Business Case: Enhancing Recipe Adaptability with Smart Ingredient Substitutions in the Tasty API Recipe Finder

Objective Statement Many people struggle with meal preparation due to missing ingredients, dietary restrictions, or lack of time. The Tasty API Recipe Finder allows users to search for recipes based on available ingredients, but what happens when a key ingredient is missing?

By integrating Smart Ingredient Substitutions, users can replace unavailable ingredients with commonly used alternatives. This ensures they can still prepare meals without requiring a last-minute grocery run, making the platform more flexible and user-friendly.

How It Works: Smart Ingredient Substitution Process

1. User Inputs Dietary Preferences – The system uses the selected dietary preferences as part of the recipe filtering process
2. User Inputs Ingredients - The system uses the entered ingredients to generate a list of recipes along with their cook time, calories and nutritional value
3. User Selects a Recipe – Selecting a recipe then displays the full ingredient list and instructions are displayed.
4. Substitutions Specified - System asks user if they would like to substitutes for any of their originally entered ingredients by asking a yes/no based question.
5. User Accepts or Modifies Substitutions – If YES, recipes are re-generated using the substitutes from two information sources: The Old Farmer's Almanac ingredient substitution table & the All Recipes ingredient substitution table
6. Final Recipe Instructions Provided – Users receive information on the quantity and item to make their desired substitution.

```
# Importing libraries for API calls, web scraping, and data handling
import requests
import json
import pandas as pd
from functools import lru_cache
from io import StringIO  # Used to wrap HTML content for pd.read_html

# Setting the RapidAPI host and API key for authentication
API_HOST = "tasty.p.rapidapi.com"
API_KEY = "e753a9ff95mshe59abd90ed3c1dep16d51cjsn4112a1a9e9b1"

# Function to search for recipes based on dietary preference and available ingredients
def search_recipes(diet, ingredients, size=5):
    query = ", ".join(ingredients)
    url = f"https://{API_HOST}/recipes/list"
    params = {"tags": diet if diet != "none" else None, "q": query, "from": 0, "size": size}
    headers = {"X-RapidAPI-Host": API_HOST, "X-RapidAPI-Key": API_KEY}

    response = requests.get(url, headers=headers, params=params)
    if response.status_code != 200:
        raise Exception(f"Failed to fetch recipes: {response.status_code} - {response.text}")

    results = response.json().get("results", [])
    if not results:
        print("No recipes found for the given criteria.")
        return pd.DataFrame()

    recipes = []
    for r in results:
        nutrition = r.get("nutrition", {})
        recipes.append({
            "ID": r["id"],
            "Name": r["name"],
            "Cook Time (mins)": r.get("cook_time_minutes", "N/A"),
            "Calories (kcal)": nutrition.get("calories", "N/A"),
            "Protein (g)": nutrition.get("protein", "N/A"),
            "Carbs (g)": nutrition.get("carbohydrates", "N/A"),
            "Fat (g)": nutrition.get("fat", "N/A")
        })
    return pd.DataFrame(recipes)

# Function to retrieve detailed recipe information via caching
@lru_cache(maxsize=100)
def get_recipe_details(recipe_id):
    url = f"https://{API_HOST}/recipes/get-more-info"
    params = {"id": recipe_id}
    headers = {"X-RapidAPI-Host": API_HOST, "X-RapidAPI-Key": API_KEY}

    response = requests.get(url, headers=headers, params=params)
    if response.status_code != 200:
        raise Exception(f"Failed to fetch recipe details: {response.status_code} - {response.text}")

    return response.json()

# New functionality as part of project 2: Web-scraping of Ingredient Substitutions Tables

def build_substitution_dict(substitutions_df):
    subs_dict = {}
    # Assumes the table has at least three columns:
    # Column 0: Ingredient name,
    # Column 1: Original amount,
    # Column 2: Substitution details.
    for _, row in substitutions_df.iterrows():
        try:
```

```
            ingredient = str(row[0]).lower().strip()
            original_amount = str(row[1]).strip()
            substitution_detail = str(row[2]).strip()
            suggestion = f"Substitute {original_amount} {ingredient} with {substitution_detail}."
            subs_dict[ingredient] = suggestion
        except Exception as e:
            continue
    return subs_dict

def load_substitutions_from_allrecipes():
    url = "https://www.allrecipes.com/article/common-ingredient-substitutions/"
    try:
        tables = pd.read_html(url)
        if tables:
            # Choose the table with the most columns (assumed to be the substitutions table)
            substitutions_df = max(tables, key=lambda t: t.shape[1])
            subs_dict = build_substitution_dict(substitutions_df)
            return subs_dict
        else:
            print("No tables found on the substitutions page (All Recipes).")
            return {}
    except Exception as e:
        print("Error loading substitutions table from All Recipes:", e)
        return {}

def load_substitutions_from_almanac():
    url = "https://www.almanac.com/content/common-ingredient-substitutions"
    # Set a User-Agent header to mimic a browser
    headers = {
        "User-Agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/114.0.0.0 Safari/537.36"
    }
    try:
        response = requests.get(url, headers=headers)
        response.raise_for_status()  # Raise an HTTPError for bad responses
        # Wrap the HTML content in StringIO to address the FutureWarning.
        tables = pd.read_html(StringIO(response.text))
        if tables:
            # Choose the table with the most columns (assumed to be the substitutions table)
            substitutions_df = max(tables, key=lambda t: t.shape[1])
            subs_dict = build_substitution_dict(substitutions_df)
            return subs_dict
        else:
            print("No tables found on the substitutions page (Almanac).")
            return {}
    except Exception as e:
        print("Error loading substitutions table from Almanac:", e)
        return {}

def merge_substitutions(dict1, dict2):
    """Merge two substitution dictionaries so that each ingredient maps to a list of suggestions."""
    merged = {}
    for source_dict in (dict1, dict2):
        for ingredient, suggestion in source_dict.items():
            key = ingredient.lower().strip()
            if key in merged:
                merged[key].append(suggestion)
            else:
                merged[key] = [suggestion]
    return merged

def get_ingredient_substitution(ingredient, subs_dict):
    key = ingredient.lower().strip()
    if key in subs_dict:
        # Return all suggestions, joined by newlines.
        return "\n".join(subs_dict[key])
    else:
        return f"No substitution found for '{ingredient}'."


# Main interactive script
if __name__ == "__main__":
    try:
        # Load substitutions from both All Recipes and Almanac.
        allrecipes_subs = load_substitutions_from_allrecipes()
        almanac_subs = load_substitutions_from_almanac()
        # Merge the dictionaries so that each ingredient key contains a list of suggestions.
        substitution_dict = merge_substitutions(allrecipes_subs, almanac_subs)

        # Step 1 - Ask the user for dietary preference
        diet = input("Enter your dietary preference (e.g., vegetarian, keto, gluten-free, or none): ").strip().lower()
        if not diet:
            diet = "none"  # Default to 'none' if no preference is specified

        # Step 2 - Ask for ingredients
        ingredients_input = input("Enter the ingredients in your fridge, separated by commas: ").split(",")
        ingredients = [ingredient.strip() for ingredient in ingredients_input]

        # Step 3 - Fetch recipes based on user input
        print("\nSearching for recipes...")
        recipes_df = search_recipes(diet, ingredients, size=8)

        if recipes_df.empty:
            print("No recipes found. Please try different inputs.")
        else:
            # Add numeric index as a column for easier selection and display
            recipes_df.insert(0, "#", range(1, len(recipes_df) + 1))
            print("\nRecipe Recommendations (with Nutritional Information):")
```

```
        print(recipes_df.to_string(index=False))

        # Step 4 - Ask the user to select a recipe
        recipe_number = int(input("\nEnter the recipe number to view details: "))
        if recipe_number < 1 or recipe_number > len(recipes_df):
            print("Invalid recipe number. Please restart and try again.")
        else:
            selected_recipe = recipes_df.iloc[recipe_number - 1]
            recipe_id = selected_recipe["ID"]

            # Fetch and display detailed recipe information
            print(f"\nFetching details for recipe: {selected_recipe['Name']}...")
            recipe_details = get_recipe_details(recipe_id)

            print(f"\nRecipe Name: {recipe_details['name']}")
            print("\nIngredients:")
            for section in recipe_details.get("sections", []):
                for component in section.get("components", []):
                    print(f"- {component['raw_text']}")

            print("\nInstructions:")
            for instruction in recipe_details.get("instructions", []):
                print(f"{instruction['position']}. {instruction['display_text']}")

            # ----------------------------------------------------------------------
            # New Step: Offer ingredient substitution suggestions using the merged substitutions table
            user_choice = input("\nWould you like to see an ingredient substitution suggestion? (yes/no): ").strip().lower()
            if user_choice in ["yes", "y"]:
                ingredient_to_substitute = input("Enter the ingredient you need a substitution for: ")
                substitution_result = get_ingredient_substitution(ingredient_to_substitute, substitution_dict)
                print(f"\nSubstitution suggestion(s) for {ingredient_to_substitute}:")
                print(substitution_result)
    except Exception as e:
        print(f"Error: {e}")
```

```
Enter your dietary preference (e.g., vegetarian, keto, gluten-free, or none): vegetarian
Enter the ingredients in your fridge, separated by commas: egg,onion,tomato

Searching for recipes...

Recipe Recommendations (with Nutritional Information):
 #   ID                                        Name  Cook Time (mins)  Calories (kcal)  Protein (g)  Carbs (g)  Fat (g)
 1  489          Cheesy Egg Toast Perfect For Breakfast                15              215           11          3       17
 2 1048                             Egg Breakfast Cups                20               87            6          5        4
 3  880 North African-Style Poached Eggs In Tomato Sauce                35              204           11         19        8
 4  384        Poached Eggs In Tomato Sauce (Shakshouka)                20              172           10         20        6
 5 1629          Muffin Tin Customizable Veggie Egg Cups                20               61            5          0        4
 6 1809                      Black Bean & Corn Burgers                10              553           25         82       13
 7 1371                   Spinach & Mushroom Quesadilla                15              683           42         32       43
 8  519                       Egg White Breakfast Cups                15              121           10          1        7

Enter the recipe number to view details: 7

Fetching details for recipe: Spinach & Mushroom Quesadilla...

Recipe Name: Spinach & Mushroom Quesadilla

Ingredients:
- 1 tablespoon olive oil
- ½ cup mushrooms, sliced
- 2 cloves garlic
- 3 cups fresh spinach
- Salt, to taste
- Pepper, to taste
- 3 eggs
- 2 large flour tortillas
- 1 cup mozzarella, shredded (double for 2 quesadillas)
- ½ cup parmesan, shredded (double for 2 quesadillas)
- Parsley
- Salsa

Instructions:
1. Let the oil heat up in the skillet and add the garlic followed by the mushrooms. Cook until the mushrooms have softened and caramelized a bit.
2. Add the spinach and cook until spinach has wilted.
3. Crack in the eggs and scramble with the veggies. Season with salt and pepper, and stir until fully cooked. Remove from the pan and set aside.
4. Place the tortilla in the skillet and add a layer of both cheeses on half of the tortilla.
5. Add the scramble, top with more cheese and fold the tortilla in half.
6. Cook for 6 minutes over medium heat, flipping half way.
7. Serve with salsa and garnish with fresh parsley.
8. Enjoy!

Would you like to see an ingredient substitution suggestion? (yes/no): yes
Enter the ingredient you need a substitution for: egg

Substitution suggestion(s) for egg:
Substitute 1 whole (3 tablespoons or 1.7 oz) egg with 2 1/2 tablespoons of powdered egg substitute plus 2 1/2 tablespoons water OR 1/4 cup liquid egg substitute OR 1/4 cup
```