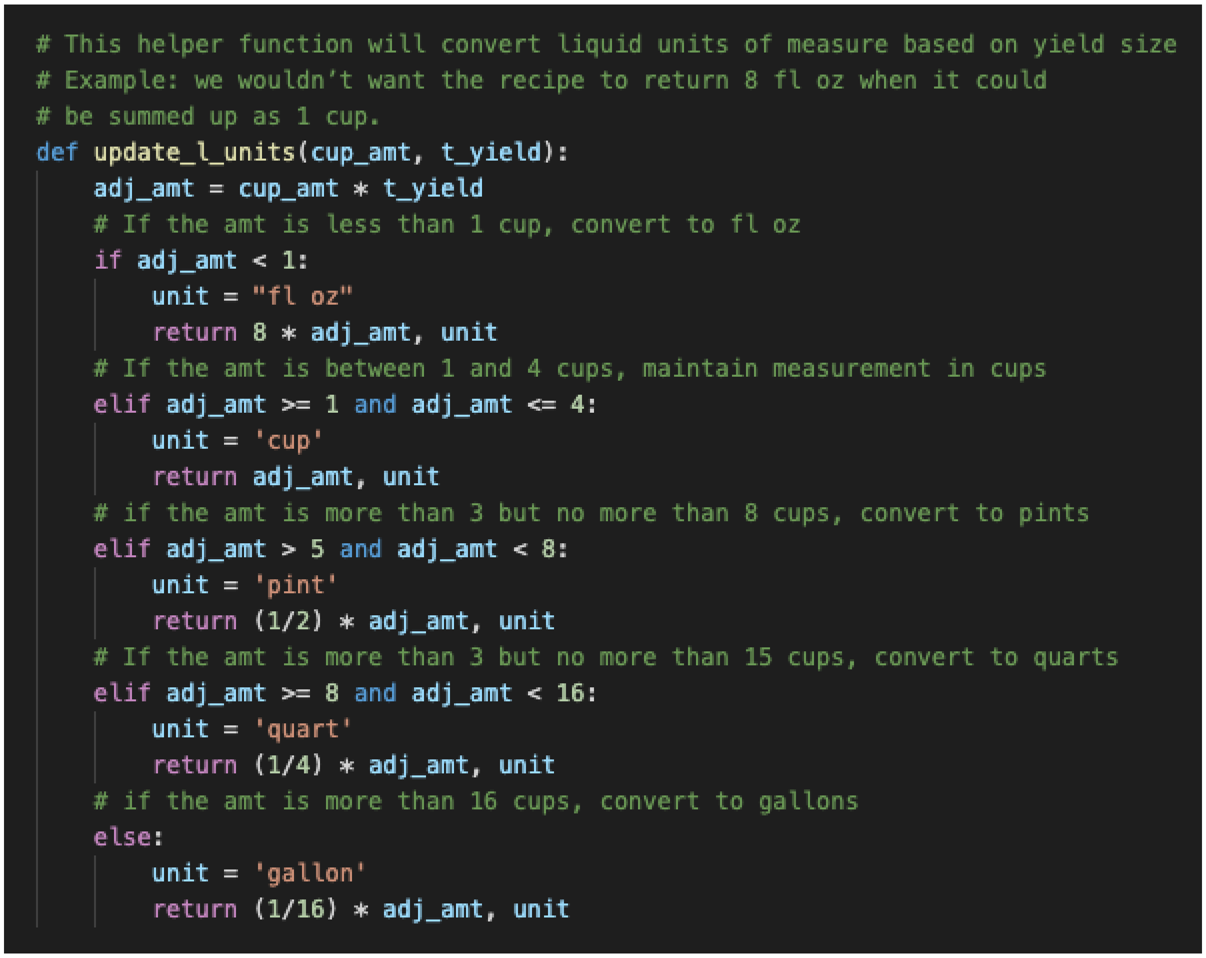
```
# This function is what will be called to perform target yield conversions from the templates
def convert_yield(t_yield, unit, cup_amt):
    if(unit == "ea"):
        output = cup_amt * t_yield
        return output, unit
    elif(unit == "tsp" or unit == "Tbsp" or unit == "cups"):
        return Convert.update_d_units(cup_amt, t_yield)
    elif(unit == "fl oz" or unit == "fl_cups" or unit == "pints" or unit == "quarts" or unit == "gallons"):
        return Convert.update_l_units(cup_amt, t_yield)
```

c. Function update_l_units



```
# This helper function will convert liquid units of measure based on yield size
# Example: we wouldn't want the recipe to return 8 fl oz when it could
# be summed up as 1 cup.
@staticmethod
def update_l_units(cup_amt, t_yield):
    adj_amt = cup_amt * t_yield
    # If the amt is less than 1 cup, convert to fl oz
    if adj_amt < 1:
        unit = "fl oz"
        return 8 * adj_amt, unit
    # If the amt is between 1 and 4 cups, maintain measurement in cups
    elif adj_amt >= 1 and adj_amt <= 4:
        unit = 'cups'
        return adj_amt, unit
    # if the amt is more than 5 but no more than 8 cups, convert to pints
    elif adj_amt > 5 and adj_amt < 8:
        unit = 'pints'
        return (1/2) * adj_amt, unit
    # If the amt is greater or equal to 8 but less than 16 cups, convert to quarts
    elif adj_amt >= 8 and adj_amt < 16:
        unit = 'quarts'
        return (1/4) * adj_amt, unit
    # if the amt is 16 or more cups, convert to gallons
    else:
        unit = 'gallons'
        return (1/16) * adj_amt, unit
```

d. Function update_d_units

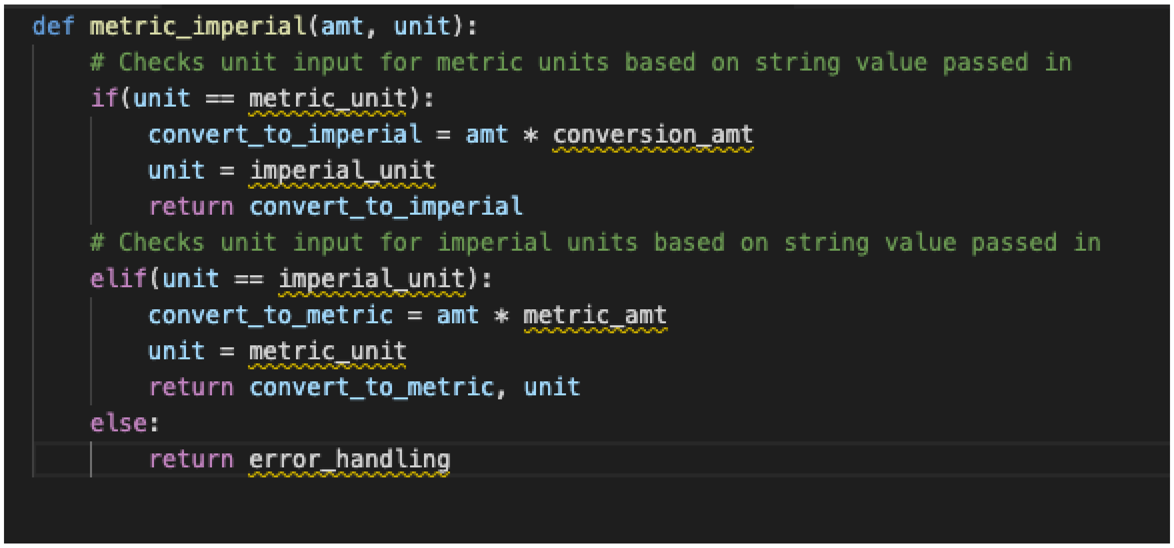


```
# This helper function will convert liquid units of measure based on yield size
# Example: we wouldn't want the recipe to return 8 fl oz when it could
# be summed up as 1 cup.
def update_l_units(cup_amt, t_yield):
    adj_amt = cup_amt * t_yield
    # If the amt is less than 1 cup, convert to fl oz
    if adj_amt < 1:
        unit = "fl oz"
        return 8 * adj_amt, unit
    # If the amt is between 1 and 4 cups, maintain measurement in cups
    elif adj_amt >= 1 and adj_amt <= 4:
        unit = 'cup'
        return adj_amt, unit
    # if the amt is more than 3 but no more than 8 cups, convert to pints
    elif adj_amt > 5 and adj_amt < 8:
        unit = 'pint'
        return (1/2) * adj_amt, unit
    # If the amt is more than 3 but no more than 15 cups, convert to quarts
    elif adj_amt >= 8 and adj_amt < 16:
        unit = 'quart'
        return (1/4) * adj_amt, unit
    # if the amt is more than 16 cups, convert to gallons
    else:
        unit = 'gallon'
        return (1/16) * adj_amt, unit
```

Convert between metric and imperial:

1. Class Convert:

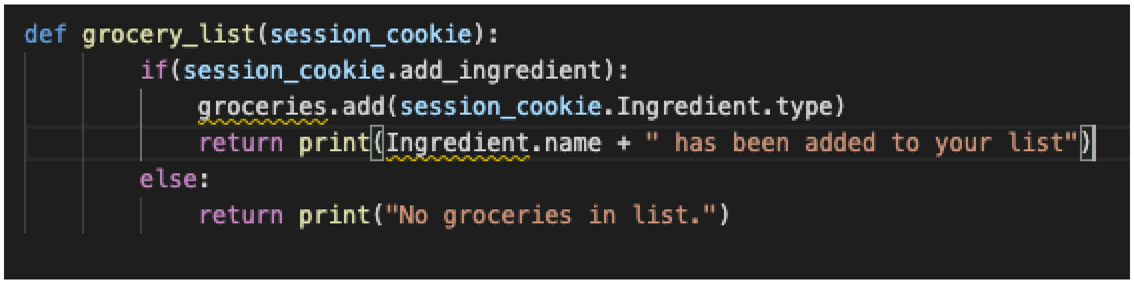
    a. Function metric_imperial



```python
def metric_imperial(amt, unit):
    # Checks unit input for metric units based on string value passed in
    if(unit == metric_unit):
        convert_to_imperial = amt * conversion_amt
        unit = imperial_unit
        return convert_to_imperial
    # Checks unit input for imperial units based on string value passed in
    elif(unit == imperial_unit):
        convert_to_metric = amt * metric_amt
        unit = metric_unit
        return convert_to_metric, unit
    else:
        return error_handling
```

Grocery list:

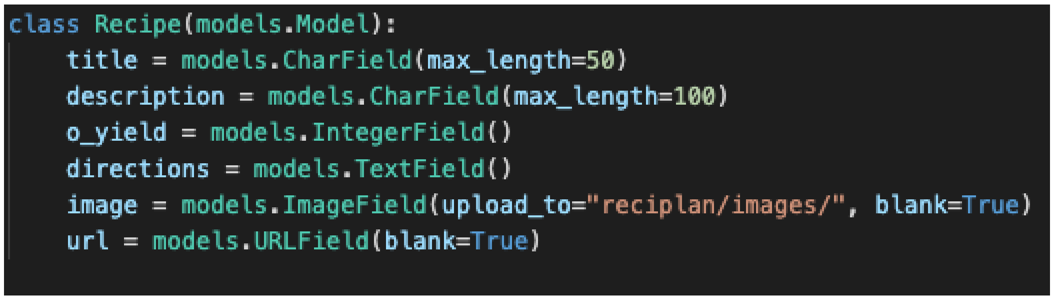
1. Class Groceries (IN WORK)



```python
def grocery_list(session_cookie):
        if(session_cookie.add_ingredient):
            groceries.add(session_cookie.Ingredient.type)
            return print(Ingredient.name + " has been added to your list")
        else:
            return print("No groceries in list.")
```

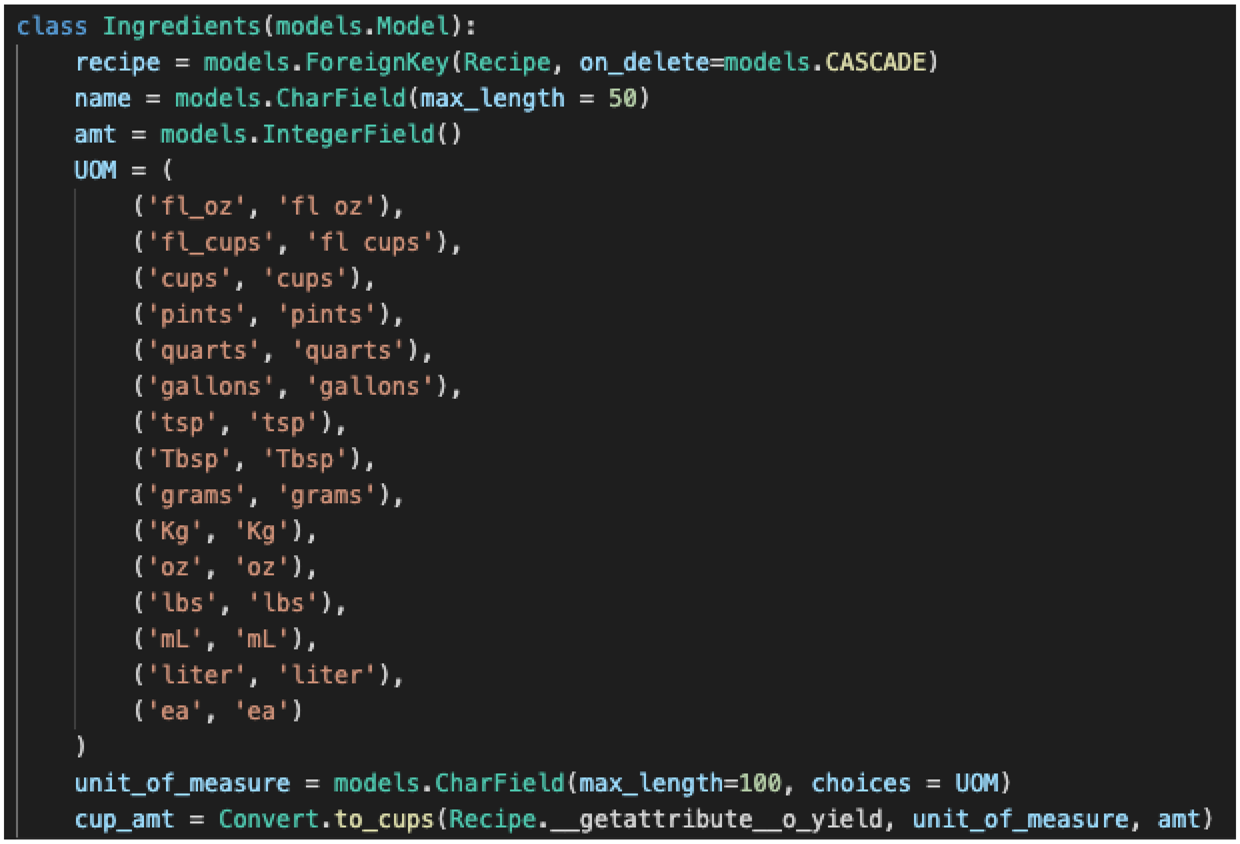
<u>SQLite DB:</u>

1. Class Recipe



```
class Recipe(models.Model):
    title = models.CharField(max_length=50)
    description = models.CharField(max_length=100)
    o_yield = models.IntegerField()
    directions = models.TextField()
    image = models.ImageField(upload_to="reciplan/images/", blank=True)
    url = models.URLField(blank=True)
```

2. Class Ingredients



```
class Ingredients(models.Model):
    recipe = models.ForeignKey(Recipe, on_delete=models.CASCADE)
    name = models.CharField(max_length = 50)
    amt = models.IntegerField()
    UOM = (
        ('fl_oz', 'fl oz'),
        ('fl_cups', 'fl cups'),
        ('cups', 'cups'),
        ('pints', 'pints'),
        ('quarts', 'quarts'),
        ('gallons', 'gallons'),
        ('tsp', 'tsp'),
        ('Tbsp', 'Tbsp'),
        ('grams', 'grams'),
        ('Kg', 'Kg'),
        ('oz', 'oz'),
        ('lbs', 'lbs'),
        ('mL', 'mL'),
        ('liter', 'liter'),
        ('ea', 'ea')
    )
    unit_of_measure = models.CharField(max_length=100, choices = UOM)
    cup_amt = Convert.to_cups(Recipe.__getattribute__o_yield, unit_of_measure, amt)
```

<u>Possible Enhancements:</u>

- Shopping cart aggregation function to add all items of same type together
- Online shopping API integration to direct order from Walmart/Instacart or similar

- Recipe scraping from popular sites such as recipes.com
- Comment box for recipes to share how you liked them

Possible Risks and Mitigations:

Injection attacks on input system: These are mitigated by using Django's built in forms which automatically sanitize data inputs.

Database overload/storage space restriction: If the application receives more than the expected traffic a postgresql AWS database will be added to handle the overflow. Migration will be performed to ensure no loss of user accounts or recipe data.