NUTRITION ASSISTANT APPLICATION

TEAM ID :PNT2022TMID18519

TEAM LEAD: SHAILAJA V

TEAM MEMBER 1: SHABRIN BEGUM S

TEAM MEMBER 2: SANGEETH KUMAR A

TEAM MEMBER 3: SATVIK SREERAM V

Functional Requirements

Functional Requirements

Following are the functional requirements of the proposed solution

FR No	Functional Requirements	Sub Requirement(Story/Sub-Task)
FR-1	User Registration	Registration through email
FR-2	User Confirmation	Confirmation via email
FR-3	Data Collection	Collection of all required input data
FR-4	Data Analysis	Process the given inputs using CNN and Nutrition API
FR-5	Data processing	Evaluate the data and store it in database and integrate in cloud containers
FR-6	Provide output to user	Display the result to the user

Non-Functional Requirements

Following are the non-functional requirements of the proposed solution

FR No	Non-Functional Requirements	Description
NFR-1	Usability	User-friendly and overall satisfaction of the user while using the website
NFR-2	Security	The website provides proper authentication and verification
NFR-3	Reliability	The site always provides reliable outputs and lacks failures
NFR-4	Performance	Provides 100% efficiency of the output
NFR-5	Availability	The product is readily available for all kinds of users when needed
NFR-6	Scalability	Effective in obtaining good accuracies

3.	Novelty / Uniqueness	The following features adds uniqueness to this project: • User interacts with the Web App to Load an image. • The image is passed to the server application, which uses Clarifai's AI-Driven Food Detection Model Service to analyse the images and Nutrition API to provide nutritional information about the analysed Image. • Nutritional information of the analysed image is returned to the app for display.
4.	Social Impact / Customer Satisfaction	• The customer satisfaction is very well improved for getting appropriate response from the nutritionist immediately. • As soon as the user uploads the image, within few seconds the prediction result page will be displayed which highly improves the customer satisfaction. • The key customer satisfaction factor is consistency. The consistency is the secret ingredient to make customers happy.
5.	Business Model (Revenue Model)	• Different algorithms are used to correctly map the food items that has been uploaded by the user. • Appropriate algorithms are used to display the prediction result page.
6.	Scalability of the Solution	• The project is capable of handling growth, especially in handling more users and evolving concurrently with the business needs. • It also relates to the app's backend, database and the servers they are hosted on which is highly satisfied by this project.

Solution Architecture

Solution Architecture:

Solution architecture is a complex process – with many sub-processes – that bridges the gap between business problems and technology solutions. Its goals are to:

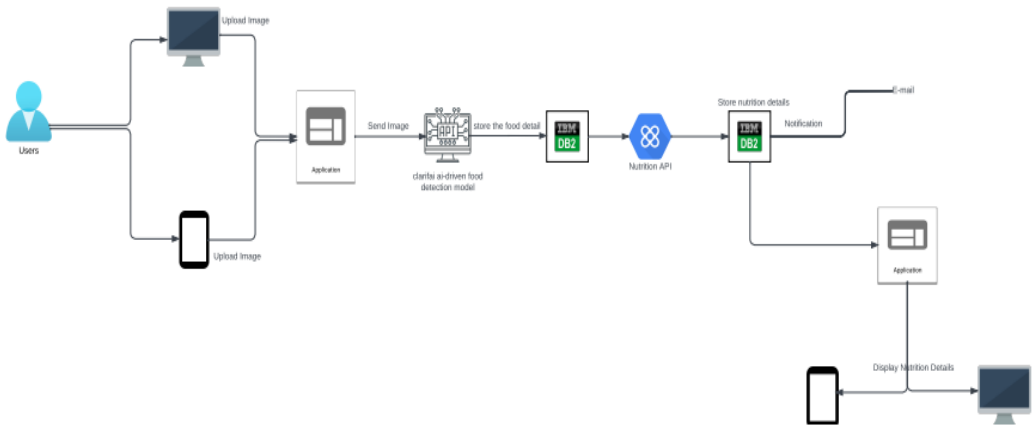
• Find the best tech solution to solve existing business problems.

• Describe the structure, characteristics, behavior, and other aspects of the software to project stakeholders.

• Define features, development phases, and solution requirements.

• Provide specifications according to which the solution is defined, managed, and delivered.

Solution Architecture Diagram:



Solution Fit Template

1. CUSTOMER SEGMENT(S)

- Working peoples
- Organizations
- Students and families
- Common people with all ages can able to track their expenses.

2. JOBS-TO-BE-DONE / PROBLEMS

- People have to track their expenses regularly.
- They need to keep their receipts and bills which shows their amount they spent.
- Also they need to manually add or remove the desired categories.

3.CUSTOMER CONSTRAINTS

- Network Issues
- Data Privacy
- Spending power
- Available devices

4.AVAILABLE SOLUTIONS

- People makes use of sticky notes or diary for knowing their diets.
- Pros:
    1. Didn’t need any devices for diary.
- Cons:
    1. Time consuming.
    2. Manual errors occur sometimes.

5.BEHAVIOUR

- People should know their diet for each meal and set appropriate health goals.
- Collect receipts regularly without fail.

LITRATURE SURVEY

DESCRIPTION

Personal Nutritionist as the name, the system can act as your personal nutritionist while this system can be used also by nutritionist gaining a lot of information and help in many ways. Fat Secret API helps the System to get the information in many ways. The user can get details about a number of nutrients, vitamins etc of a fruit or vegetable. The user can add his recipes or get recipes using the API.

The System basically helps the user in what to eat and which is good, what will help him and etc, the system will help him filter things easily.The System also allows the user to make a diet plan and remind him his food timings.

PROS & CONS

Advantages :

1. The user is allowed to make a diet plan with the help of RECIPE-FOOD-NUTRITION API the user can gain a lot of information about a food or eatable.
2. The application does not contain login so everyone can make their plan easily.

Disadvantages :

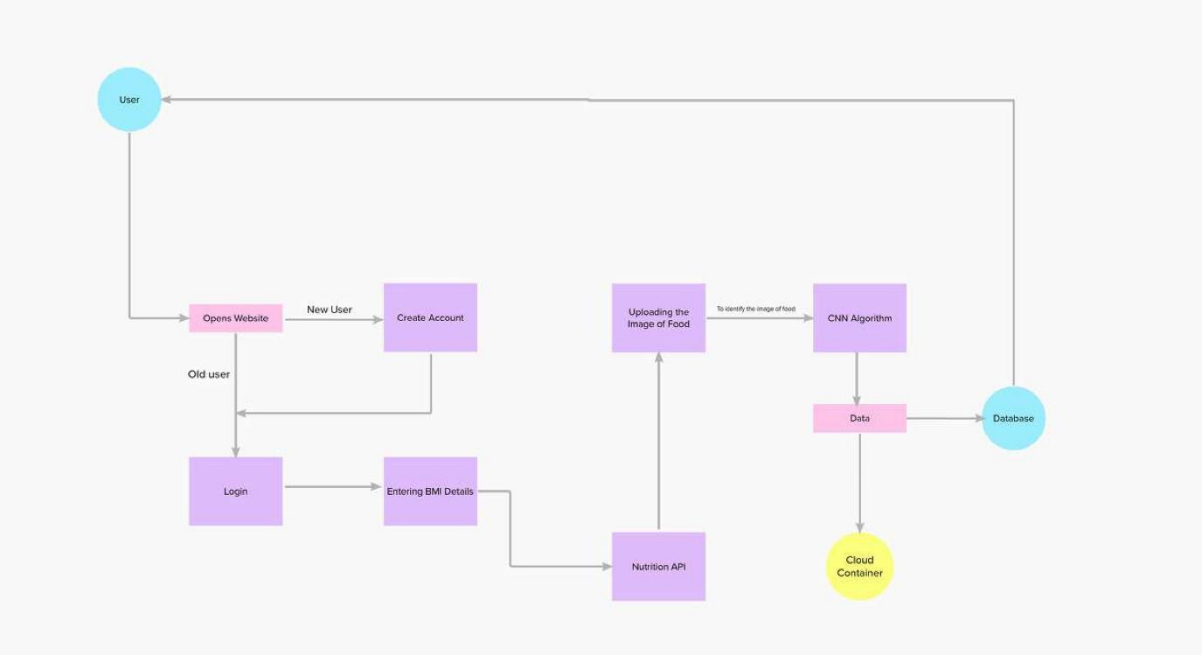
1. The system needs active Internet Connection.
2. The data may be inaccurate if there is problem with internet.

EMPATHY MAP

Personal Nutritionist as the name, the system can act as your personal nutritionist while this system can be used also by nutritionist gaining a lot of information and help in many ways. Fat Secret API helps the System to get the information in many ways. The user can get details about a number of nutrients, vitamins etc of a fruit or vegetable. The user can add his recipes or get recipes using the API.
The System basically helps the user in what to eat and which is good, what will help him and etc, the system will help him filter things easily. The System also allows the user to make a diet plan and remind him his food timings.

Data Flow Diagrams & User Stories

Data Flow Diagram:



User Stories

User Type	Functional Requirements (Epic)	User Story Number	User Story/Task	Acceptance Criteria	Priority	Release
User(All common people)	User Registration	USN-1	As a user ,I can register for the application by entering my Name, email, password.	I can access my dashboard.	High	Sprint-1
	Login	USN-2	As a user, I can login to the application using my given credentials.	I can access my dashboard.	High	Sprint-1
	BMI Calculation	USN-3	As a user, I enter my height and weight details.	I can get to know about my BMI.	High	Sprint-1
	Uploading the Image	USN-4	As a user, I will upload the image of food that I want to eat.	I can upload the image to decide whether to eat or not.	High	Sprint-1
	Providing output to user	USN-5	As a user, I will get to know the results of the inputs I've given.	I will get to know if I can eat the food or not.	Medium	Sprint-2
Administrator	Data Analysis	USN-6	As an admin, I will develop algorithms and modules to process the data.	I can store the result in database.	High	Sprint-1
	Integrating with Cloud	USN-7	As a admin, I integrate the results in cloud containers.	I can deploy the data in cloud.	High	Sprint-1

Customer Journey Map

Bill Williamson

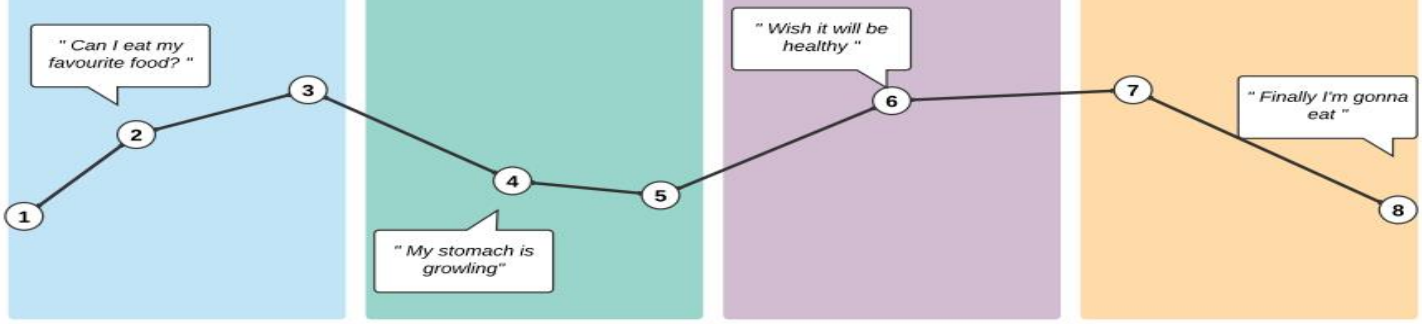
Scenario

Bill Williamson is an obese person who never walks and sits idle most of the time. He wants to check his nutrition and maintain his diet.

Expectations

- Become healthy
- Overcome his laziness
- Wants to maintain diet regularly

Decide	Consult the Tracker app	Suggestions by the app	Experience
1. Sees his friends are healthier than him. 2. He decides to maintain his diet and his daily nutrition consumption. 3. He searches on the Internet to find an Nutrition Assistant app.	4.Opens the nutrition tracker application . 5.Enters his BMI as asked by the application. 6. Also uploads the image of the food he wants to eat. 7.Waits fot the suggestion of the application.	7. Based on the BMI , the app decides whether he wants to eat the food or not. 8. If he is unhealthy, the application suggests other foods that are healthier and so that he can maintain his diet.	7. He may feel healthier. 8.Gets an idea about his daily diet. 9. His data is updated in database and cloud regularly, as the tracker always monitors his daily diet.



CODING:-

main.py

```
from flask import Flask, render_template, request, redirect, url_for, session
import ibm_db
import re
import http.client
import json
from werkzeug.utils import secure_filename
import math
import os


app = Flask(__name__)

app.secret_key = 'a'

conn = ibm_db.connect("DATABASE=bludb;HOSTNAME=6667d8e9-9d4d-4ccb-ba32-
21da3bb5aafc.c1ogj3sd0tgtu0lqde00.databases.appdomain.cloud;PORT=30376;SECURITY=SSL;SSLServerCertifi
cate=DigiCertGlobalRootCA.crt;UID=ntv37394;PWD=i5QfgqIlnGog6H1Y","")


@app.route("/signup")
def signup():
    return render_template("signup.html")

@app.route("/addrec",methods=['GET','POST'])
def addrec():
    msg = ''
    if request.method == 'POST':
        username = request.form['username']
        password = request.form['password']
        email = request.form['email']

        sql = "SELECT * FROM users1 WHERE username =?"
        stmt = ibm_db.prepare(conn, sql)
        ibm_db.bind_param(stmt, 1, username)
        ibm_db.execute(stmt)
        account = ibm_db.fetch_assoc(stmt)
        if account:
            return render_template("signup.html",msg="Already a user")
        else:
            insert_sql = "INSERT INTO users1 VALUES (?, ?, ?)"
            prep_stmt = ibm_db.prepare(conn, insert_sql)
            ibm_db.bind_param(prep_stmt, 1, username)
            ibm_db.bind_param(prep_stmt, 2, email)
            ibm_db.bind_param(prep_stmt, 3, password)

            ibm_db.execute(prep_stmt)
            msg = 'You have successfully registered !'

        return render_template('signup.html', msg=msg)
```

```
@app.route('/')
def login():
    return render_template('login.html')

@app.route('/login')
def login1():
    return render_template('login.html')

@app.route('/dashboard')
def dashboard():
    return render_template('dashboard.html')


@app.route('/authenticate',methods=['GET','POST'])
def authenticate():
    global userId;
    if request.method == 'POST':
        password = request.form['password']
        email = request.form['email']
        print(email)
        sql = "SELECT * FROM users1 WHERE email =? and password=?"
        stmt = ibm_db.prepare(conn, sql)
        ibm_db.bind_param(stmt, 1, email)
        ibm_db.bind_param(stmt, 2, password)
        ibm_db.execute(stmt)
        account = ibm_db.fetch_assoc(stmt)
        if account:
            session['loggedin']=True
            session['id']=account['EMAIL']
            userId=account['EMAIL']
            session['email']=account['EMAIL']
            return render_template("dashboard.html",msg=email)
        else:
            return render_template("login.html",msg="incorrect")


@app.route("/checkpass",methods=['GET','POST'])
def checkpass():
    msg=''
    if request.method == 'POST':
        password = request.form['password']
        email = request.form['email']
        sql = "select * from users1 where email=?"
        stmt = ibm_db.prepare(conn, sql)
        ibm_db.bind_param(stmt, 1, email)
        ibm_db.execute(stmt)
        account=ibm_db.fetch_assoc(stmt)
        if account:
            sql1="update users1 set password=? where email=?"
            stmt = ibm_db.prepare(conn, sql1)
            ibm_db.bind_param(stmt, 1, password)
            ibm_db.bind_param(stmt, 2, email)

            ibm_db.execute(stmt)
            return render_template('forgotpw.html',msg="changed")
        else:
```

```
        return render_template('forgotpw.html',msg="incorrect")
@app.route('/forgotpw')
def forgotpw():
    return render_template('forgotpw.html')

@app.route('/getnutri',methods=['GET','POST'])
def getnutri():
    if request.method=="POST":
        name=request.form['name']
        print(name)
        conn = http.client.HTTPSConnection("spoonacular-recipe-food-nutrition-v1.p.rapidapi.com")

        headers = {
            'X-RapidAPI-Key': "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx",
            'X-RapidAPI-Host': "spoonacular-recipe-food-nutrition-v1.p.rapidapi.com"
        }

        conn.request("GET", "/recipes/guessNutrition?title="+name, headers=headers)

        res = conn.getresponse()
        data = res.read()
        r=json.loads(data)
        val=len(r)

        if val == 2:
            return render_template("getnut.html",msg="invalid")
        else:
            calories = r["calories"]["value"]
            fat = r["fat"]["value"]
            protein = r["protein"]["value"]
            carbs = r["carbs"]["value"]
            def add():
                conn = ibm_db.connect(
                    "DATABASE=bludb;HOSTNAME=6667d8e9-9d4d-4ccb-ba32-
21da3bb5aafc.c1ogj3sd0tgtu0lqde00.databases.appdomain.cloud;PORT=30376;SECURITY=SSL;SSLServerCertifi
cate=DigiCertGlobalRootCA.crt;UID=ntv37394;PWD=i5QfgqIlnGog6H1Y",
                    ", ")
                insert_sql = "INSERT INTO historyi VALUES (?, ?, ?, ?, ?)"
                calories1=round(calories,2)
                protein1=round(protein,2)
                fat1 = round(fat, 2)
                carbs1 = round(carbs, 2)
                print(calories1)
                prep_stmt = ibm_db.prepare(conn, insert_sql)
                ibm_db.bind_param(prep_stmt, 1, name)
                ibm_db.bind_param(prep_stmt, 2, calories1)
                ibm_db.bind_param(prep_stmt, 3, protein1)
                ibm_db.bind_param(prep_stmt, 4, fat1)
                ibm_db.bind_param(prep_stmt, 5, carbs1)

                ibm_db.execute(prep_stmt)



            add()
            return render_template('getnut.html',calories=calories,fat=fat,protein=protein,carbs=carbs)
```

```
    return render_template('getnut.html')

@app.route('/up')
def up():
    return render_template('up.html')
# UPLOAD_FOLDER = 'C:\\Users\\bestr\\PycharmProjects\\Nutrition Assistant\\static\\images'
# app.config['UPLOAD_FOLDER']=UPLOAD_FOLDER
@app.route('/uploader', methods = ['GET', 'POST'])
def uploader():
   if request.method == 'POST':
      f = request.files['file']
      f.save(secure_filename(f.filename))
      # f.save(os.path.join(app.config['UPLOAD_FOLDER'],f.filename))
      print(f.filename)
      # n=f.filename
      # file_extns=n.split(".")
      # q=repr(file_extns[0])
      # w=repr(file_extns[-1])
      # a2 = q.strip("\'")
      # print(q)
      a = f.filename
      b = a[:-1]
      c = b[:-1]
      d = c[:-1]
      e = d[:-1]
      print(e)

      return render_template('getnut.html',msg=e,img=a)


@app.route('/display')
def display():
    history=[]
    sql = "SELECT * FROM historyi"
    stmt = ibm_db.exec_immediate(conn, sql)
    dictionary = ibm_db.fetch_both(stmt)
    while dictionary != False:
        # print ("The Name is : ",  dictionary)
        history.append(dictionary)
        dictionary = ibm_db.fetch_both(stmt)
    if history:
        return render_template('display.html',history = history)
```