# NAAN MUDHALVAN

**CLOUD APPLICATION**

**DEVELOPMENT**

**SARANATHAN COLLEGE OF ENGINEERING**

**PROJECT TITLE: PIXEL PERFECTION: TRANSFORMING YOUR PHOTOS WITH OUR CUTTING-EDGE IMAGE EDITING PLATFORM**

By:

Siva Kumar A

Mohamed Ashik A

Mohamed Ibrahim R

Sanjai P

# INTRODUCTION:

## 1.1 Project Overview:

The current problem in the image editing landscape is the lack of a comprehensive and user-friendly platform that allows users to achieve precise and high-quality edits with ease. Existing software often requires complex installations, has limited functionality, or lacks intuitive interfaces, resulting in a frustrating and time-consuming editing process for users. Additionally, there is a need for specialized tools that address specific editing needs, such as background removal, face beauty enhancements, and cartoon effects. Existing solutions may not provide efficient and accurate methods for performing these tasks, leading to subpar results and wasted time.

## 1.2 Purpose:

The purpose of Pixel Perfection is to provide users with an innovative and powerful image editing platform that allows them to transform and enhance their photos with ease and precision. So that people who doesnt know enough knowledge about image editing can also editing their pictures easily without any effort. The platform is designed to serve the following purposes:

1.Image Background Removal: With our application the user can easily able to remove any kind of background from his image and other capabilities. User can easily remove unwanted backgrounds or replace them, resulting in a clear and more visually appealing images.

2.Vehicle Background Removal: With this feature the user can be able to remove any kind of unwanted background from his image. The unwanted backgrounds can be easily removed resulting in a good looking image of a vehicle with enhanced image.

3:Cartoon yourself: This provides the capability for the user to transform his images into cartoon version with different kinds of styles such as Handrawn, manga style, Japanese anime style, pixar style and so on.

4.Beautify yourself: This feature can also be called as Image enhancer which enhances the quality of the image like saturation , color, whitening of the image and so on.

5.Easy UI: The platform is designed with a user-friendly interface that makes it easy for both professionals and beginners to navigate and utilize the editing tools effectively. The intuitive interface ensures a seamless editing experience.

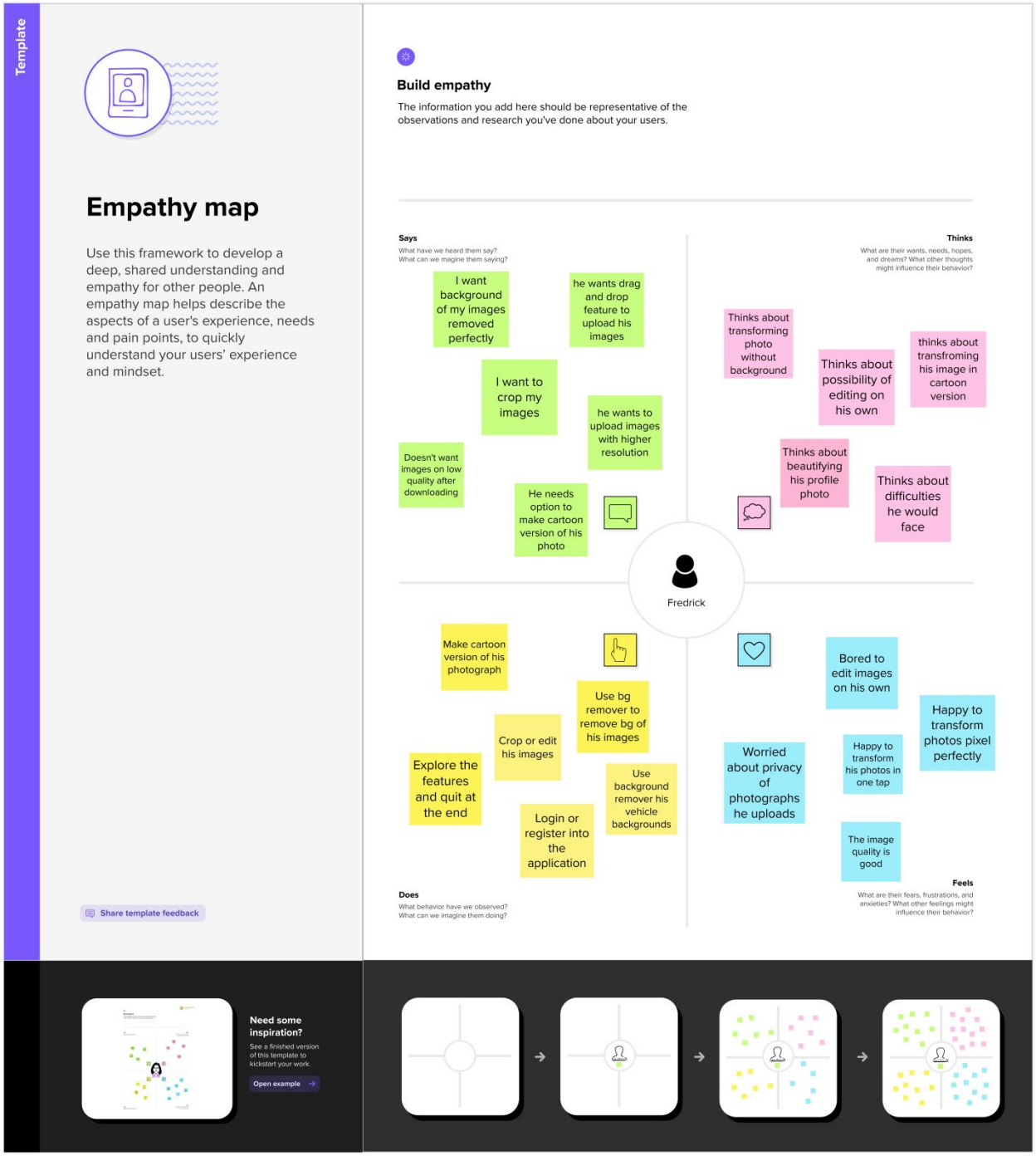
## 2. Ideation & Proposed Solution:

## 2.1 Problem statement Definition:

Pixel Perfection is an innovative image editing platform that allows users to transform their photos with ease and precision. Our cutting-edge software provides a wide range of tools and features that enable users to edit their images to achieve pixel-perfect results. Whether you're a professional photographer, graphic designer, or just someone who loves to take photos, Pixel Perfection is the perfect tool for enhancing your images. With its intuitive user interface and powerful editing tools, you can easily adjust or remove your image backgrounds, car image backgrounds, Cartoon your face & Face beauty, and more to create stunning images that are sure to impress. At Pixel Perfection, we understand the importance of high-quality images in today's digital age, and we're committed to providing our users with the tools they need to achieve pixel-perfect results. Whether you're looking to enhance your personal photos or create professional-quality images for your business or clients, Pixel Perfection has everything you need to get the job done.

## 2.1 Empathy Map Canvas:

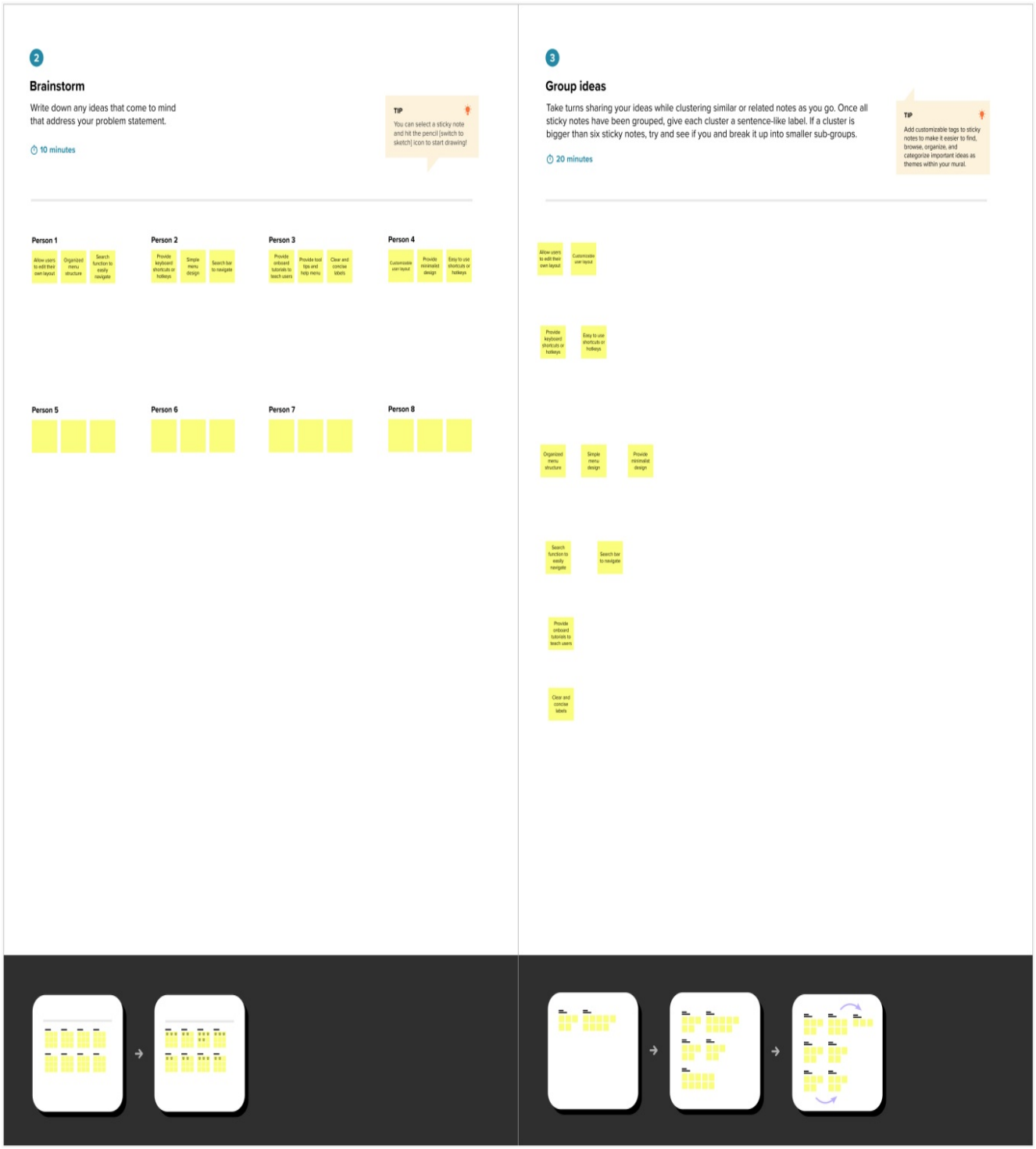
An Empathy map is a simple, easy-to-digest visual that captures knowledge about user's behaviours and attitudes. Using the empathy map canvas the team can able to understand the customer requirements and what all are the functions that the customers wants to get implemented in the project. This also allows developers to consider different types of user perspective along with their goals and challenges. This allows the developers to easily implement the project.




## 2.3 : Ideation & Brainstorming:

Brainstorming provides a free and open environment that encourages everyone within a team to participate in the creative thinking process that leads to problem solving. Prioritizing volume over value, out-of-the box ideas are welcome and

built-upon, and all participants are encouraged to collaborate ,helping eachother develop a rich amount of creative solutions.



4

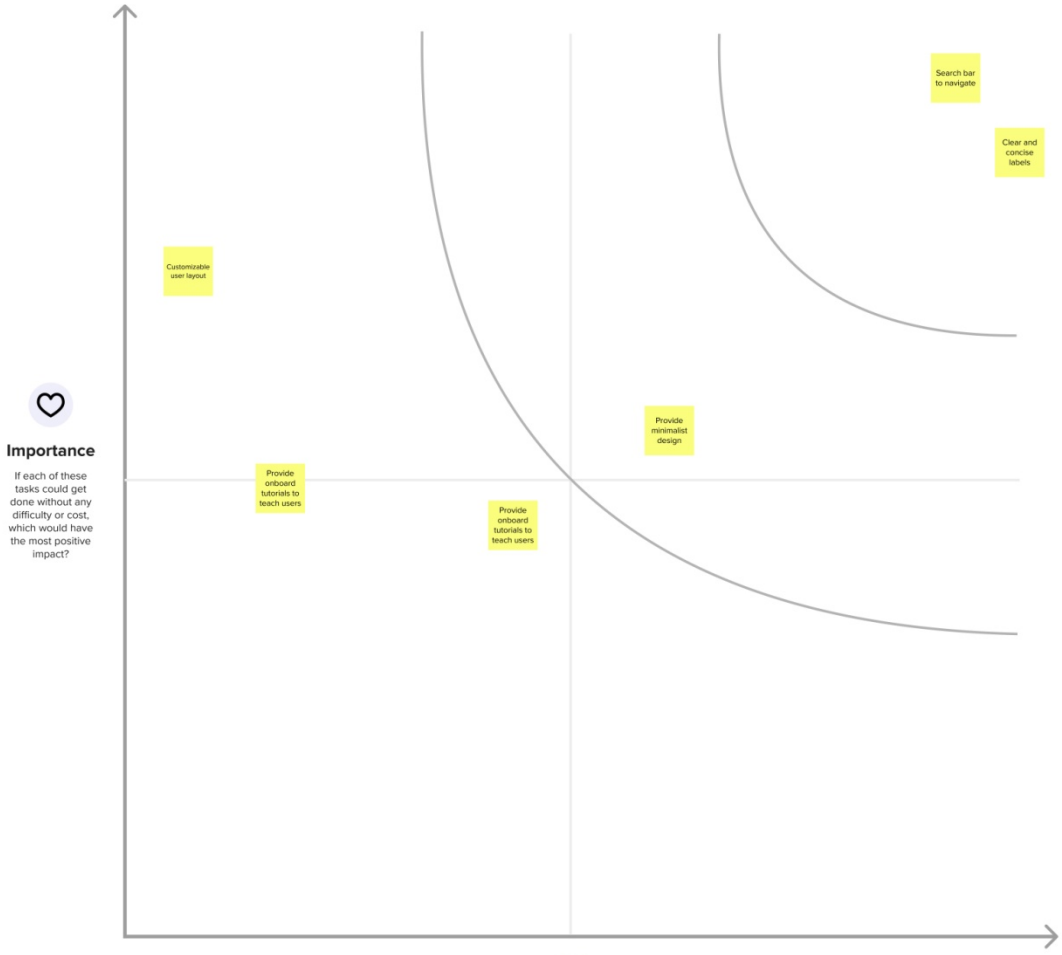
## Prioritize

Your team should all be on the same page about what's important moving forward. Place your ideas on this grid to determine which ideas are important and which are feasible.

20 minutes

**TIP**

Participants can use their cursors to point at where sticky notes should go on the grid. The facilitator can confirm the spot by using the laser pointer holding the **H key** on the keyboard.




## 2.4 : Proposed Solution:

Proposed solution: Pixel perfection is a innovative image editing platform that allows users to transform their images or high resolution photographs with ease and precision. The platform is designed to be intuitive and straight foward, making it accessible to everyone who wants to improve their photographs. It is a web based application which means that user can access it from anywhere with an internet facility. The user can register for an account. After they created their login credentials they can login into the platform and edit the images with different tools, they just upload their pictures and choose an option which edits the picture in a specific way. They can save and view their previous images.

Novelty/Uniqueness: This lies under its user friendly UI and easy to use tools with an organized menu structure that enables the user to achieve their goals of high resolution result without any prior experience on editing images. Since it is a web based platform, it allows users to access it from anywhere with just internet connection and appropriate device.

## 3. : Requirement Analysis:

## 3.1:Functional requirement

The functional requirements of the proposed solution are as follows,

FR-1: User Registration - Registration of user through email or google.

FR-2: Image editing tools- Person background removal, Cartoonish face, beautify face, vehicle background removal

FR-3: Image upload and download- Upload/download images in different formats such as jpeg,png.

FR-4: High resolution images- Provide user with quality images without any disortion or pixel loss.

FR-5: Cloud storage: Retriveal of images from anywhere and anytime.

## 3.2:Non-Functional requirement:

NF-1: Usability- The application should have a user friendly and intuitive interface, with easy to understand and navigate.

NF-2:Security- The applicatin should prioritize the privacy of its user images by implementing or enabling strong ecryption or data protection measures.

NF-3: Reliability-The platform should be designed to handle high volume of user traffic and image uploads without slowing down or crashing the server.

NF-4: Performance- The platform should have minimum latency which ensures effective upload and download of images by user.

NF-5: Availability- The platform should be accessible by the user 24/7, with the high level of uptime. It should have redundant robust infrastructure that can handle unexpected spikes in user traffic without affecting the availability of the platform.

NF-6: Scalability- It should be able to accomodate an increasing number of users and image uploads without negatively affecting the performance.

It should have scalable architecture that can easily add or remove resource as needed to handle changes in user traffic or usage.

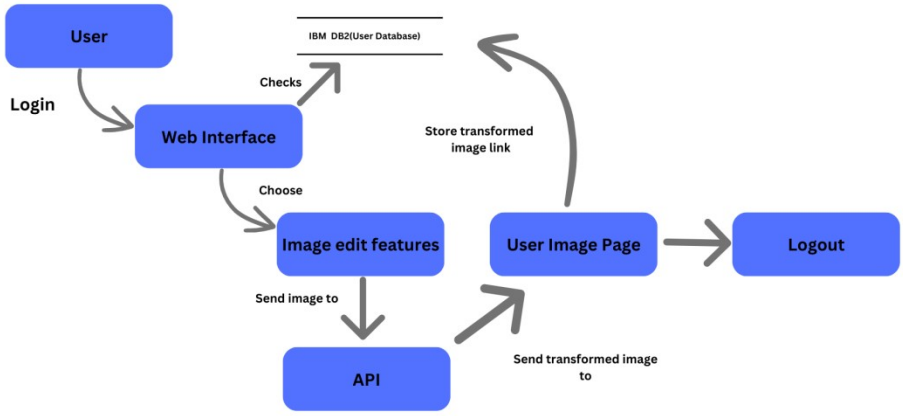
## 4:Project Design:

## 4.1:Data flow diagrams

A data flow diagram is a traditional visual representation of the information flows within a system.

A near and clear DFD can depict the right amount of the system requirement graphically.

It shows how data enters and leaves the system, what changes the information, and where the data is stored.




1.First the user login's into the system.

2.Then choose one of the image editing services available on the web interface.

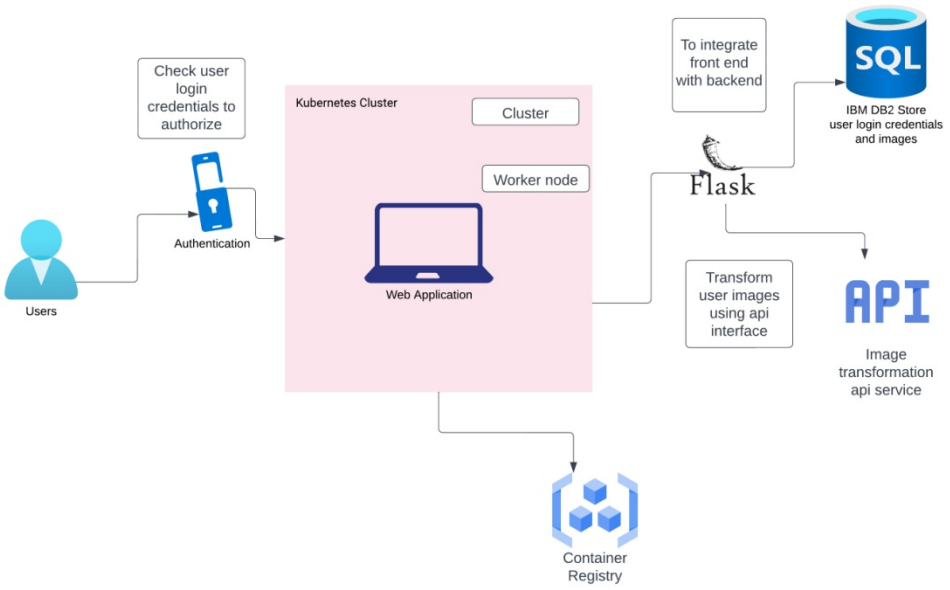
3.Once choosen the service and uploading his image, an api call is made with the user image to transform it however he wants.

4. Then the transformed image is added to the user's db and the image is retrieved to be displayed on the user image page.

5. Then the transformed image link is stored in the db2 database.

6. Then the user logs out of the platform.

# 4.2:Solution and Technical Architecture:



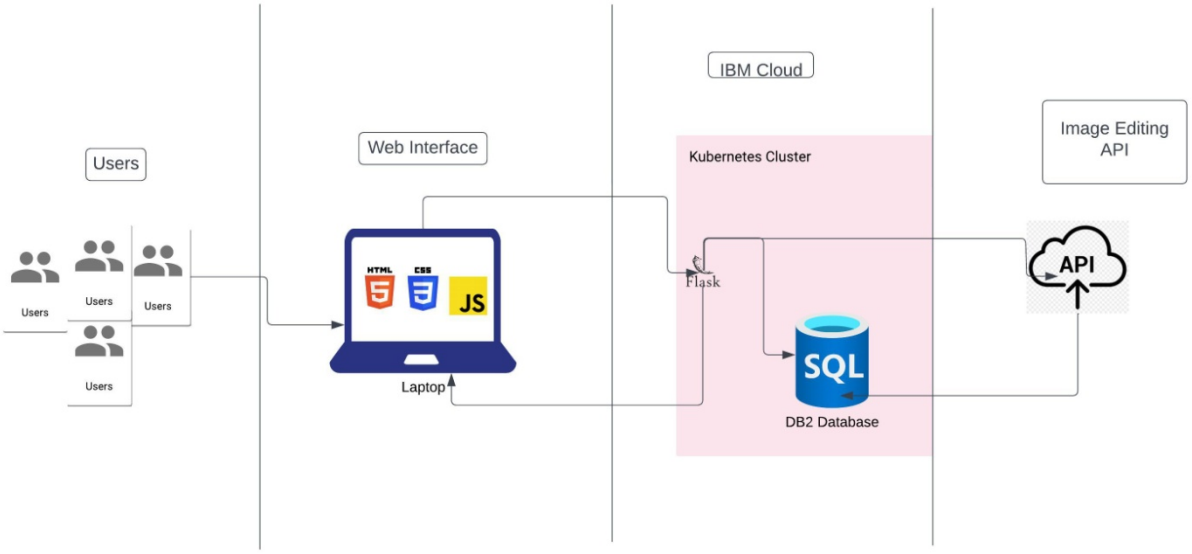



Table-1 : Components & Technologies:

| S.No | Component | Description | Technology |
|---|---|---|---|
| 1. | User Interface | How user interacts with application e.g. Web UI, Mobile App, etc. | HTML, CSS, JavaScript etc. |
| 2. | Image background removal | How can user easily remove only the background of his images | Human background removal api |
| 3. | Vehicle background removal | How user can easily remove background of his vehicle images | Vehicle background removal api |
| 4. | Cartoon your face | Transform user image into cartoon version | Cartoon yourself api |
| 5. | Database | Storing user login details and registration details | IBM DB2 |
| 6. | Cloud Database | Storing user data on cloud to access from anywhere | IBM DB2, IBM COS |
| 7. | Password encryption | Encrypting user passwords using hashing algorithms | Password hashing algorithm |
| 8. | Beautify face | Beautify user face using external api | AI skin Beauty api |
| 9. | Infrastructure (Server / Cloud) | Application deployment on cloud so it can be accessed from anywhere | Docker, Kubernetes , IBM Cloud |

Table-2: Application Characteristics

| S.No | Characteristics | Description | Technology |
|---|---|---|---|
| 1. | Open source Frameworks | Flask is one of the open source used in this application | Micro-Web Framework |
| 2. | Security Implementations | HTTPS encryption to provide security to user data | SHA-256 |
| 3. | Scalability | Web application can be easily scaled up | IBM cloud |
| 4. | Availability | Use of multiple replica of | Kubernets, Docker |

| | | | |
|---|---|---|---|
| | | application hosting to ensure availability | |
| 5. | Performance | Faster response time for requests in the application and minimizing traffic on the web platform. | Vertical scalability and horizontal scalability. |

## 4.3:User Stories:

**User Stories**

| User Type | Functional Requirement (Epic) | User Story Number | User Story / Task | Acceptance criteria | Priority | Team Member |
|---|---|---|---|---|---|---|
| Customer (Mobile user) | Registration | USN-1 | As a user, I can register for the application by entering my email, password, and confirming my password. | I can access my account / dashboard | High | Siva |
| Customer (Web User) | Registration | USN-2 | As a user, I will receive confirmation email once I have registered for the application | I can receive confirmation email & click confirm | High | Siva |
| Customer (Web User) | Uploading images | USN-3 | I want to be able to upload my images in various formats such as png,jpeg. | The application should allow to provide uploading of different formats | High | Aashik |
| Admin (Web User) | Manage Users | USN-4 | As a admin , I want to manage my users in the application itself | The dashboard should display a list of users and their works. | Low | Aashik |
| Customer (Web user) | Downloadin images | USN-5 | As a user, I want to download my images in different formats such as png, jpeg. | The application allow exporting files in png and jpeg format. | High | Ibrahim |
| Customer (Web User) | Drafts | USN-6 | As a user, I want to draft my images so I can use it later to edit | The application allows users to make drafts. | Medium | Ibrahim |
| Customer (Mobile user) | Design layout | USN-7 | As a user, I want a mobile version of the web platform. | I can choose desktop site to use the platform on mobile. | High | Sanjai |
| Customer (Web User) | Batch Processing | USN-8 | As a user, I want to upload and edit multiple images at a time | I can drag and drop multiple files to edit on the platform | High | Sanjai |

## 5.:Coding Solution:

## 5.1:Register

```
@app.route('/register',methods=['GET','POST'])

def register():

   if request.method=='POST':

      username=request.form['myname']

      usermail=request.form['mymail']

      userpassword=request.form['mypass']

      sql='select * from user where email=?'
```

```
    stmt=ibm_db.prepare(conn,sql)
    ibm_db.bind_param(stmt,1,usermail)
    ibm_db.execute(stmt)
    info=ibm_db.fetch_assoc(stmt)
    if info:
        msg2="You have been already registered : Kindly Login"
        return render_template("login.html",msg=msg2)
    else:
        sql='select count(*) from user'
        stmt=ibm_db.prepare(conn,sql)
        ibm_db.execute(stmt)
        length=ibm_db.fetch_assoc(stmt)
        print(length)
        insert_sql='insert into user values (?,?,?,?)'
        prep_stmt=ibm_db.prepare(conn,insert_sql)
        ibm_db.bind_param(prep_stmt,1,username)
        ibm_db.bind_param(prep_stmt,2,usermail)
        ibm_db.bind_param(prep_stmt,3,userpassword)
        ibm_db.bind_param(prep_stmt,4,length ['1']+1)
        ibm_db.execute(prep_stmt)
        msg2="You are successfully registered: Kindly Login"
        return render_template("login.html",msg=msg2)
  return render_template('register.html')
```

Once the user visit our application the first thing he needs to do before surfing into the application is to register himself with email address and password. This helps us to manage and track the users data. The following is the flask code for register:

This code routes the user to register page of our application and manage the activities in that page. Here we defined the methods as get and post which means that it can handle both requests.

Inside the function we check whether the method is post, if the method is post then the function retrieves the username, useremail, and userpassword from the form data using 'request.form['myname'] and so on.

Then the code prepares an sql query to select a user from a table called user based on the provided email. It has a placed holder for the email value.

Then the sql statement is prepared and executed using the IBM DB2 functions ibm_db.prepare() and so on. The result of the sql query is fetched using ibm_db.fetch_assoc(stmt) which retrieves the first row of the result set as an associative array.

If a matching account is found, this means that the user is already present in the database. So an error message is rendered in the login.html. If no matching account is found, the code proceeds to insert the new users information into the user table. First an sql query is executed to count the number of existing users in the table.The count result is fetched using the fetch_assoc() and the number of users is accessed with length['1']. Here we get the length of table.

Then the code prepares an sql insert statement to insert the new user's information into the user table. The sql insert statement is preapred and executed using IBM DB2 function. The respective values are bind to the parameters.

If the user is successfully registered, a success message is displayed by rendering the "login.html" template with the msg2 variable containing the success message.

## 5.2:Login

```
@app.route('/login', methods=['GET', 'POST'])

def login():
```

```
    if request.method == 'POST':
        u_email = request.form['umail']
        u_pass = request.form['upass']
        app.logger.info(f"The email id of the user: {u_email} and password: {u_pass}")
        sql="select * from user where email= ? and password=?"
        stmt=ibm_db.prepare(conn,sql)
        ibm_db.bind_param(stmt, 1,u_email)
        ibm_db.bind_param(stmt,2,u_pass)
        ibm_db.execute(stmt)
        account=ibm_db.fetch_assoc(stmt)
        if account:
            session['Loggedin']=True
            session['USERD']=account['USERD']
            session['NAME']=account['NAME']
            return redirect(url_for('imageai'))
        else:
            msg="Check Email and password you have entered"
            return render_template("login.html",msg=msg)
    return render_template('login.html')
```

This code implements the login functionality in our application. The function will be triggered when the "/login" URL is accessed, and it can handle both the GET and POST request.

Inside the function we check whether the method is post, if the method is post then the function retrieves the useremail and password from the form data using 'request.form['umail'],request.form['upass'].

Then the code prepares an sql query to select a user from a table called user based on the provided email and password. The sql statement includes placeholders for the email and password values.

The sql statement is prepared and executed using IBM DB2 functions. The u_mail and u_pass values are used to bind parameters for the email and password. The result of the sql query is fetched using ibm_db.fetch_assoc(stmt) which retrieves the first row of the result set as an assosciative array.

If matching account is found the user is considered logged in. Session variables are set to store the user's logged in status. Then if a matching user is found the user is redirected to the imageai route using 'redirect(url_for('imageai')).

## 5.3: Background Image Removal

```
@app.route('/bgremoval',methods=['GET','POST'])

def bgremoval():

   if request.method=='POST':

      user_img=request.files['userfile']

      url = "https://human-background-removal.p.rapidapi.com/cutout/portrait/body"

      files = { "image": user_img }

      headers = {

            "X-RapidAPI-Key": "230d665706msh8c981a10569b6aep1c5006jsn77776aeae50e",

            "X-RapidAPI-Host": "human-background-removal.p.rapidapi.com"

      }

      response = requests.post(url, files=files, headers=headers)

      json_data = response.json()

      try:

         image_url = json_data['data']['image_url']

      except KeyError:

         error_msg = json_data.get('error_msg', 'Unknown error occurred.')
```

```
        return render_template("backgroundremoval.html", text=error_msg)
    sql="insert into image_url (USERD,IMAGE_BG) values(?,?)"
    stmt=ibm_db.prepare(conn,sql)
    ibm_db.bind_param(stmt,1,session['USERD'])
    ibm_db.bind_param(stmt,2,image_url)
    ibm_db.execute(stmt)
    return render_template("backgroundremoval.html", image_url=image_url,user_img=user_img)
return render_template("backgroundremoval.html",text="")
```

This code implements the ability to remove background from any image that the user provides. The function will be triggered when the "/bgremoval" URL is accessed, and it can handle both the get and post requests.

The if condition checks if method is post, if it is then the function retrieves the uploaded file from the request using user_img=request.files['userfile'].

Then the code prepares the necessary data to send a POST request to a remote API endpoint. The image file is sent as a part of the request's payload, and the API key and host headers are sent in the headers dictionary.

The POST request is made using the request.post() method passing url,file,and headers as arguments. The response from the API is stored in the response variable.

The response data is the parsed as JSON using response.json and the image url is extracted from the json data. If the image url is successfully obtained, the code performs a database operation to insert the image URL into a table. The sql statement is prepared and executed using IBM_DB2 functions. The session values is used to bind parameter for the user ID, and image_url is used for image url.

Finally the function renders a template called "backgroundremoval.html" and passed the image url,user image and any error message from api as template variables.

## 5.4: Vehicle Background Removal

```
@app.route('/vhremoval',methods=['GET','POST'])
def vhremoval():
    if request.method=='POST':
        user_img=request.files['userfile']
        url = "https://vehicle-background-removal.p.rapidapi.com/cutout/universal/vehicle"
        files={"image": user_img}
        headers = {
                "X-RapidAPI-Key": "230d665706msh8c981a10569b6aep1c5006jsn77776aeae50e",
                "X-RapidAPI-Host": "vehicle-background-removal.p.rapidapi.com"
        }
        response = requests.post(url, files=files, headers=headers)
        json_data = response.json()
        try:
            image_url = json_data['data']['elements'][0]['image_url']
        except KeyError:
            error_msg = json_data.get('error_msg', 'Unknown error occurred.')
            return render_template("vehicleremoval.html",text=error_msg)
        sql="insert into image_url (USERD,VEHICLE_BG) values(?,?)"
        stmt=ibm_db.prepare(conn,sql)
        ibm_db.bind_param(stmt,1,session['USERD'])
        ibm_db.bind_param(stmt,2,image_url)
        ibm_db.execute(stmt)
        return render_template("vehicleremoval.html", image_url=image_url,user_img=user_img)
    return render_template("vehicleremoval.html",text="")
```

The above code implements the ability to remove background from any vehicle image that the user provides. The function will be triggered when the "/vhremoval" URL is accessed, and it can handle both the get and post requests.

The if condition checks if method is post, if it is then the function retrieves the uploaded file from the request using user_img=request.files['userfile'].

Then the code prepares the necessary data to send a POST request to a remote API endpoint. The image file is sent as a part of the request's payload, and the API key and host headers are sent in the headers dictionary.

The POST request is made using the request.post() method passing url,file,and headers as arguments. The response from the API is stored in the response variable.

The response data is the parsed as JSON using response.json and the image url is extracted from the json data.

In this case, the image_url is accessed as json_data['data']['elements'][0]['image_url']. If the image url is successfully obtained, the code performs a database operation to insert the image URL into a table. The sql statement is prepared and executed using IBM_DB2 functions. The session values is used to bind parameter for the user ID, and image_url is used for image url.

Finally the function renders a template called "vehicleremoval.html" and passed the image url,user image and any error message from api as template variables.

## 5.5: Cartoon Yourself

```
@app.route('/cartoonimage',methods=['GET','POST'])

def cartoonimage():

    selected_option = None

    if request.method=='POST':

        selected_option = request.form['my-select']
```

```
    user_img=request.files['userfile']
    url = "https://cartoon-yourself.p.rapidapi.com/facebody/api/portrait-animation/portrait-animation"
    files = { "image": user_img }
    payload = { "type": selected_option }
    headers = {
            "X-RapidAPI-Key": "230d665706msh8c981a10569b6aep1c5006jsn77776aeae50e",
            "X-RapidAPI-Host": "cartoon-yourself.p.rapidapi.com"
    }
    response = requests.post(url,data=payload , files=files, headers=headers)
    json_data = response.json()
    try:
       image_url = json_data['data']['image_url']
    except KeyError:
       error_msg = json_data.get('error_msg', 'Unknown error occurred.')
       return render_template("cartoonimage.html", text=error_msg)
    sql="insert into image_url (USERD,CARTOON_IMG) values(?,?)"
    stmt=ibm_db.prepare(conn,sql)
    ibm_db.bind_param(stmt,1,session['USERD'])
    ibm_db.bind_param(stmt,2,image_url)
    ibm_db.execute(stmt)
    return render_template("cartoonimage.html", image_url=image_url,user_img=user_img)
  return render_template("cartoonimage.html",text="")
```

The above code implements the ability to transform any image that the user provides into a cartoon version of the same image. The function will be triggered when the "/cartoonimage" URL is accessed, and it can handle both the get and post requests.

The if condition checks if method is post, if it is then the function retrieves the uploaded file from the request using user_img=request.files['userfile'].

Then the code prepares the necessary data to send a POST request to a remote API endpoint. The image file is sent as a part of the request's payload, and the API key and host headers are sent in the headers dictionary.

The POST request is made using the request.post() method passing url,file,and headers as arguments. The response from the API is stored in the response variable.

The response data is the parsed as JSON using response.json and the image url is extracted from the json data. The image_url is accessed by json_data['data']['image_url'].

If the image url is successfully obtained, the code performs a database operation to insert the image URL into a table. The sql statement is prepared and executed using IBM_DB2 functions. The session values is used to bind parameter for the user ID, and image_url is used for image url.

Finally the function renders a template called "cartoonimage.html" and passed the image url,user image and any error message from api as template variables.

## 5.5: Beautify Yourself

```
@app.route('/beautyimage',methods=['GET','POST'])

def beautyimage():

   if request.method=='POST':

      user_img=request.files['userfile']

      retouch=request.form['retouch']

      whitening=request.form['whitening']

      url = "https://ai-skin-beauty.p.rapidapi.com/face/editing/retouch-skin"

      files = { "image": user_img}

      payload = {
```

```
            "retouch_degree": retouch,
            "whitening_degree": whitening
}
    headers = {
            "X-RapidAPI-Key": "230d665706msh8c981a10569b6aep1c5006jsn77776aeae50e",
            "X-RapidAPI-Host": "ai-skin-beauty.p.rapidapi.com"
}
    response = requests.post(url, data=payload, files=files, headers=headers)
    json_data=response.json()
    try:
       image_url = json_data['data']['image_url']
    except KeyError:
       error_msg = json_data.get('error_msg', 'Unknown error occurred.')
       return render_template("beautyimage.html", text=error_msg)
    sql="insert into image_url (USERD,SKIN_BEAUTY) values(?,?)"
    stmt=ibm_db.prepare(conn,sql)
    ibm_db.bind_param(stmt,1,session['USERD'])
    ibm_db.bind_param(stmt,2,image_url)
    ibm_db.execute(stmt)
    return render_template("beautyimage.html",image_url=image_url,user_img=user_img)
  return render_template("beautyimage.html",text="")
```

This code implements the capability to enhance the image that the user proviedes.

The above code implements the ability to transform any image that the user provides into a cartoon version of the same image. The function will be triggered when the "/beautyimage" URL is accessed, and it can handle both the get and post requests.

The if condition checks if method is post, if it is then the function retrieves the uploaded file from the request using user_img=request.files['userfile'].

Then the code prepares the necessary data to send a POST request to a remote API endpoint. The image file is sent as a part of the request's payload, and the API key and host headers are sent in the headers dictionary.

The POST request is made using the request.post() method passing url,file,and headers as arguments. The response from the API is stored in the response variable.

The response data is the parsed as JSON using response.json and the image url is extracted from the json data. The image_url is accessed by json_data['data']['image_url'].

If the image url is successfully obtained, the code performs a database operation to insert the image URL into a table. The sql statement is prepared and executed using IBM_DB2 functions. The session values is used to bind parameter for the user ID, and image_url is used for image url.

Finally the function renders a template called "beautyimage.html" and passed the image url,user image and any error message from api as template variables.

# 6. Results:

## 6.1 Performance Metrics

The following are the performance metrics for the pixel perfection application,

1. Speed and responsiveness: Measures the speed at which pixel perfection executes tasks and responds to the user interactions. This metric includes the time it takes for the software to load and process image edits.

2. Editing Features and Versatality: The range and versatility of editing features available in pixel perfection. This metric assesses the variety of tools which help to transform the images.

3. Stability and reliability: The stability and reliability of pixel perfection during editing sessions. This metric evaluates the occurences of crashes, freezes, or other technical issues that may interrupt the editing process.

4. Output Quality: The quality of the final edited images produced by the pixel perfection. This metric assesses factors such as color accuracy,sharpness, resolution and overall visual appeal of the edited images. High quality output of the images ensures that it meets user satisfaction.

## 7.Advantages & Disadvantages:

## Advantages:

1.Wide range of editing features: Pixel perfection offers a diverse set of editing tools and features, allowing users to make various adjustments to their images.

2.Intuitive User Interface: This software boasts an intuitive user interface, making it easy for users of all skill levels to navigate and utilize its features.

3.BackgroundRemoval: It offers a dedicated feature for removing and replacing backgrounds.

4.Face editing feature: This software includes tools specifically dsigned for facial editing and enhancement.

## Disadvantages:

1.Compatibility and integration: It is quite hard to integrate this appication with any other existing applications.

2.Availability and cost: It is essential to consider the pricing structure and evaluate whether it aligns with your budget and editing requirements.

3.Limited and Advanced Features: While our application offers variety of editing tools, it may lack some more advanced features found in professional-grade software.

4. Reliance on internet connection: Since this is a web based application user may need a stable internet connection to access and use the software.

## 8.Conclusion:

In conclusion, Pixel Perfection is an innovative image editing platform that offers users a wide range of tools and features to achieve pixel-perfect results. With its intuitive user interface and precise editing capabilities, it caters to both beginners and experienced users. The software excels in background removal and replacement, as well as facial editing, allowing for creative and professional image enhancements. However, users should be aware of the potential learning curve, limited compatibility, possible costs, and the availability of advanced features. Overall, Pixel Perfection provides a solid editing solution for those seeking to enhance their personal photos or create high-quality images for professional purposes.

## 9.Future Scope:

The pixel perfection application has wide range of future scope and many more functionalities can be added to it.

Advanced AI-Powered Editing: As artificial intelligence and machine learning continue to advance, Pixel Perfection could integrate more advanced AI-powered editing features. This could include intelligent automatic adjustments, content-aware editing, and advanced object recognition for more efficient and accurate editing workflows.

Cloud-Based Collaboration: Collaborative editing is becoming increasingly important in today's digital landscape. Pixel Perfection could expand its capabilities to allow users to collaborate on editing projects in real-time, enabling

multiple users to work together on the same image simultaneously, regardless of their physical locations.

Mobile Integration: With the proliferation of smartphones and mobile photography, integrating Pixel Perfection with mobile platforms could be a significant area of development. A mobile app could provide on-the-go editing capabilities, allowing users to edit and enhance their photos directly from their mobile devices.

# 10. Appendix:

## Source code: app.py

```
from flask import Flask,render_template,request,redirect,url_for,session,json

import ibm_db,requests

app=Flask(__name__)

app.secret_key='fujsbfijsdbfjdi'

conn=ibm_db.connect("database=bludb;hostname=125f9f61-9715-
46f99399c8177b21803b.c1ogj3sd0tgtu0lqde00.databases.appdomain.cloud;port=30426;          uid=rgj24177;
password=TOI51voQjEovGoD5; security=SSL; SSLServercertificate=DigiCertGlobalRootCA.crt","","")

print("Connection successfull")

@app.route('/')

def home():

   return render_template("homepage.html")

@app.route('/register',methods=['GET','POST'])

def register():

   if request.method=='POST':

      username=request.form['myname']

      usermail=request.form['mymail']

      userpassword=request.form['mypass']

      sql='select * from user where email=?'

      stmt=ibm_db.prepare(conn,sql)

      ibm_db.bind_param(stmt,1,usermail)

      ibm_db.execute(stmt)

      info=ibm_db.fetch_assoc(stmt)
```

```
        if info:
            msg2="You have been already registered : Kindly Login"
            return render_template("login.html",msg=msg2)
        else:
            sql='select count(*) from user'
            stmt=ibm_db.prepare(conn,sql)
            ibm_db.execute(stmt)
            length=ibm_db.fetch_assoc(stmt)
            print(length)
            insert_sql='insert into user values (?,?,?,?)'
            prep_stmt=ibm_db.prepare(conn,insert_sql)
            ibm_db.bind_param(prep_stmt,1,username)
            ibm_db.bind_param(prep_stmt,2,usermail)
            ibm_db.bind_param(prep_stmt,3,userpassword)
            ibm_db.bind_param(prep_stmt,4,length ['1']+1)
            ibm_db.execute(prep_stmt)
            msg2="You are successfully registered: Kindly Login"
            return render_template("login.html",msg=msg2)
    return render_template('register.html')
@app.route('/login', methods=['GET', 'POST'])
def login():
    if request.method == 'POST':
        u_email = request.form['umail']
        u_pass = request.form['upass']
        app.logger.info(f"The email id of the user: {u_email} and password: {u_pass}")
        sql="select * from user where email= ? and password=?"
        stmt=ibm_db.prepare(conn,sql)
        ibm_db.bind_param(stmt, 1,u_email)
        ibm_db.bind_param(stmt,2,u_pass)
        ibm_db.execute(stmt)
        account=ibm_db.fetch_assoc(stmt)
        if account:
```

```
            session['Loggedin']=True
            session['USERD']=account['USERD']
            session['NAME']=account['NAME']
            return redirect(url_for('imageai'))
        else:
            msg="Check Email and password you have entered"
            return render_template("login.html",msg=msg)
    return render_template('login.html')
@app.route('/imageai')
def imageai():
    return render_template('imageai.html')
@app.route('/bgremoval',methods=['GET','POST'])
def bgremoval():
    if request.method=='POST':
        user_img=request.files['userfile']
        url = "https://human-background-removal.p.rapidapi.com/cutout/portrait/body"
        files = { "image": user_img }
        headers = {
                "X-RapidAPI-Key": "230d665706msh8c981a10569b6aep1c5006jsn77776aeae50e",
                "X-RapidAPI-Host": "human-background-removal.p.rapidapi.com"
        }
        response = requests.post(url, files=files, headers=headers)
        json_data = response.json()
        try:
            image_url = json_data['data']['image_url']
        except KeyError:
            error_msg = json_data.get('error_msg', 'Unknown error occurred.')
            return render_template("backgroundremoval.html", text=error_msg)
        sql="insert into image_url (USERD,IMAGE_BG) values(?,?)"
        stmt=ibm_db.prepare(conn,sql)
        ibm_db.bind_param(stmt,1,session['USERD'])
        ibm_db.bind_param(stmt,2,image_url)
```

```
        ibm_db.execute(stmt)
        return render_template("backgroundremoval.html", image_url=image_url,user_img=user_img)
    return render_template("backgroundremoval.html",text="")
@app.route('/vhremoval',methods=['GET','POST'])
def vhremoval():
    if request.method=='POST':
        user_img=request.files['userfile']
        url = "https://vehicle-background-removal.p.rapidapi.com/cutout/universal/vehicle"
        files={"image": user_img}
        headers = {
                "X-RapidAPI-Key": "230d665706msh8c981a10569b6aep1c5006jsn77776aeae50e",
                "X-RapidAPI-Host": "vehicle-background-removal.p.rapidapi.com"
        }
        response = requests.post(url, files=files, headers=headers)
        json_data = response.json()
        try:
            image_url = json_data['data']['elements'][0]['image_url']
        except KeyError:
            error_msg = json_data.get('error_msg', 'Unknown error occurred.')
            return render_template("vehicleremoval.html",text=error_msg)
        sql="insert into image_url (USERD,VEHICLE_BG) values(?,?)"
        stmt=ibm_db.prepare(conn,sql)
        ibm_db.bind_param(stmt,1,session['USERD'])
        ibm_db.bind_param(stmt,2,image_url)
        ibm_db.execute(stmt)
        return render_template("vehicleremoval.html", image_url=image_url,user_img=user_img)
    return render_template("vehicleremoval.html",text="")
@app.route('/cartoonimage',methods=['GET','POST'])
def cartoonimage():
    selected_option = None
    if request.method=='POST':
        selected_option = request.form['my-select']
```

```
        user_img=request.files['userfile']
        url = "https://cartoon-yourself.p.rapidapi.com/facebody/api/portrait-animation/portrait-animation"
        files = { "image": user_img }
        payload = { "type": selected_option }
        headers = {
                "X-RapidAPI-Key": "230d665706msh8c981a10569b6aep1c5006jsn77776aeae50e",
                "X-RapidAPI-Host": "cartoon-yourself.p.rapidapi.com"
        }
        response = requests.post(url,data=payload , files=files, headers=headers)
        json_data = response.json()
        try:
           image_url = json_data['data']['image_url']
        except KeyError:
           error_msg = json_data.get('error_msg', 'Unknown error occurred.')
           return render_template("cartoonimage.html", text=error_msg)
        sql="insert into image_url (USERD,CARTOON_IMG) values(?,?)"
        stmt=ibm_db.prepare(conn,sql)
        ibm_db.bind_param(stmt,1,session['USERD'])
        ibm_db.bind_param(stmt,2,image_url)
        ibm_db.execute(stmt)
        return render_template("cartoonimage.html", image_url=image_url,user_img=user_img)
    return render_template("cartoonimage.html",text="")
@app.route('/beautyimage',methods=['GET','POST'])
def beautyimage():
    if request.method=='POST':
        user_img=request.files['userfile']
        retouch=request.form['retouch']
        whitening=request.form['whitening']
        url = "https://ai-skin-beauty.p.rapidapi.com/face/editing/retouch-skin"
        files = { "image": user_img}
        payload = {
             "retouch_degree": retouch,
```

```
            "whitening_degree": whitening
}
        headers = {
            "X-RapidAPI-Key": "230d665706msh8c981a10569b6aep1c5006jsn77776aeae50e",
            "X-RapidAPI-Host": "ai-skin-beauty.p.rapidapi.com"
}
        response = requests.post(url, data=payload, files=files, headers=headers)
        json_data=response.json()
        try:
            image_url = json_data['data']['image_url']
        except KeyError:
            error_msg = json_data.get('error_msg', 'Unknown error occurred.')
            return render_template("beautyimage.html", text=error_msg)
        sql="insert into image_url (USERD,SKIN_BEAUTY) values(?,?)"
        stmt=ibm_db.prepare(conn,sql)
        ibm_db.bind_param(stmt,1,session['USERD'])
        ibm_db.bind_param(stmt,2,image_url)
        ibm_db.execute(stmt)
        return render_template("beautyimage.html",image_url=image_url,user_img=user_img)
    return render_template("beautyimage.html",text="")
@app.route('/beautyimages')
def beautyimages():
    sql = "SELECT (SKIN_BEAUTY) FROM IMAGE_URL WHERE USERD=" +str(session['USERD'])
    stmt = ibm_db.prepare(conn, sql)
    ibm_db.execute(stmt)
    row=[]
    while True:
        data=ibm_db.fetch_assoc(stmt)
        if not data:
            break
        else:
            row.append(data)
```

```
            app.logger.info(row)
    return render_template("beautyimages.html",rows=row)
@app.route('/bgremoveimages')
def bgremoveimages():
    sql = "SELECT (IMAGE_BG) FROM IMAGE_URL WHERE USERD=" +str(session['USERD'])
    stmt = ibm_db.prepare(conn, sql)
    ibm_db.execute(stmt)
    row=[]
    while True:
        data=ibm_db.fetch_assoc(stmt)
        if not data:
            break
        else:
            row.append(data)
            app.logger.info(row)
    return render_template("bgremoveimages.html",rows=row)
@app.route('/cartoonimages')
def cartoonimages():
    sql = "SELECT CARTOON_IMG FROM IMAGE_URL WHERE USERD=" +str(session['USERD'])
    stmt = ibm_db.prepare(conn, sql)
    ibm_db.execute(stmt)
    row=[]
    while True:
        data=ibm_db.fetch_assoc(stmt)
        if not data:
            break
        else:
            row.append(data)
            app.logger.info(row)
    return render_template("cartoonimages.html",rows=row)
@app.route('/vehiclebgimages')
def vehiclebgimages():
```

```
    sql = "SELECT (VEHICLE_BG) FROM IMAGE_URL WHERE USERD=" +str(session['USERD'])
    stmt = ibm_db.prepare(conn, sql)
    ibm_db.execute(stmt)
    row=[]
    while True:
        data=ibm_db.fetch_assoc(stmt)
        if not data:
            break
        else:
            row.append(data)
            app.logger.info(row)
    return render_template("vehiclebgimages.html",rows=row)
@app.route('/myimages')
def myimages():
    return render_template('myimages.html')
@app.route('/about')
def about():
    return render_template('about.html')
@app.route('/logout')
def logout():
    session.pop('Loggedin',None)
    session.pop('USERD',None)
    return render_template('login.html')
if __name__=='__main__':
    app.run(debug=True,host='0.0.0.0')
```

**Source code for the html and css pages can be found in the github link.**

**Application url : http://159.122.178.3:32265/**

**Github: https://github.com/naanmudhalvan-SI/IBM--17828-1682661341**

**Project Demo Link: https://youtu.be/N__3Whxfie8**