COMP4910 Senior Design Project 1, Fall 2024
Advisor: Assoc. Prof. Dr. Ömer ÇETİN

GLOWG: Personalized Skin Care Powered by AI
Requirements Specifications Document

12.01.2025
Revision 2.0

By:
Ceren Sude Yetim, 21070001045
İrem Demir, 20070001029
Gizem Tanış, 20070001047
Ece Topuz, 21070001057

Revision History

Revision	Date	Explanation
1.0	10.11.2024	Initial requirements
1.1	09.12.2024	Made some adjustments on 2.1, 2.2, 2.3, 2.6, 2.7.
1.2	24.12.2024	GlowG introduction title edited. 2.2 The registration form takes email, password and password verification information. Added email uniqueness check. 2.6 User preferences made clear, made additions to the user interface “For each step besides serum user can choose 1, for serum step user can choose from 1 to 3 products among 5 recommended products.“ Added UML diagrams to functions 2.2, 2.3, 2.4, 2.5, 2.6 and 2.7.
1.3	30.12.2024	2.3 Test questions updated. 3.1 Algorithms changed, instead of decision tree decided on random forest, and added clustering algorithms.
2.0	12.01.2025	Since the project’s purpose has changed, revised the whole document accordingly.

Table of Contents

1. Introduction to GlowGenie

This section presents the problem GlowGenie aims to solve, the solutions it offers through AI-driven personalization, an overview of relevant literature in AI-powered skincare recommendation systems, challenges addressed by the application, and GlowGenie's personalized approach to revolutionizing skincare.

Problem Definition:

Skin care is an area of great importance for both the physical health and aesthetic concerns of individuals. However, correctly identifying skin type and choosing products that are suitable for this information can be a complex and confusing process for most individuals. Difficulties in determining skin type often lead to the use of the wrong products, which can have negative effects on the skin such as irritation, dryness, oiliness, and acne. Wrong product choices not only endanger the health of the skin, but also lead to financial losses and users losing confidence in skin care.

The fast pace of modern life causes individuals to not have enough time to access accurate information, while the variety of products on the market makes it even more difficult for users to make choices. Moreover, users who do not have sufficient information about the content of many products and the effects of these contents on skin types are often forced to make choices based on advertisements and guidance. This can lead users to make unconscious decisions and use products that will negatively affect skin health.

Problem Solution:

With the increasing demand for skin care, GlowGenie is designed as an artificial intelligence-supported skin care platform that aims to facilitate individuals' access to accurate and reliable solutions. This platform has an artificial intelligence model that allows users to accurately determine their skin type. Although skin type information is of critical importance in choosing the right products, many people do not know their skin type or evaluate it incorrectly. To solve this problem, GlowGenie offers an intelligent system that analyzes users' skin types with simple questions. Thus, users can make a conscious and safe start to their skin care journey.

Another basic function of GlowGenie is to offer a model that evaluates users' current or potential product content in terms of suitability for their skin type. This model analyzes the content of the products and determines whether they are compatible with their skin type. At the same time, when the user requests a recommended product from the application, the application can also respond to this request. Products in the desired categories are recommended to the user in accordance with the skin type.

At this point, the need for technological tools that offer reliable, fast and personalized solutions is increasing. GlowGenie, The developed artificial intelligence-supported system facilitates the processes of determining skin type, evaluating product compatibility according to the person's skin type and providing recommendations for personal needs. GlowGenie offers a solution that users can choose with confidence, making skin care a more accessible, more effective and more user-friendly experience. By providing users with a reliable source of information and recommendations, it enables them to make more conscious and effective choices in their skin care decisions.

Literature Review:

AI-based recommendation systems have gained popularity due to their ability to provide personalized suggestions with high accuracy. For example, Kumar et al. [1] developed a college recommendation system using a content-based approach that matches user profiles with college profiles. Similarly, in skincare, machine learning models, like the CNN developed by Saidah et al. [2], are used to classify skin types and recommend products accordingly.

The increasing demand for personalized skincare recommendations has led to the development of content-based systems. Vinutha et al. [3] proposed a system that not only considers the chemical composition of products but also adjusts based on users' skin types and preferences. This system, like other AI models, focuses on user-specific features to provide more tailored suggestions, similar to how GlowGenie uses machine learning for skincare recommendations.

Jadhav et al. [4] also explore the potential of machine learning for building a personalized skincare recommendation system. Their system analyzes user data such as skin type, concerns, and product preferences to provide tailored skincare product suggestions. This aligns with the trend of leveraging user data to create highly personalized experiences in skincare.

Additionally, the Intelligent Facial Skin Care Recommendation WebApp proposed by Lokesh et al. [5] uses Convolutional Neural Networks (CNN) to classify skin types based on uploaded images. The system then recommends tailored skincare products based on attributes such as dryness, oiliness, and sensitivity. This concept of integrating both

dermatological expertise and user preferences for building effective skincare product recommendations further enhances the accuracy and personalization of the system.

These developments suggest that content-based filtering methods are increasingly being applied across industries, including skincare, to create personalized experiences. As seen with the systems by Jadhav et al. [4] and Lokesh et al. [5], such approaches are likely to influence recommendation systems in other fields as well.

Challenges Addressed by GlowGenie:

The GlowGenie app overcomes several common challenges in creating effective skincare that vary from person to person.

With countless skincare products on the market containing a variety of active ingredients, users often struggle to determine which products and ingredients are best suited to their skin type. Factors such as skin type, sensitivities, allergies, and specific skincare goals add to the complexity of skincare. GlowGenie aims to overcome these challenges by making users an active part of the process, facilitating the process by encouraging users to share basic information about their skin type (oily, dry, combination, normal), known allergies, and product preferences. This information is processed by the app's ML models, allowing for skin-type-specific analyses for each user, creating a solid foundation for personalized skincare.

Another major challenge is the difficulty users face in assessing the compatibility of product ingredients with their skin type. Based on ingredient analysis, GlowGenie identifies products that are compatible with their skin type and makes recommendations that minimize possible side effects. The app also offers customized recommendations based on users' preferences in specific product categories (cleansers, moisturizers, sunscreens, toners, serums).

Personalized Approach:

GlowGenie offers a personalized approach that shapes skin care processes according to the needs of individuals. Each individual's skin structure and skin care goals may differ. For this reason, GlowGenie aims to be a platform that deeply understands users' needs and offers them special solutions. GlowGenie's personalized approach is based on machine learning models developed to learn users' skin type, allergies, and product preferences. As a first step, a machine learning model is activated that helps users correctly determine their skin type. This model analyzes users' skin type correctly and enables them to choose products that suit their needs.

In the subsequent stage, GlowGenie assesses the ingredients of the products that users are currently using or considering. By analyzing the composition of these products, the system provides detailed information regarding their ingredients and offers feedback on the suitability of the products for the user's specific skin type. This evaluation ensures that users make informed decisions about their skincare choices.

GlowGenie's personalized recommendation system also supports users in creating customized skin care routines according to their needs. For example, if the user is looking for a cleanser, GlowGenie only recommends cleansers suitable for their skin type. If there is a specific allergy, these factors are also taken into consideration and possible side effects are minimized.

Conclusion:

Although skin care is an important area for the health and aesthetic appearance of individuals, choosing the right product is a complex and time-consuming process. GlowGenie is an artificial intelligence-powered platform that offers solutions to these challenges, allowing users to accurately analyze their skin type and choose the most suitable products for their skin. With its personalized approach and smart recommendation system, GlowGenie aims to provide an effective and accessible skin care experience that users can safely choose.

2. Functional Requirements

2.1. Main User Interface and Functions

Main User Interface (Main Menu) and Functions are as follows:

User Account Operations

I want to log in or register for an account.
I want to update my profile information (skin type, skin color, allergens,etc.).
I want to take or retake the skin type test.
I want to log out.

Product Operations

I want to check if specific products are suitable for my skin type.
I want to view product details, including ingredients and get feedback on the product's suitability.
I want to generate products that are suitable for my skin's characteristics, with the product categories I have chosen.
I want to view all the products and filter them by their categories.

2.2. Enter Registration Information

Users must register to use the app. Registration includes providing personal information and optional preferences.

User Registration Form:
The user will fill out a form containing the following information. Fields marked with a '*' are mandatory.

- Email*
- Password*
- Confirm Password*
- Skin Type (select from oily, dry, normal, combination, or "I don't know")*
- Skin Color (Light skin, Medium Skin, Dark Skin)*
- Allergens (optional text box)
- Submit & Cancel Buttons

Submit:

- Check if the email is unique.
- If unique, enter into the email verification process.
- If the email is already registered, it displays an error message: "*This email address is already registered.*"
- Checks if all the mandatory areas are filled, if not displays an error message: "*Please fill the mandatory areas.*"
- Checks if allergen input is in the desired form if not displays an error message: "*Please enter the allergens in the desired form. Such as Paraben, Alcohol, etc…*"
- Checks if the password includes at least one uppercase letter, one lowercase letter, one number, and one special character; otherwise, displays an error message: "*Password must contain at least one uppercase letter, one lowercase letter, one number, and one special character.*"

Cancel: Clears all fields and redirects to the main page.

Email Verification Process:

- The system sends a 4-digit confirmation code to the user's email. Displays a message "*We have sent you an email so we can verify your account. Could you please check your email address?"* The user must enter the code to proceed, and if the entered code does not match the sent code, an error message is displayed: *"Confirmation code is incorrect."* Users also have the option to request a new code if necessary. However, to prevent misuse: If a user requests a new code more than twice within 10 minutes, the system displays the message: "*Please wait 10 minutes before requesting a new code.*" If the user enters the wrong code three times,

the system displays the message: "*You have entered the wrong code three times. Please request a new code.*" Once the correct code is submitted, a countdown of 7 seconds starts and the system displays a message "*Your email address has been verified, you can log in."* redirects the user to the home page.

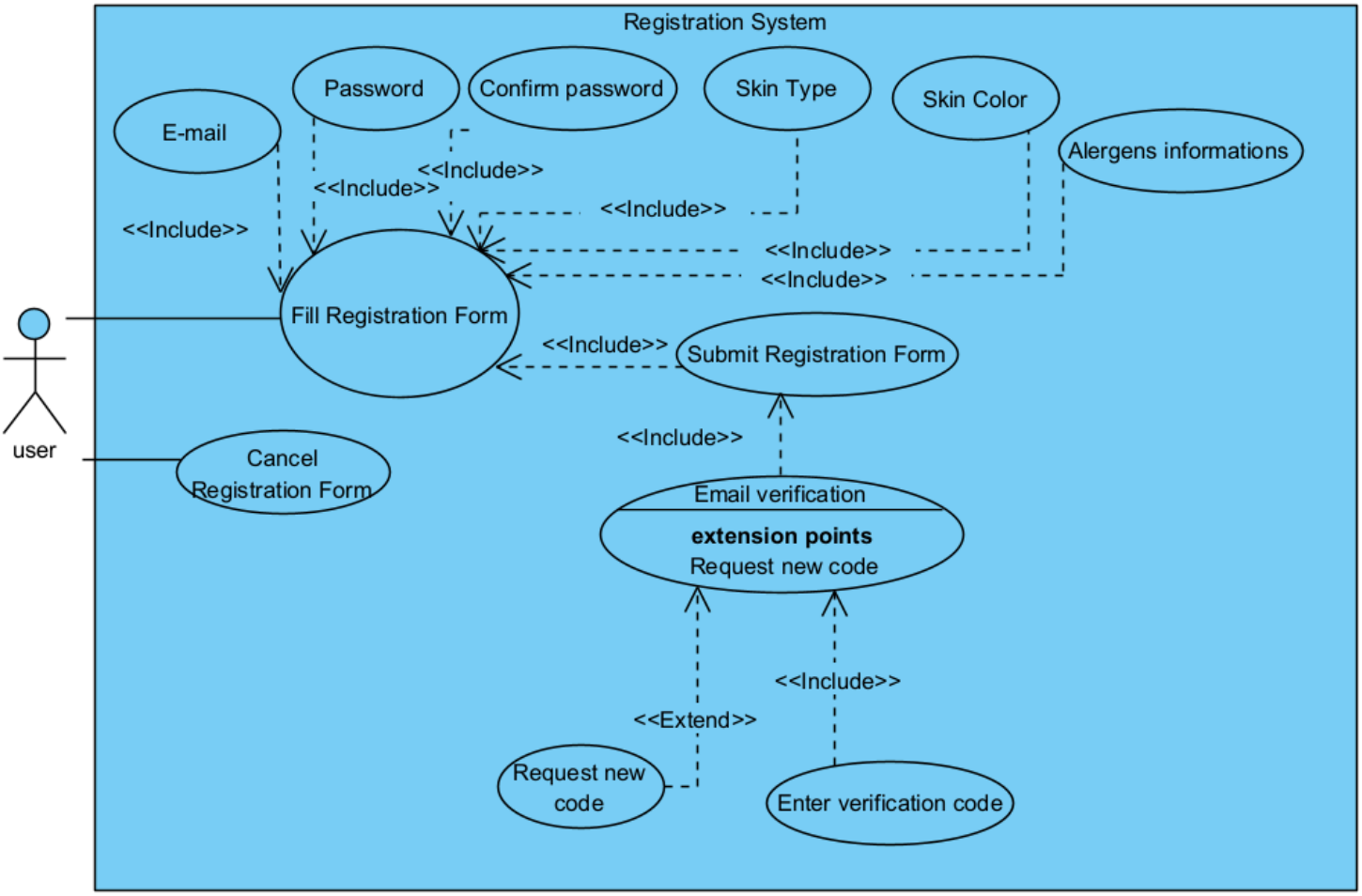

Figure 1 Registration Use Case Diagram

The provided use case diagram in *Figure 1* offers a detailed overview of the user registration process, highlighting essential actions, dependencies, and optional extensions. It effectively illustrates the interactions between the user and the registration system, focusing on core registration activities, validation processes, and optional actions.

Key Actors and Components:

- **User:** The primary actor who interacts with the system by filling out and submitting the registration form or canceling the process.
- **System Use Cases:**
  - **Fill Registration Form:** The user initiates the registration by providing necessary information, such as a unique email, password, confirmation password, skin type, skin color, and allergen information. Each of these fields is included in the form through <> relationships.
  - **Submit Registration Form:** After completing the form, the user submits it. This action triggers the system to validate the provided data, ensuring all required fields are filled, the email is unique, and allergens are correctly specified.
  - **Email Verification:** Upon submission, the system sends a verification code to the user's email. The user must enter this code to verify their email address. This process is essential for account activation.
  - **Enter Verification Code:** This mandatory step requires the user to input the received verification code. The system validates the code to complete the registration.

- **Request New Code (Extension Point):** If the user does not receive the verification code or loses it, they can request a new code. This optional action is represented as an <> relationship in the diagram, providing flexibility during the email verification process.
- **Cancel Registration Form:** The user can cancel the registration process at any stage, which redirects them back to the main page, ensuring user control over the process.

Use Case Diagram Relationships:

- **Include Relationships:** These represent mandatory steps within the registration process, such as entering an email, password, and other necessary information when filling out the form, as well as validating the verification code during the email verification step.
- **Extend Relationships:** These capture optional or alternative flows, such as the ability to request a new verification code if the original one is not received.

Summary:

This use case diagram provides a comprehensive view of the registration system, covering form completion, submission, validation, and email verification. The inclusion of optional extension points, like requesting a new verification code and canceling the registration, enhances the user experience by offering flexibility and control. The diagram effectively demonstrates system behavior and user interaction, ensuring a structured and user-friendly registration process.

Figure 2 illustrates the user registration process, divided into two key sections: User and Registration System. The User section represents actions and decisions made by the user, while the Registration System section outlines the processes and responses managed by the system.

Initial Display and Registration Form:

- The registration page is displayed.
- The user fills out the registration form and submits it.

Email Uniqueness Check:

- The system checks if the provided email address is unique.
- If unique, a verification email with a 4-digit code is sent to the user.
- If already registered, the system displays an error: *"This email address is already registered."*

Verification Code Entry:

- The user enters the 4-digit code received in their email.
- The system verifies if the entered code matches the one sent.
- If correct, the registration is successful, and the user is redirected to the home page.
- If incorrect, an error message is shown: *"Confirmation code is not correct."*

Handling Incorrect Code Attempts:

- The user can re-enter the code or request a new one by clicking the *"resend the code"* button.

Restriction on Code Requests:

- Users can request a new code a maximum of two times within a 10-minute window.
- If exceeded, the system displays: *"Please wait 10 minutes to request a new code."*

Multiple Incorrect Attempts Handling:

- If the wrong code is entered three times, the system prompts: *"You've entered the wrong code 3 times, please request a new one."*
- This prevents repeated incorrect attempts without resolving the issue.

Key Flow Points:

- Success Path:
  - Users enter the correct code, complete registration, and are redirected to the home page.
- Error Feedback:
  - The system provides clear messages for duplicate email detection, incorrect code entries, and limits on code requests or failed attempts.

User Guidance:

- At every decision point, users are given options to correct their actions or proceed, ensuring a smooth and user-friendly registration process.

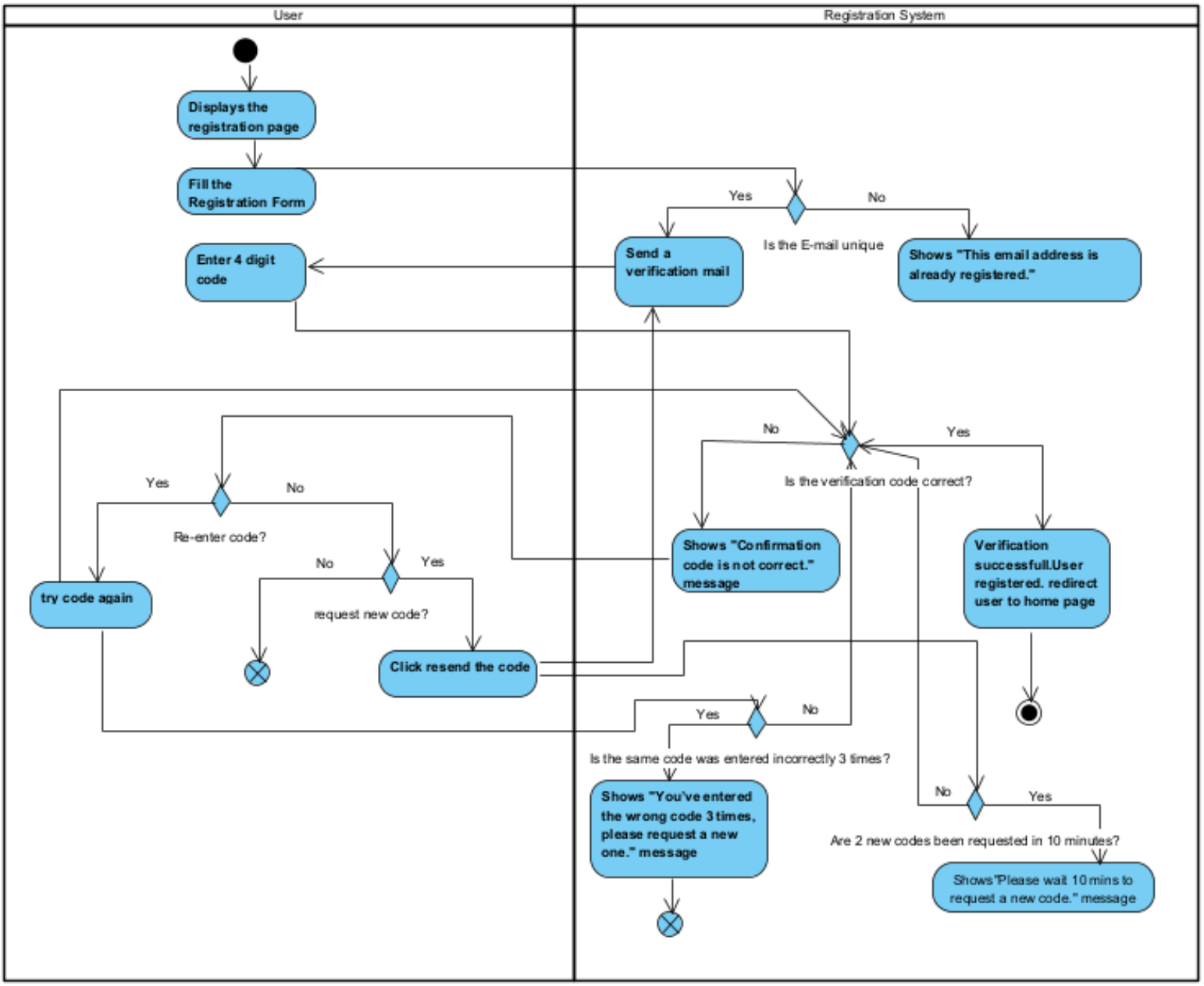

Figure 2 Registration Activity Diagram

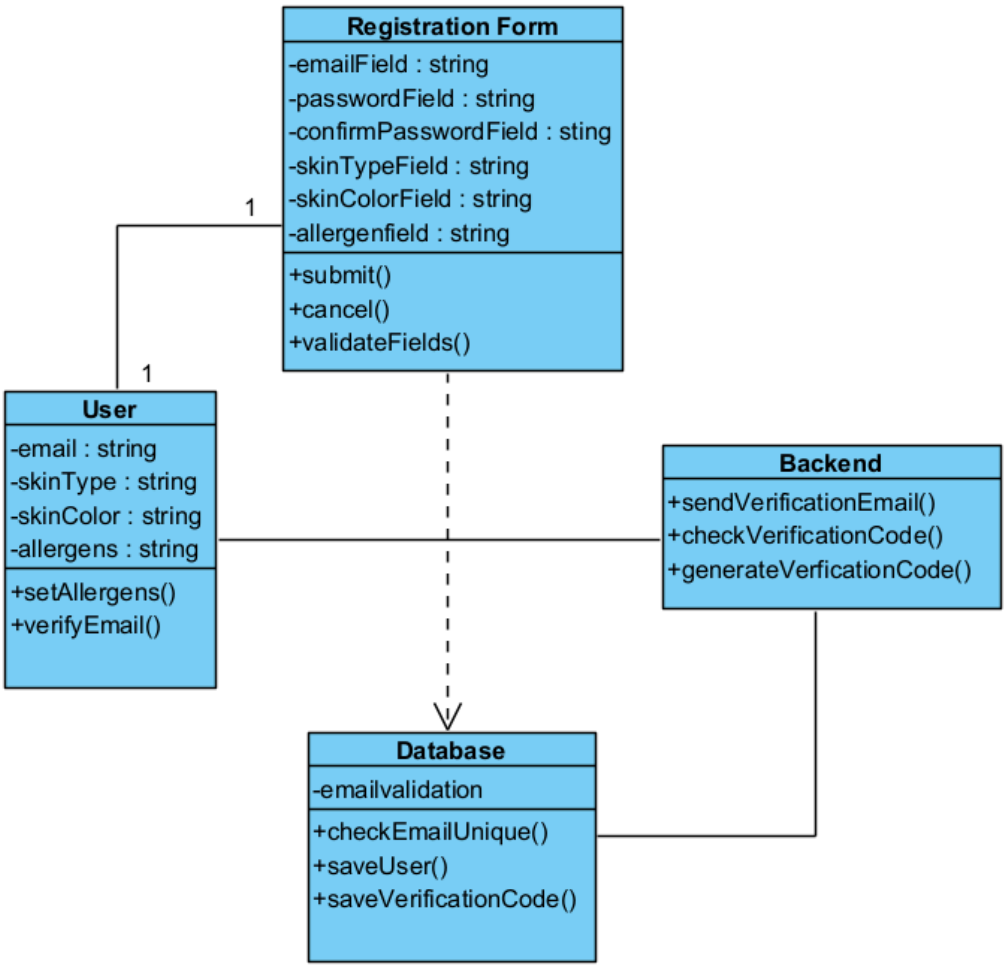

Figure 3 Registration Class Diagram

This *Figure 3* outlines the relationships between various classes and their methods, attributes, and functionalities, providing a detailed view of the system architecture.

Registration Form Class:

- **Purpose:** Serves as the user interface for inputting registration details.
- **Attributes:**
  - emailField, passwordField, confirmPasswordField for credentials.
  - skinTypeField, skinColorField, allergenField for personal details.
- **Methods:**
  - submit(): Sends data for processing.
  - cancel(): Aborts the registration process.
  - validateFields(): Ensures input meets required criteria (e.g., email format, password match).

User Class:

- **Purpose:** Represents the individual registering on the platform.
- **Attributes:**
  - email, skinType, skinColor, allergens to store user information.
- **Methods:**
  - setAllergens(): Allows users to specify allergens.
  - verifyEmail(): Checks the confirmation code sent to the user's email.

Backend Class:

- **Purpose:** Manages behind-the-scenes operations and logic.
- **Methods:**
    - sendVerificationEmail(): Sends confirmation emails.
    - checkVerificationCode(): Validates the entered code.
    - generateVerificationCode(): Creates unique codes for email confirmation.

Database Class:

- **Purpose:** Handles data storage and retrieval.
- **Methods:**
    - emailValidation(): Verifies email format and existence.
    - checkEmailUnique(): Ensures the email is not already registered.
    - saveUser(): Saves user details.
    - saveVerificationCode(): Stores generated codes for verification.

Class Relationships:

- **User and Registration Form:** One-to-one relationship; the user inputs data through the form.
- **Registration Form and Backend:** The form relies on the backend for data processing, sending emails, and code validation.
- **Backend and Database:** The backend interacts with the database to validate emails, store user data, and save verification codes.

2.3.1. Log In

The login form allows users to access their accounts with their email and password.

Login Form:

- Email*
- Password*
- Login, Forgot Password

Login:

- If the entered email is not registered/doesn't match or the entered password is incorrect displays an error message: "*Login information is not correct. Try again.*" If no errors occur, the user logs in.

Forgot Password:

- The system sends a 4-digit confirmation code to the user's email that is entered in the login form if mail is not entered an error message is displayed: "Please enter your email address first.*"*. If the entered email is not registered, an error message is displayed: *"Email is not registered."* The user must enter the code to proceed, and if the entered code does not match the sent code, an error message is displayed: *"Confirmation code is incorrect."* Users also have the option to request a new code if necessary. However, to prevent misuse: If a user requests a new code more than twice within 10 minutes, the system displays the message: "*Please wait 10 minutes before requesting a new code.*" If the user enters the wrong code three times, the system displays the message: "*You have entered the wrong code three times. Please request a new code.*"Once the correct code is submitted, the user is allowed to reset their password. If the new password and its confirmation do not match, an error message is shown: *"Passwords don't match."* When the new password is successfully saved, a confirmation message is displayed: *"Your new password is saved."* Redirects the user to the home page.
- Checks if the password includes at least one uppercase letter, one lowercase letter, one number, and one special character; otherwise, displays an error message: "*Password must contain at least one uppercase letter, one lowercase letter, one number, and one special character.*"

Log Out:

2.4. Users can log out using the "LogOut" button.
2.5. Upon clicking the "LogOut" button, the system terminates the user's session and redirects them to the main page.

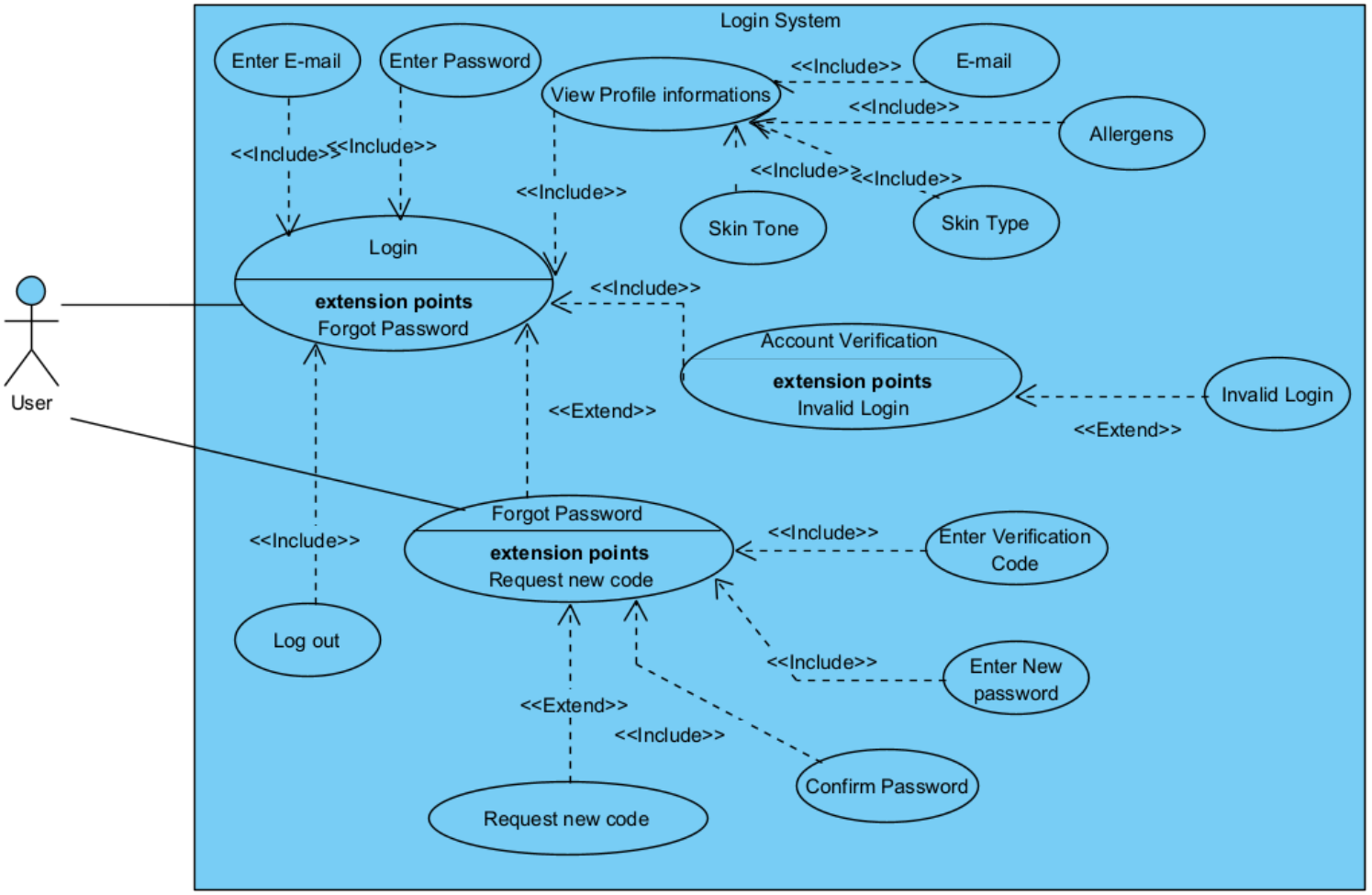

Figure 4 LogIn Use Case Diagram

The *Figure 4* diagram provides an overview of the interactions between the user and the system, focusing on functionalities like logging in, resetting passwords, viewing profile details, and logging out.

Primary Actor:

- The **User** is the main actor who performs all key system operations.

Login Process:

- Users log in by entering their email and password.
- **Include Relationships:**
  - Mandatory steps like entering email and password.
- **Extend Relationships:**
  - **Forgot Password:** Allows account recovery for forgotten passwords.
  - **Invalid Login:** Handles incorrect login attempts with error messages.
  - **Account Verification:** Ensures the user's identity during login.

Forgot Password Process:

- Users recover accounts by entering a verification code received via email and creating a new password.
- The system verifies the code and ensures the password is confirmed correctly.
- **Extend Relationships:**
  - **Request New Code:** Users can request a new verification code if needed.

Account Verification:

- Ensures the validity of the user's account during login.
- If verification fails, the system triggers the "Invalid Login" scenario, providing recovery options and feedback.

View Profile Information:

- After logging in, users can view their personal details, such as:
    - Skin tone, skin type, allergens, and email address.
- These details are essential and are included under the "Include" relationship.

Log Out Process:

- Users securely exit the system, ending their session.
- Redirects to the main page or login screen for security.

Relationships in the Diagram:

- **Include Relationships:** Represent mandatory steps, like entering credentials or verification codes.
- **Extend Relationships:** Introduce optional paths for handling invalid logins or requesting new verification codes.

Conclusion:

- The use case diagram represents the system's core operations, ensuring a smooth user experience.
- It balances mandatory actions with flexible extensions, covering login, password recovery, profile viewing, and logout functionalities.

Figure 5 flow chart provides a detailed breakdown of a login system process, clearly separating user actions and system responses into two swimlanes: User and Login System. The flowchart ensures a structured pathway for users to either log in or recover their accounts, emphasizing feedback and error handling throughout the process.

Overview

- The flowchart separates the login system process into two swimlanes: **User** and **Login System.**
- It highlights two main pathways: logging in and recovering accounts.

Starting Point

- The process begins with the system displaying the **Login Form** to the user.
- Users can:
    - Enter credentials for login.
    - Select the **Forgot Password** option for account recovery.

Login Process

- The user inputs **email** and **password**.
- The system verifies the credentials:
    - **If valid:** The user is logged in and redirected to the home page.
    - **If invalid:** The system displays the error message:
        - *"Your login information is not correct. Try again."*

Forgot Password Process

- The user requests a **verification code.**
- The system checks the email address:
    - **If not registered:** Displays the message:
        - *"Email is not registered."*
    - **If valid:** Sends a **4-digit verification code** to the user's email.

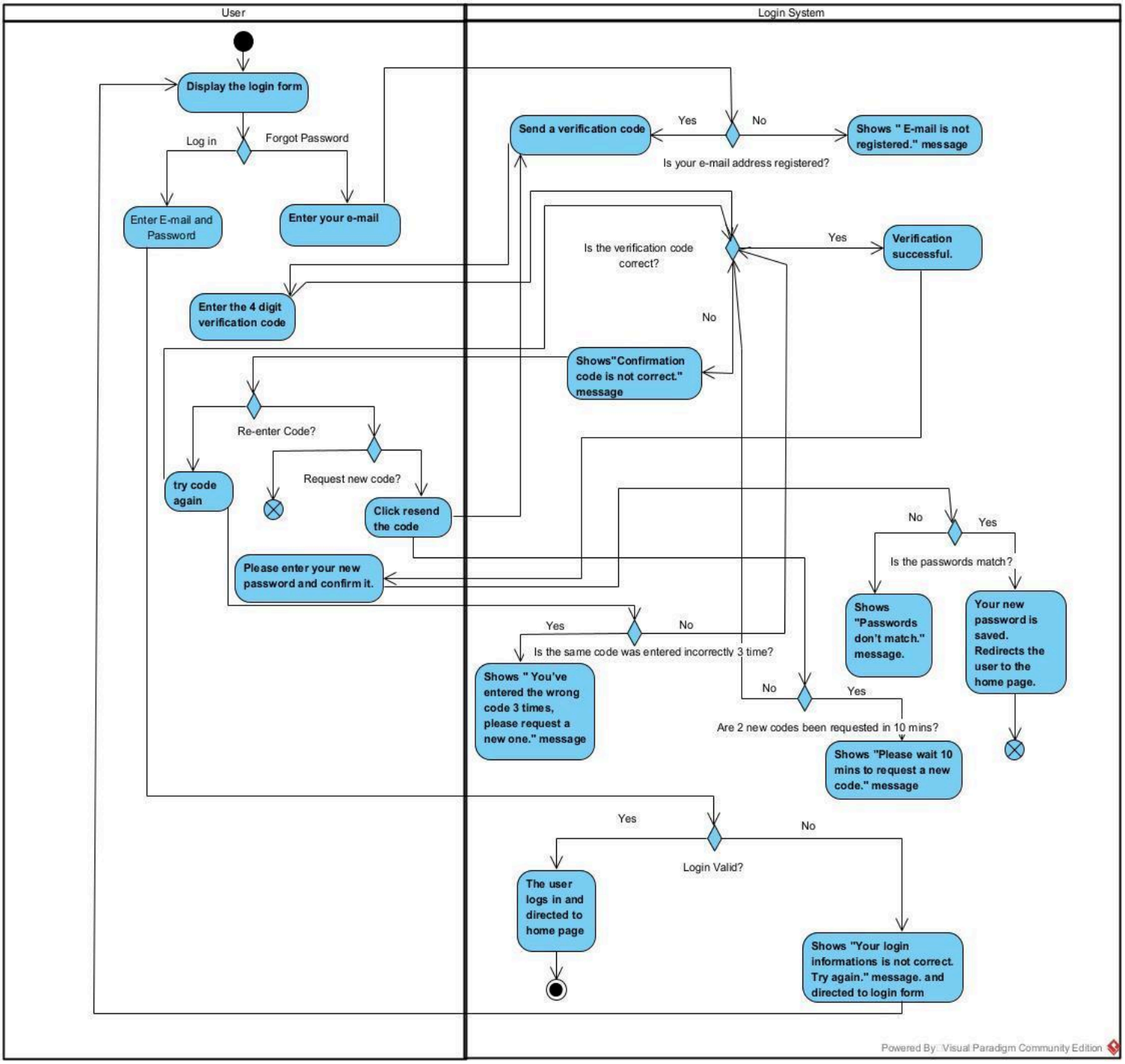

Figure 5 Log In Activity Diagram

Verification Code Entry

- The user enters the received code:
  - **If correct:** Proceeds to the password reset process.
  - **If incorrect:** Displays the message:
    - *"Confirmation code is not correct."*
  - Users can:
    - Re-enter the code.
    - Request a new code.

6. Error Handling in Verification Code Entry

- If the wrong code is entered **3 times:** Displays the message:
    - *"You've entered the wrong code 3 times. Please request a new one."*
- If more than **two codes** are requested within **10 minutes:** Displays the message:
    - *"Please wait 10 minutes to request a new code."*

Password Reset Process

- The user enters and confirms a **new password.**
- The system checks for matching passwords:
    - **If they match:** The password is saved, and the user is redirected to the home page.
    - **If they don't match:** Displays the message:
        - *"Passwords don't match."*

Login Completion

- The process ends when the user:
    - Logs in successfully with valid credentials.
    - Completes the password reset process.
- The user is redirected to the home page.

Error Messages and Feedback

The system provides clear feedback to guide the user:

- *"Email is not registered."*
- *"Confirmation code is not correct."*
- *"You've entered the wrong code 3 times. Please request a new one."*
- *"Passwords don't match."*
- *"Please wait 10 minutes to request a new code."*
- *"Your login information is not correct. Try again."*

Conclusion

- The flowchart provides a clear and user-friendly pathway for login and account recovery.
- It emphasizes feedback and error handling to ensure users navigate the process efficiently and with minimal confusion.

Figure 6 class diagram for the login system illustrates a modular architecture where different components interact to provide authentication, password reset, and email verification functionalities. This design ensures a clear separation of concerns, promoting maintainability and scalability. Below is a detailed explanation of each class and their roles in the system.

Login Form

- **Role:** Represents the user interface for login interactions.
- **Attributes:**
    - emailField: Stores the email entered by the user.
    - passwordField: Stores the password entered by the user.
- **Methods:**
    - submitForm(): Sends the entered credentials to the backend for authentication.
    - forgotPassword(): Redirects the user to the password reset process.

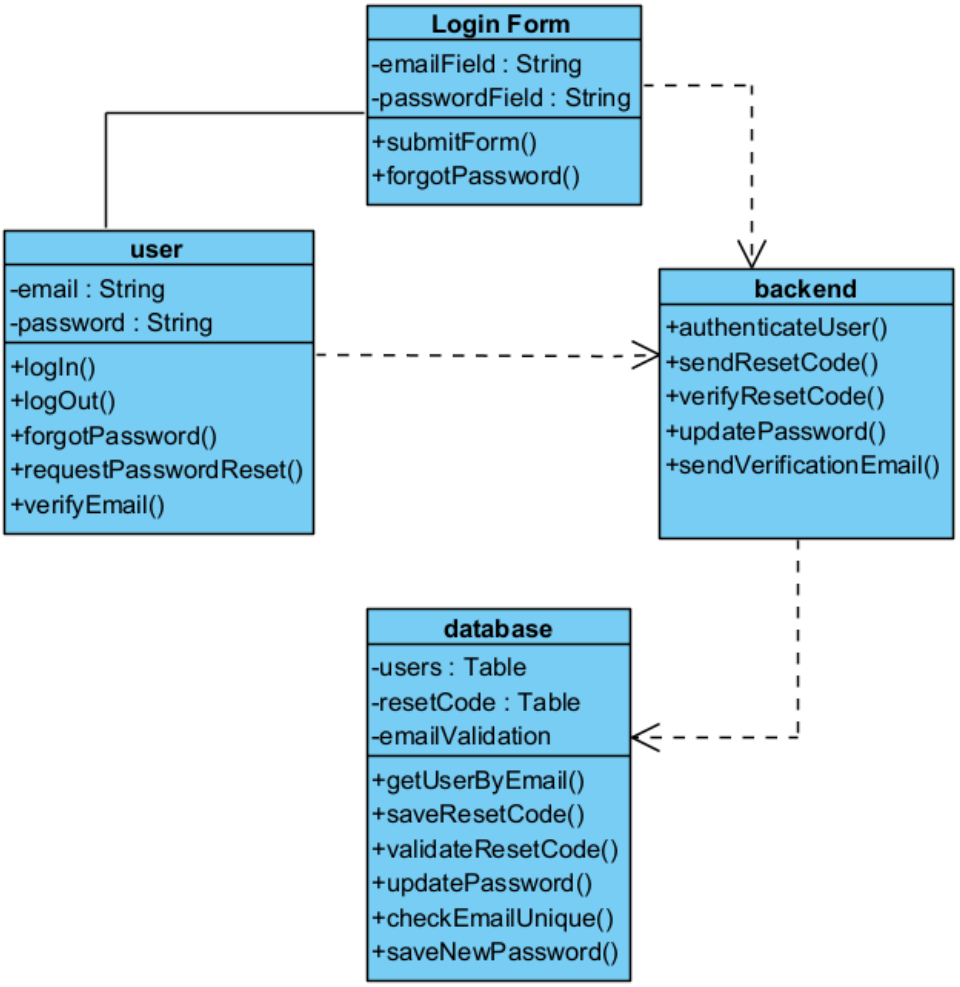

Figure 6 Log In Class Diagram

User

- **Role:** Models the individual interacting with the system.
- **Attributes:**
  - email: Stores the user's email address.
  - password: Stores the user's password.
- **Methods:**
  - logIn(): Authenticates the user.
  - logOut(): Ends the user session.
  - forgotPassword(): Initiates the password recovery process.
  - requestPasswordReset(): Requests a reset code for password recovery.
  - verifyEmail(): Manages email verification tasks.

Backend

- **Role:** Handles the core business logic of the system.
- **Methods:**
  - authenticateUser(): Validates the user's login credentials.
  - sendResetCode(): Sends a password reset code to the user's email.
  - verifyResetCode(): Confirms the validity of the provided reset code.
  - updatePassword(): Updates the user's password after verification.
  - sendVerificationEmail(): Sends an email verification message to the user.

4. Database

- **Role:** Serves as the system's storage mechanism for user data and reset codes.
- **Tables:**
  - users: Stores user email and password information.
  - resetCode: Temporarily holds reset codes for password recovery.
- **Methods:**
  - getUserByEmail(): Retrieves user details based on the email address.
  - saveResetCode(): Saves the reset code for password recovery.
  - validateResetCode(): Confirms the correctness of the reset code.
  - updatePassword(): Updates the password in the database.
  - checkEmailUnique(): Ensures that email addresses are unique.

5. Relationships and Workflow

- **Login Form → User Class:** Facilitates login and password reset requests.
- **User → Backend:** Executes authentication, password recovery, and email verification tasks.
- **Backend → Database:**
  - Retrieves and updates user information.
  - Manages reset codes.
  - Validates email uniqueness.

6. Layered Architecture

- **Frontend (Login Form):** Handles user interactions and input.
- **Backend:** Processes logic and coordinates workflows.
- **Database:** Ensures data persistence and consistency.

7. Key Functionalities

- **Authentication:** Validates login credentials for secure access.
- **Password Recovery:** Supports users in resetting forgotten passwords.
- **Email Verification:** Confirms user email validity during the registration or login process.

8. Conclusion

- The modular design ensures a clear **separation of concerns**, supporting maintainability and scalability.

2.4. Update Profile

Users can update their profile information, including:

- **Email**
  Users can change their email address by entering a new one in the textbox. If the entered email address is already in the database, an error message will be displayed: "This email address is already registered."
- **Skin Type and Skin Tone**
  Users can select a new skin type or skin tone from a dropdown list that appears when clicking on the respective fields.
- **Allergens**
  Users can update their allergens in the textbox. If the input is not in the correct format, the system displays an error message:
  "Please enter allergens in the desired form, such as Paraben, Alcohol, etc."

Update: All updated information is saved to the database if there are no errors.
Cancel: Reverts changes.

Change Password:

- A 4-digit confirmation code is sent to the user's email. The user must enter the code to proceed. If the code is incorrect, an error message appears: *"Confirmation code is incorrect."*
- Users can request a new code if needed. However, the procedures that the system has when sending a verification code also apply to this stage.
- Checks if the password includes at least one uppercase letter, one lowercase letter, one number, and one special character; otherwise, displays an error message: "*Password must contain at least one uppercase letter, one lowercase letter, one number, and one special character.*"
- Once the correct code is entered, the user can set a new password. If the new password and confirmation do not match, the system shows an error: *"Passwords don't match."*
- When successfully updated, a message is displayed: *"Your new password is saved."* Redirects the user to the home page while the user stays logged in.

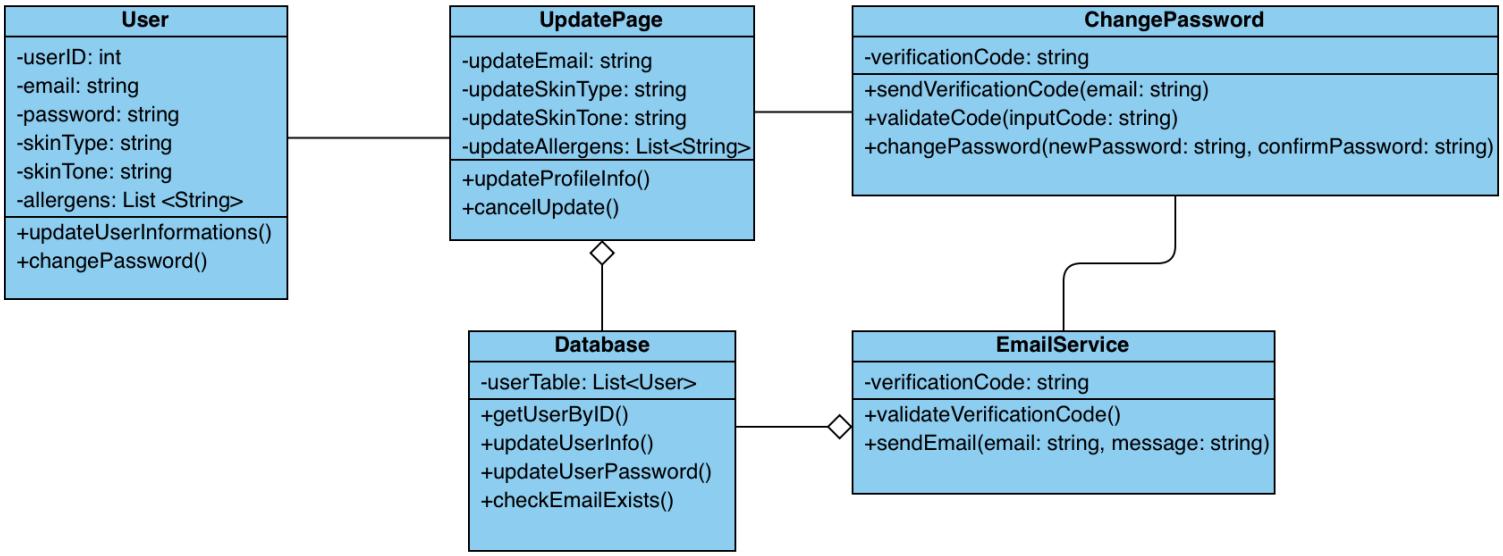

Figure 7 Update Profile Information Class Diagram

This *Figure 7* Class Diagram depicts the architecture and interaction of a user profile management system designed to handle user information updates, password changes, and email-based verification. The system ensures secure and efficient management of user data with functionalities like profile updates and password recovery.

Classes and Attributes:

1. **User**
   - **Attributes:**
     - userID: int – Unique identifier for the user.
     - email: string – Email address of the user.
     - password: string – User's login password.
     - skinType: string – User's skin type (e.g., oily, dry, sensitive).
     - skinTone: string – User's skin tone for personalized recommendations.
     - allergens: List<String> – A list of allergens to avoid in products.
   - **Methods:**
     - updateUserInformations() – Updates the user's personal information.
     - changePassword() – Initiates the password change process.
2. **UpdatePage**
   - **Attributes:**
     - updateEmail: string – New email address to update.
     - updateSkinType: string – New skin type to update.
     - updateSkinTone: string – New skin tone to update.
     - updateAllergens: List<String> – New list of allergens to update.

- **Methods:**
  - updateProfileInfo() – Saves the updated profile information.
  - cancelUpdate() – Cancels the ongoing update process.

3. **Database**
   - **Attributes:**
     - userTable: List<User> – A collection of all user profiles.
   - **Methods:**
     - getUserByID() – Retrieves user details using their ID.
     - updateUserInfo() – Updates user details in the database.
     - updateUserPassword() – Updates the user's password in the database.
     - checkEmailExists()- Checks if the email already exists.
4. **ChangePassword**
   - **Attributes:**
     - verificationCode: string – Code used for email verification.
   - **Methods:**
     - sendVerificationCode(email: string) – Sends a verification code to the specified email address.
     - validateCode(inputCode: string) – Validates the entered verification code.
     - changePassword(newPassword: string, confirmPassword: string) – Changes the user's password upon successful validation.
5. **EmailService**
   - **Attributes:**
     - verificationCode: string – A generated code for email verification.
   - **Methods:**
     - validateVerificationCode() – Verifies the correctness of the sent code.
     - sendEmail(email: string, message: string) – Sends an email to the specified address with a custom message.

Class Relationships:

- The **User** class connects to the **UpdatePage** class, enabling users to update their profile information.
- The **Database** class serves as the central repository, storing and updating user data.
- The **ChangePassword** class works in conjunction with **EmailService** to handle secure password changes, including email-based verification.
- The **EmailService** class facilitates email communication, including sending verification codes and validating them for password recovery or updates.

The Update Profile use case *Figure 8* allows the user to update various attributes of their profile. The following functionalities are supported as extension points:

1. **UpdateEmail:**
   - Allows the user to update their email address.
   - Requires email verification for security.
   - Extends the Update Profile use case.
2. **UpdateSkinType:**
   - Enables the user to modify their skin type information.
   - Extends the Update Profile use case.
3. **UpdateSkinTone:**
   - Allows users to update their skin tone information.
   - Extends the Update Profile use case.
4. **UpdateAllergens:**
   - Allows users to update any allergen information in their profile.
   - Extends the Update Profile use case.
5. **Cancel Update:**
   - Lets the user cancel the profile update process at any point.
   - Extends the Update Profile use case.
6. **Change Password:**
   - Provides the functionality to change the user's password.
   - Includes sub-processes such as entering a confirmation password and submitting a new password.

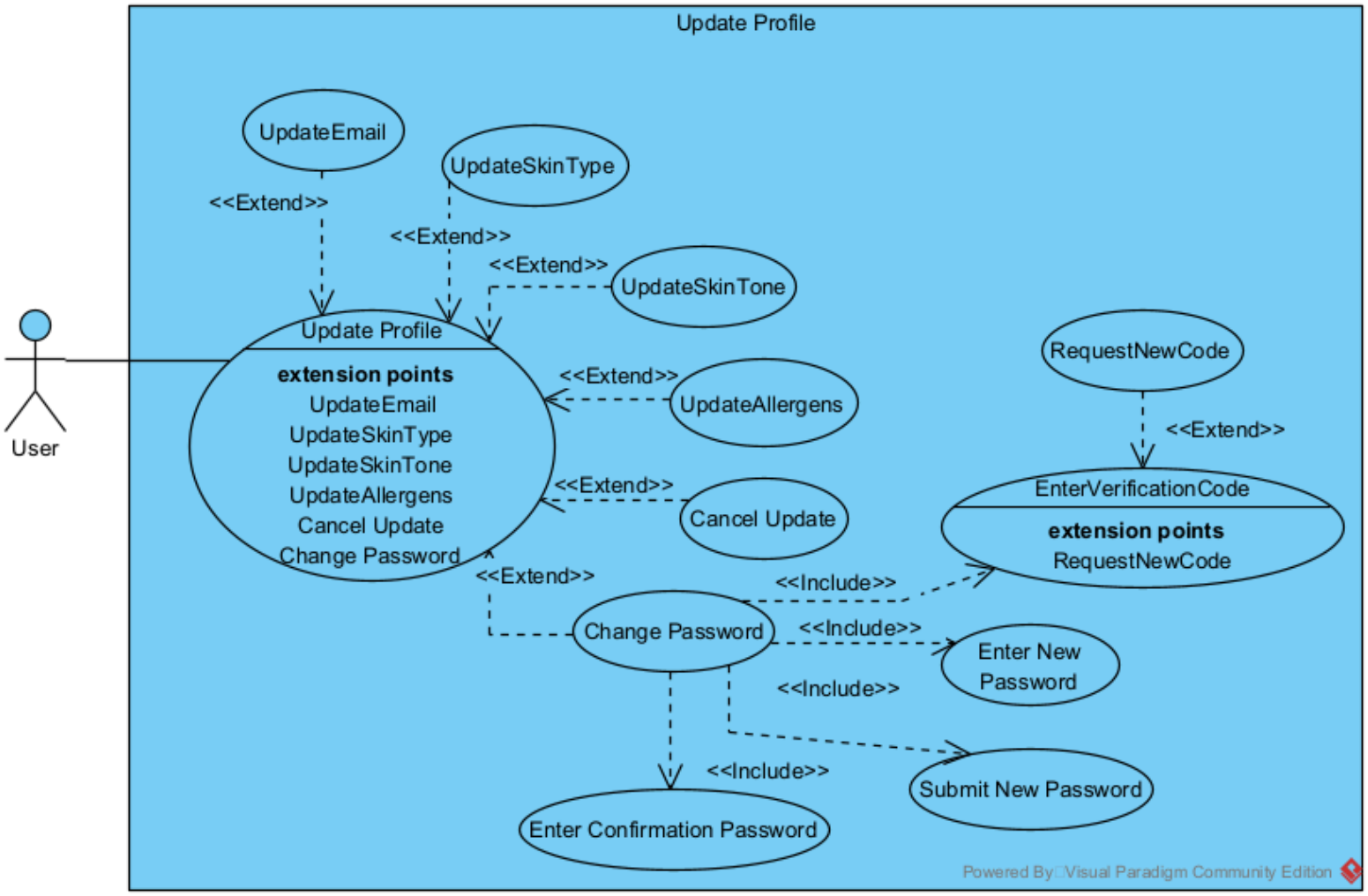

Figure 8 Update Profile Information Use Case Diagram

Sub-Use Cases of Change Password

1. **Enter Confirmation Password:**
   - Users must enter their current password to confirm their identity before changing their password.
   - Included in the Change Password use case.
2. **Enter New Password:**
   - Users input their desired new password.
   - Included in the Change Password use case.
3. **Submit New Password:**
   - Finalizes the password change process by saving the new password.
   - Included in the Change Password use case.

Additional Use Case: EnterVerificationCode

1. **RequestNewCode:**
   - If the verification code is not received or has expired, the user can request a new code.
   - Extends the EnterVerificationCode use case.
2. **EnterVerificationCode:**
   - Handles the process of inputting the verification code required to validate actions like updating the email address.
   - Provides secure verification as an extension point.

Key Relationships

- **Extend:**
  - The Update Profile use case is extended by specialized update functionalities like updating email, skin type, skin tone, allergens, and canceling updates.
  - Similarly, EnterVerificationCode is extended by the option to request a new code if necessary.
- **Include:**
  - The Change Password use case includes smaller steps, such as entering a confirmation password, inputting the new password, and submitting it for finalization.

2.5. Skin Type Test

The Skin Type Test is a core component of GlowGenie that allows users to accurately determine their skin type. This feature is essential for users who are unsure of their skin type or want to validate their knowledge. The process uses machine learning to analyze user responses and classify skin types into the following categories:

- **Dry**
- **Oily**
- **Combination**
- **Normal**

HomePage: Directs the user to the home page.

Submit: Submits the test to be evaluated and saved to the database.

Input

User Responses:

- Users answer a series of multiple-choice questions regarding how their skin behaves under different conditions. These questions are provided in Appendix D of Final Report.

- The form allows users to select one answer per question from predefined options.
- The questions are designed to cover various aspects of skin characteristics, ensuring a comprehensive evaluation.

Mandatory Fields:

- All questions in the Skin Type Test are mandatory. If users attempt to submit the form without completing all required fields, the system will display the following error message **"***Please fill in the mandatory areas.*" and won't proceed to save answers unless all questions are answered.

Processing

1. **Data Collection:**
   - User responses are collected and preprocessed to ensure completeness.
2. **Feature Extraction:**
   - Each question maps to specific features (e.g., high oil production maps to "Oily," low hydration to "Dry").
3. **Machine Learning Model:**
   - A trained classification model (SVM) analyzes the responses.
   - The model is trained on labeled datasets containing skin type information derived from user surveys.
   - Based on the answers, the model predicts the most likely skin type for the user.
4. **Model Output:**
   - The model returns one of the four skin types: Dry, Oily, Combination, or Normal.
5. **Profile Update:**
   - Once the skin type is determined, it is automatically saved to the user's profile.

Output

- The identified skin type is displayed to the user immediately after submission.

Figure 9 in the next page represents the user interactions and underlying workflows of a skin type detection application. The diagram features the **User** actor, interacting with various use cases through **Include** and **Extend** relationships, illustrating how the system responds to different scenarios.

Diagram Components:

1. **Actor:**
    - **User:** The individual who interacts with the system and performs the test.
2. **Use Cases:**
    - **Take Test:** The user initiates the skin type test.
    - **Answer Questions:** The user answers questions related to their skin condition during the test.
    - **Submit Test:** The user submits their responses after completing the test.
    - **View Test Result:** After submission, the user views their skin type result.
    - **Update User Profile:** The user's profile is updated based on the test results.
    - **Handle Error:** If there are missing fields, the system notifies the user and prompts correction.

Diagram Relationships:

1. Include Relationship:

- The **"Take Test"** process **includes** the **"Answer Questions"** use case.
    - **Reason:** For a test to be completed, the user must answer the questions. This step is **mandatory** and occurs as an **integral part** of every test without exception.
    - **Technical Explanation:** The "Answer Questions" use case is **essential** for the "Take Test" process to proceed. The test cannot advance if the questions are not answered.
- The **"Submit Test"** use case **includes** the **"Update User Profile"** use case.
    - **Reason:** When the test is submitted, the user's profile is automatically updated. This process occurs **sequentially** and does not require direct user intervention.
    - **Technical Explanation:** The "Update User Profile" use case is **an inherent part** of the "Submit Test" process. Once the test data is submitted, the profile update **inevitably** takes place.

2. Extend Relationship:

- The **"Submit Test"** use case is **extended** by the **"Handle Error"** use case.
    - **Reason:** When the user submits the test, the system automatically checks for missing or incorrect fields. If any are detected, the **"Handle Error"** process is triggered.
    - **Technical Explanation:** The "Handle Error" use case is **conditional** and activates **only if** missing responses or errors are found during test submission. If the test is submitted without errors, the "Handle Error" process is **skipped**, and the main flow continues uninterrupted.
    - **Decision Point:** The "Submit Test" process contains a **decision point**, and the "Handle Error" use case operates as an **alternate flow** triggered by error detection.

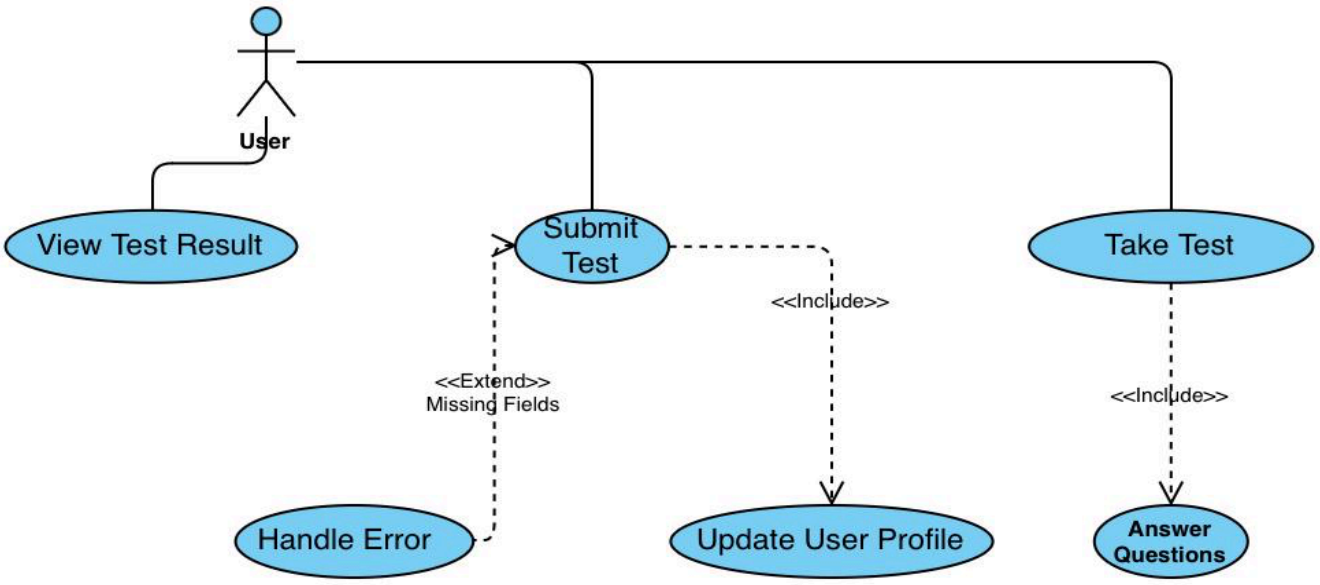

Figure 9 Skin Type Test Use Case Diagram

Figure 10 represents the process of a skin type detection test, from the initiation by the user to the storage of results in the database. The diagram consists of five components (swimlanes): User, Frontend, Backend, Machine Learning (ML Model), and Database.

1. **Start:**
   - The user initiates the process.
2. **Display Questions:**
   - The system displays test questions to the user. This action takes place in the User swimlane.
3. **Answer Questions:**
   - The user answers the questions.

4. **Start:**
   - The user initiates the process.
5. **Display Questions:**
   - The system displays test questions to the user. This action takes place in the User swimlane.
6. **Answer Questions:**
   - The user answers the questions.
7. **Submit Answer:**
   - The user submits their responses, transitioning the process to the Frontend layer.
8. **Check for Mandatory Fields:**
   - The Frontend checks if all mandatory fields are filled.
   - If fields are missing, the process proceeds to the Show Error step, requiring the user to complete the missing fields.
   - If no fields are missing, the process moves to the Backend.
9. **Send Answers to ML Model:**
   - The Backend sends the responses to the Machine Learning Model for analysis.
10. **Evaluate Answers:**
    - The ML Model evaluates the responses and determines the user's skin type.
11. **Display Result:**
    - The detected skin type is sent to the Frontend and displayed to the user.
12. **Save Skin Type:**
    - The identified skin type is saved in the Database through the Backend. This is the final step of the process.
13. **End:**
    - The process concludes once the skin type is stored in the database.

Technical Points:

- **Decision Node:**
  - Mandatory fields are checked. If any fields are missing, the user is prompted with an error message.

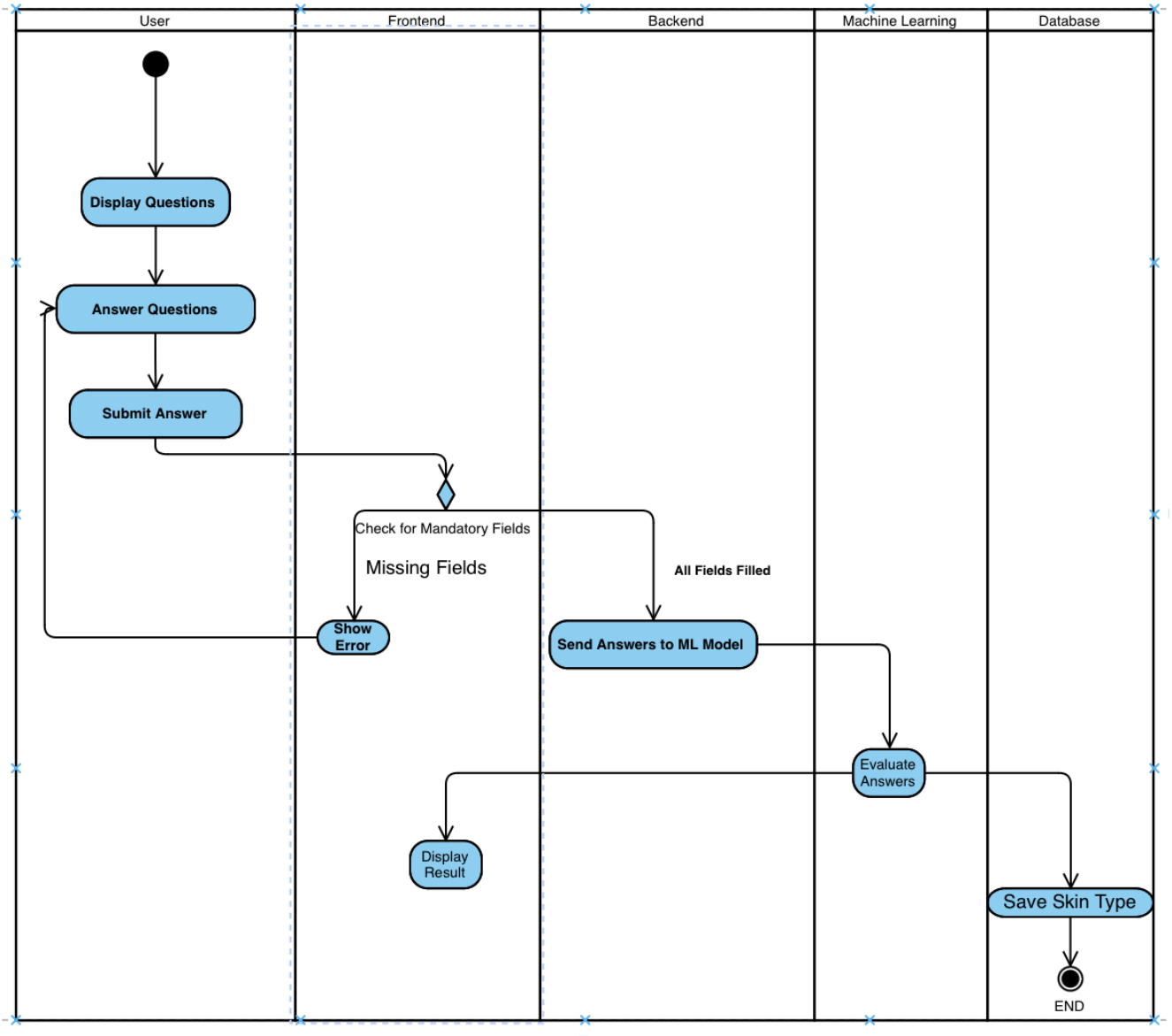

Figure 10 Skin Type Test Activity Diagram

Figure 11 illustrates the core components and relationships within a skin type detection system. The diagram represents the flow where a user completes a form, submits answers, and the machine learning model processes these answers to predict the skin type, updating the user's profile with the results.

Classes and Attributes:

1. **SkinTypeTestForm:**
   - **Attributes:**
     - questions: List<String> – A list of questions presented to the user.
     - answers: List<String> – A list of responses provided by the user.
   - **Methods:**
     - displayForm() – Displays the test form to the user.
     - validateForm() – Validates user responses for completeness and accuracy.

2. **MLModel:**
    - **Attributes:**
        - usersAnswers – Answers submitted by the user.
    - **Methods:**
        - predictSkinType() – Analyzes responses and predicts the user's skin type.
3. **Result:**
    - **Attributes:**
        - skinType: String – The detected skin type.
        - date: Date – The date the test was conducted.
    - **Methods:**
        - displayResult() – Displays the test results to the user.
4. **User:**
    - **Attributes:**
        - userID: Integer – Unique identifier for the user.
        - skinType: String – The user's skin type.
        - profileData: String – Additional profile information.
        - skinTone: String – The user's skin tone.
    - **Methods:**
        - updateProfile() – Updates the user's profile based on test results.

Class Relationships:

- The **SkinTypeTestForm** class enables the user to complete the skin type test (**completes**).
- Answers are sent to the **MLModel** class (**sends answers to**), where they are analyzed, and predictions are made.
- The **MLModel** returns results to the **Result** class (**returns**).
- The **Result** class updates the **User** class with the detected skin type (**updates**).

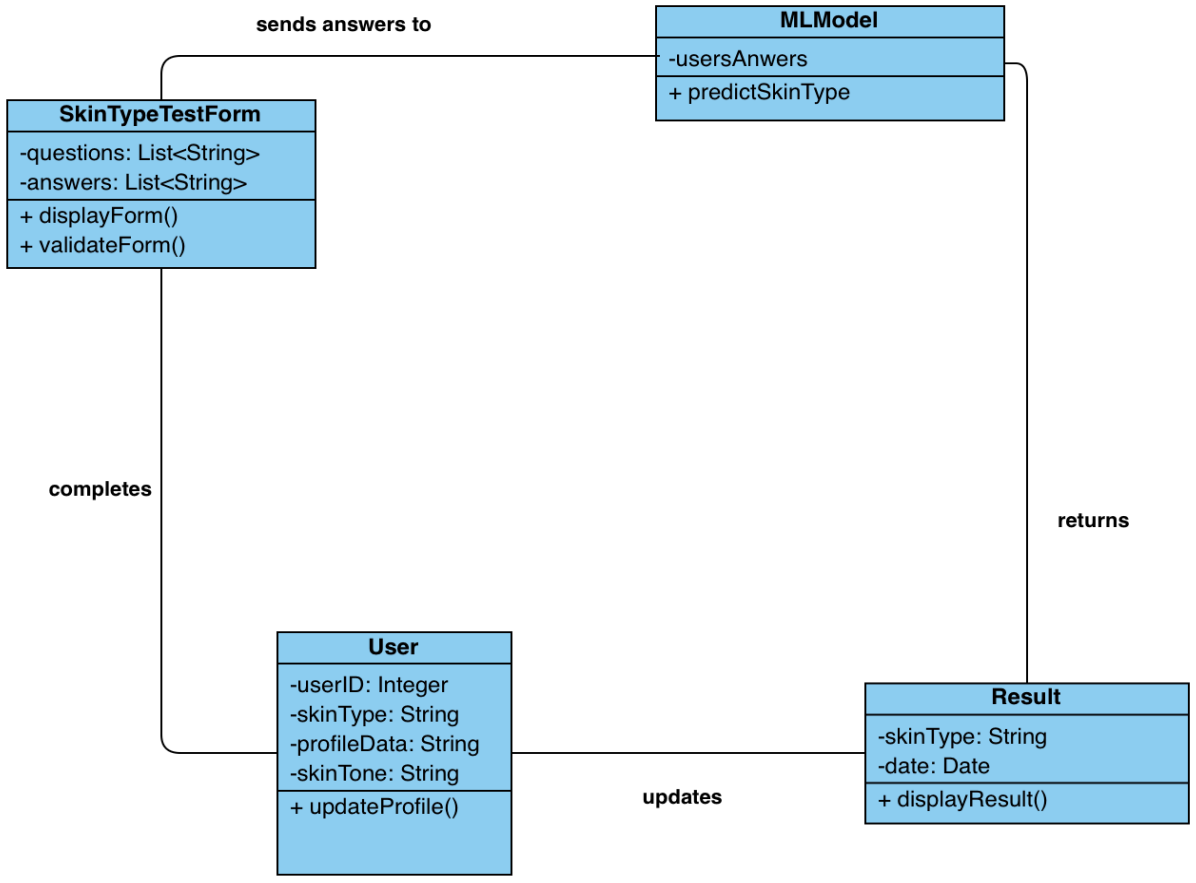

Figure 11 Skin Type Prediction Class Diagram

2.6. Product Suitability Feedback

This feature enables users to evaluate whether a specific product is compatible with their skin type. It is implemented using a combination of machine learning and GPT API integration, ensuring personalized and accurate recommendations.

Input:

If skinType = "I don't know", display a pop-up when the user attempts to access the suitability feature.

- Navigate to the home page to update their skin type.
- Navigate to take the Skin Type Test.

HomePage: Directs the user to the home page.

Processing Steps

1. Data Retrieval

The system searches a product database or external APIs to fetch details about the entered product. The system searches products in the database according to the user's profile information (skin type, skin tone, allergens if entered). It shows whether the product the user enters is suitable for their skin type. These include:

- **Ingredient List**: The core components of the product.
- **Concentration Levels**: Ingredient proportions, which are important for determining their impact.
- **Category**: The category of the product (must match one of the five predefined categories: *moisturizer, toner, sunscreen, cleanser, serum*).
    - If the product is a **sunscreen**, the system also retrieves information about which **skin tones** it is suitable for.

2. Skin Type Labeling Model

- This model is responsible for assigning a skin type label to a product based on its ingredients.

Purpose:

- To classify newly queried products and label them with the skin type(s) they are suitable for.

Input:

- **Features**
    - Ingredients List
    - Product Category
    - Skin Type It's Suitable for (to filter for sunscreen).

Output

- **Skin Type Label:**

  A classification label (e.g., "Oily", "Dry", "Normal", "Combination").

Model Training:

- Random forest is going to be used to train the model.

3.Ingredient Contribution Analysis Model

- This model is responsible for assigning a skin type label to a product based on its ingredients.

Purpose:

- To analyze and give feedback on which ingredients list the chosen product being classified as suitable or unsuitable for a specific skin type.

Input:

Features:

- Ingredient List(same as Skin Type Labeling Model).
- Product Category
- Skin type It's Suitable for (to filter sunscreen) output from Predicted skin type label.

Output:

- A list of ingredients contributing to suitability or unsuitability for a specific skin type.

Model Training:

- Random forest model paired with SHAP (Shapley Additive Explanations) to evaluate feature importance.

Ingredient Analysis:

- After Skin Type Labeling Model assigns a skin type label to a product, going to use SHAP values or feature importances to determine which ingredients contributed most to the classification

The *Figure 12* in the next page illustrates the step-by-step workflow for generating tailored product recommendations based on the user's skin type, skin tone, and allergens. The process includes five key components: **User**, **Frontend**, **Backend**, **Database**, and **ML Model**.

1. **User Interaction**:
   - The user clicks on the "Generate Products for Me" button.
   - If the user's skin type is marked as "I don't know," a pop-up is shown to prompt them to either update their skin type on the home page or take a skin type test.
2. **Frontend Actions**:
   - After the user updates or confirms their skin type, they proceed to choose product categories (e.g., cleanser, toner).
   - A filtered list of products from the selected categories is displayed based on user preferences.
3. **Backend Actions**:
   - The backend gathers necessary input (skin type, tone, allergens, and categories) to query the product database.
   - If no products satisfy the user's preferences for a category, an error message is displayed, listing the categories without suitable products.
4. **ML Model Integration**:
   - When the user selects a product, the ML model evaluates its ingredients to determine suitability for the user's skin type.
   - The ML model checks if the product has been previously evaluated for the same skin type.
   - Feedback is generated, explaining why the product is suitable or unsuitable based on its ingredients.
5. **Output**:
   - Suitable products are framed in **green**, and unsuitable products in **red**.
   - Users can view detailed feedback on product suitability.

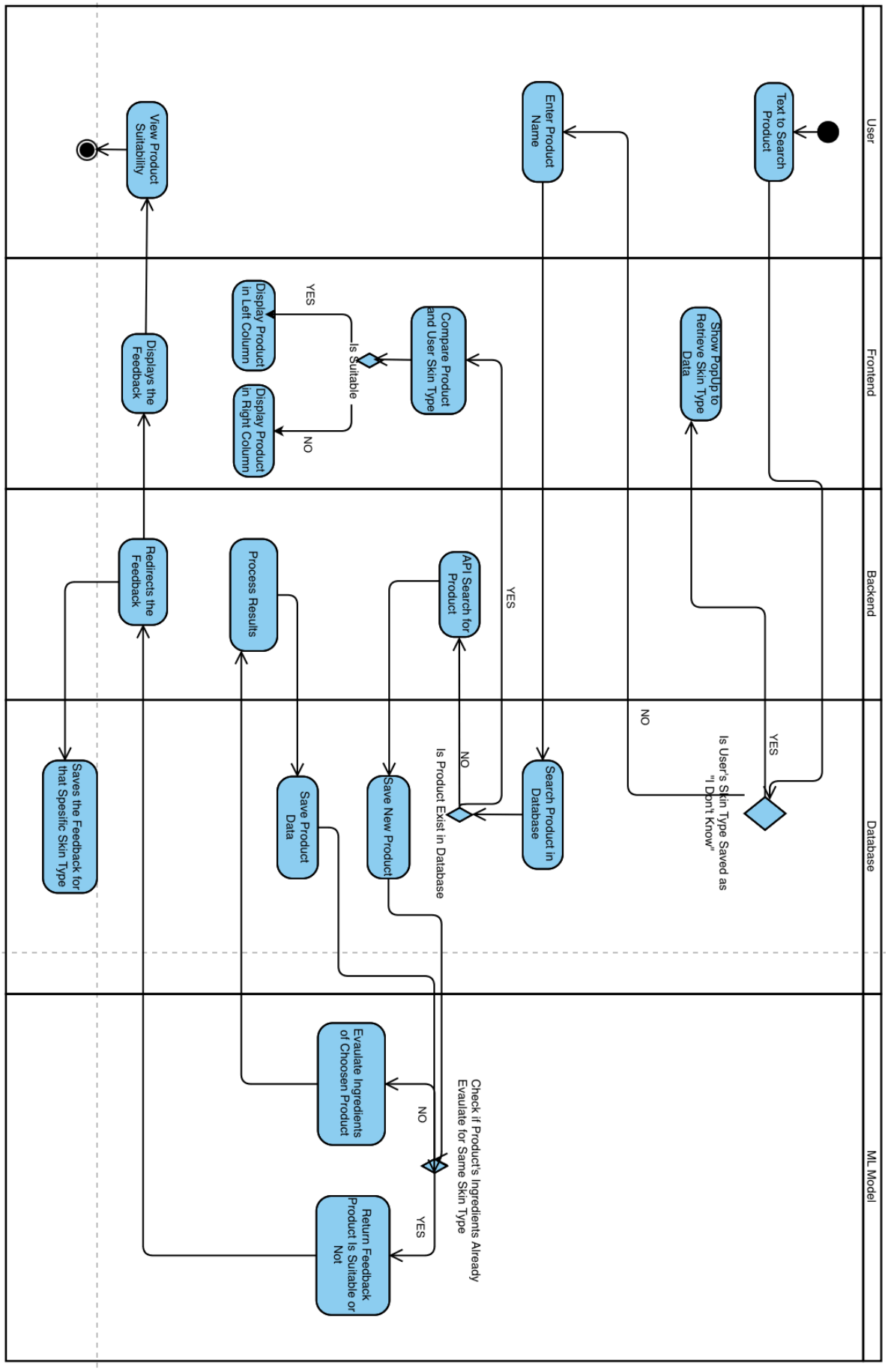

Figure 12 Product Suitability Feedback System Activity Diagram

Compatibility Result

The results are presented in two columns:

1. **Left Column**: Products suitable for the user's skin type.
2. **Right Column**: Products unsuitable for the user's skin type.

Each product is displayed in a **card/frame** format containing:

- **Green**: Suitable for the user.
- **Red**: Not suitable for the user.

Users can click on a product to view its **detailed information** in a pop-up, including:

- Product Name
- Ingredients
- Feedback that explains which specific ingredients in the product make it suitable or unsuitable for the user's skin type

This Activity Diagram illustrates the workflow of the Product Suitability Feedback feature, guiding the process from a user's product search to the final evaluation of product compatibility with their skin type. The diagram is divided into five swimlanes: **User**, **Frontend**, **Backend**, **Database**, and **ML Model**, representing different components involved in the process.

Diagram Components and Workflow:

1. **Start:**
    - The process begins when the user initiates a product search by entering text in the search bar.
2. **Retrieve Skin Type:**
    - If the user's skin type is saved as "I Don't Know," a pop-up appears prompting the user to take a skin type test or update their profile on the home page. This ensures the system has sufficient information to provide accurate recommendations.
3. **Enter Product Name:**
    - The user inputs the name of the product they wish to evaluate.
4. **Product Search:**
    - The backend performs a product search in the database.
    - If the product exists, the system proceeds to API calls to retrieve further details.
    - If the product is not found, it is saved as a new entry in the database along with relevant product data.
5. **Evaluate Product Ingredients::**
    - The system checks if the product's ingredients have already been evaluated for the user's skin type.
    - If the evaluation exists, the ML Model returns feedback immediately.If no fields are missing, the process moves to the Backend.
    - If not, the ML Model evaluates the product ingredients and classifies the product as either suitable or unsuitable.
6. **Comparison and Display::**
    - The frontend compares the evaluated product with the user's skin type.
    - Products deemed suitable are displayed in the left column, while unsuitable products appear in the right column.
7. **Feedback and Viewing:**
    - The user can view the product suitability details, which include ingredient analysis and explanations.
8. **Save Feedback:**
    - The feedback, including product compatibility with the user's skin type, is saved in the database for future reference.
9. **End:**
    - The process concludes when the user views the product feedback, and the system saves the evaluation results.

Technical Points:

- **Decision Node:**
  - The system checks if the user's skin type is known. If unknown, the user must update their profile before proceeding.
  - The product existence check determines whether the product needs to be added to the database.
- **Machine Learning Integration:**
  - The ML Model evaluates the product's ingredient list using classification techniques and SHAP values to provide transparent ingredient contribution analysis.
- **Database Update:**
  - Product evaluations and feedback are stored in the database for recurring user queries, improving response times for future searches.

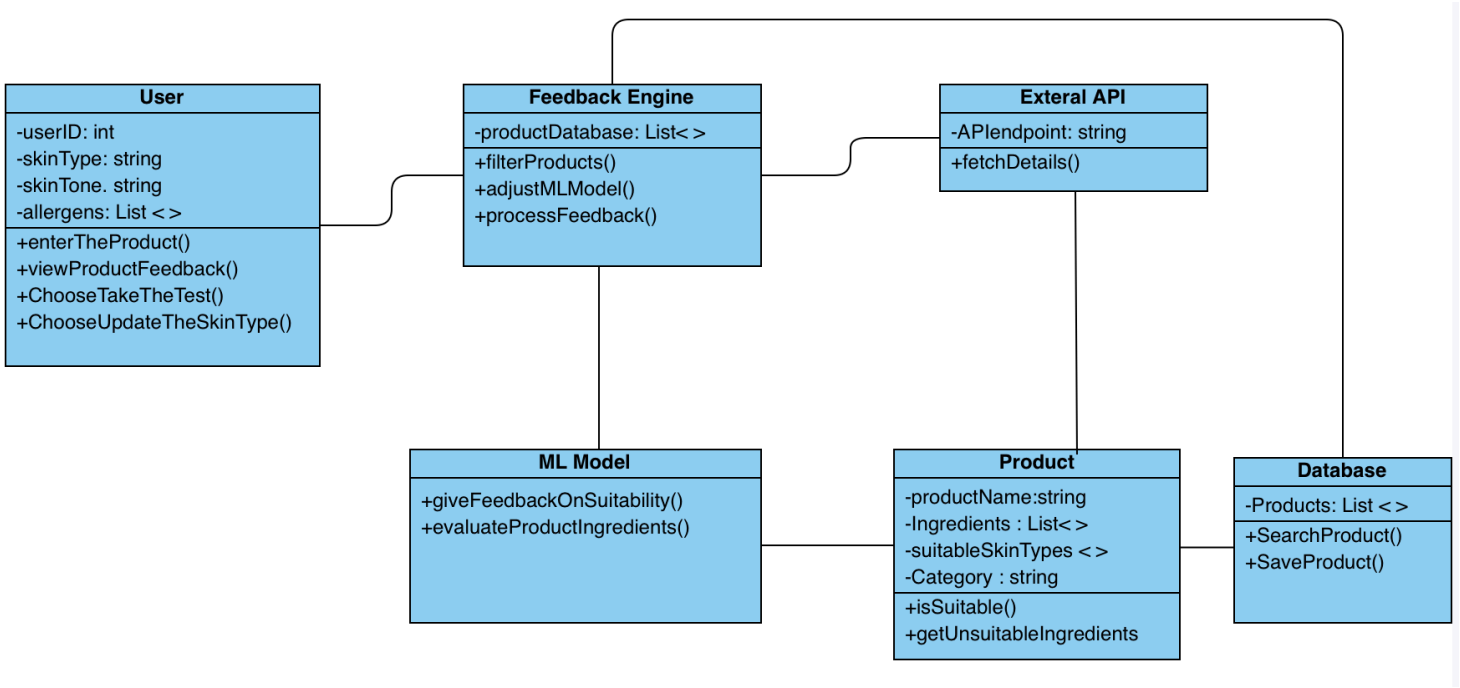

Figure 13 Product Suitability Feedback Class Diagram

This *Figure 13* Class Diagram illustrates the architecture and interactions within the product suitability feedback system. The system evaluates skincare products based on user attributes (skin type, skin tone(for sunscreen) and allergens) and provides feedback using machine learning models and external product databases.

Classes and Attributes:

1. **User:**
   - **Attributes:**
     - userID: int – Unique identifier for the user.
     - skinType: string – User's skin type (e.g., oily, dry, sensitive).
     - skinTone: string – User's skin tone (for evaluating products like sunscreen).
     - allergens: List<> – List of allergens to avoid.
   - **Methods:**
     - enterTheProduct() – Allows the user to input a product name.
     - viewProductFeedback() – Displays the feedback on product compatibility.
     - ChooseTakeTheTest() – Directs the user to take a skin type test.
     - ChooseUpdateTheSkinType() – Updates user's skin type profile.
2. **Feedback Engine:**
   - **Attributes:**
     - productDatabase: List<> – A list of products available for feedback and analysis.
   - **Methods:**
     - filterProducts() – Filters products based on user attributes and queries.
     - adjustMLModel() – Adjusts machine learning parameters for better predictions.
     - processFeedback() – Processes and refines product suitability feedback.

3. **ML Model:**
    - **Methods:**
        - giveFeedbackOnSuitability() – Generates feedback on whether a product is suitable for the user.
        - evaluateProductIngredients() – Evaluates the ingredients of a product to determine its compatibility.
4. **Product:**
    - **Attributes:**
        - productName: string – The name of the skincare product.
        - Ingredients: List<> – A list of ingredients present in the product.
        - suitableSkinTypes<> – Skin types for which the product is suitable.
        - Category: string – Product category (e.g., moisturizer, sunscreen).
    - **Methods:**
        - isSuitable() – Determines if the product is suitable for the user's skin type.
        - getUnsuitableIngredients() – Lists ingredients that may cause issues for the user.
5. **External API:**
    - **Attributes:**
        - APIendpoint: string – The endpoint used to fetch external product details.
    - **Methods:**
        - fetchDetails() – Fetches product data from external databases.
6. **Database:**
    - **Attributes:**
        - Products: List<> – A list of stored products.
    - **Methods:**
        - SearchProduct() – Searches for products in the database.
        - SaveProduct() – Saves new product entries.

Class Relationships:

- The **User** class interacts with the **Feedback Engine** to input products and receive feedback.
- The **Feedback Engine** communicates with both the **ML Model** and the **External API** to filter, adjust, and process product data.
- The **ML Model** evaluates product suitability and returns results to the **Feedback Engine.**
- **Products** are stored and retrieved through the **Database**, and the **External API** can fetch additional product details if needed.

2.7. Generate Product Recommendations

The system provides product recommendations tailored to the user's skin type, skin tone (for sunscreen), and allergens, if specified. The process involves:

Input

- **Skin Type**: Known from user input or determined via the skin type test. If the user's skin type is saved as **"I don't know"**, a mandatory pop-up will appear before accessing the recommendation interface. This pop-up requires the user to either:
    - ➔ Redirects the user to the Home Page so they update their skin type.
    - ➔ Choose to take the **Skin Type Test**.
- **Skin Tone**: Entered during registration.
- **Allergens**: Optional, entered by the user in a textbox.
- **Categories**: Selected by the user from available options (moisturizer, toner, sunscreen, cleanser, serum).

Processing

- Products in the database are filtered and listed for the user's view according to their profile information (skin type, skin tone, allergens if entered.) and also the categories they've chosen. The user can view a filtered list of items. If the user selects a product, a pre-trained machine learning model (Ingredient Contribution Analysis Model from 2.7) analyzes its ingredients and provides feedback, explaining why the product is suitable or unsuitable for their skin type.

- Products are visually framed:
    - **Green**: Suitable for the user.
    - **Red**: Not suitable for the user.

Output

- A list of products under each selected category, including:
    - Product Name
    - Ingredients
    - Feedback that explains which specific ingredients in the product make it suitable or unsuitable for the user's skin type

Error Handling

- At the start, the product database may not contain many entries. If no compatible product is found in the selected category or for the user's skin type, the system will return the following error message:
  "*There are no suitable products in [chosen_categories] category/categories for your skin type in our database at the moment. You can help us improve by querying products through our Suitability Service. This will add new products to our database and expand our recommendations over time.***"**

User Interface

- The user can:
    - View all products that are in the database, with suitability highlighted using the green/red framing system.
    - Select a product to view its detailed information including its ingredients and feedback given from the system.

HomePage: Directs the user to the home page.

Figure 14 illustrates the step-by-step workflow for generating tailored product recommendations based on the user's skin type, skin tone, and allergens. The process includes five key components: **User**, **Frontend**, **Backend**, **Database**, and **ML Model**.

1. **User Interaction**:
    - The user clicks on the "Generate Products for Me" button.
    - If the user's skin type is marked as "I don't know," a pop-up is shown to prompt them to either update their skin type on the home page or take a skin type test.
2. **Frontend Actions**:
    - After the user updates or confirms their skin type, they proceed to choose product categories (e.g., cleanser, toner).
    - A filtered list of products from the selected categories is displayed based on user preferences.
3. **Backend Actions**:
    - The backend gathers necessary input (skin type, tone, allergens, and categories) to query the product database.
    - If no products satisfy the user's preferences for a category, an error message is displayed, listing the categories without suitable products.
4. **ML Model Integration**:
    - When the user selects a product, the ML model evaluates its ingredients to determine suitability for the user's skin type.
    - The ML model checks if the product has been previously evaluated for the same skin type.
    - Feedback is generated, explaining why the product is suitable or unsuitable based on its ingredients.
5. **Output**:
    - Suitable products are framed in **green**, and unsuitable products in **red**.
    - Users can view detailed feedback on product suitability.

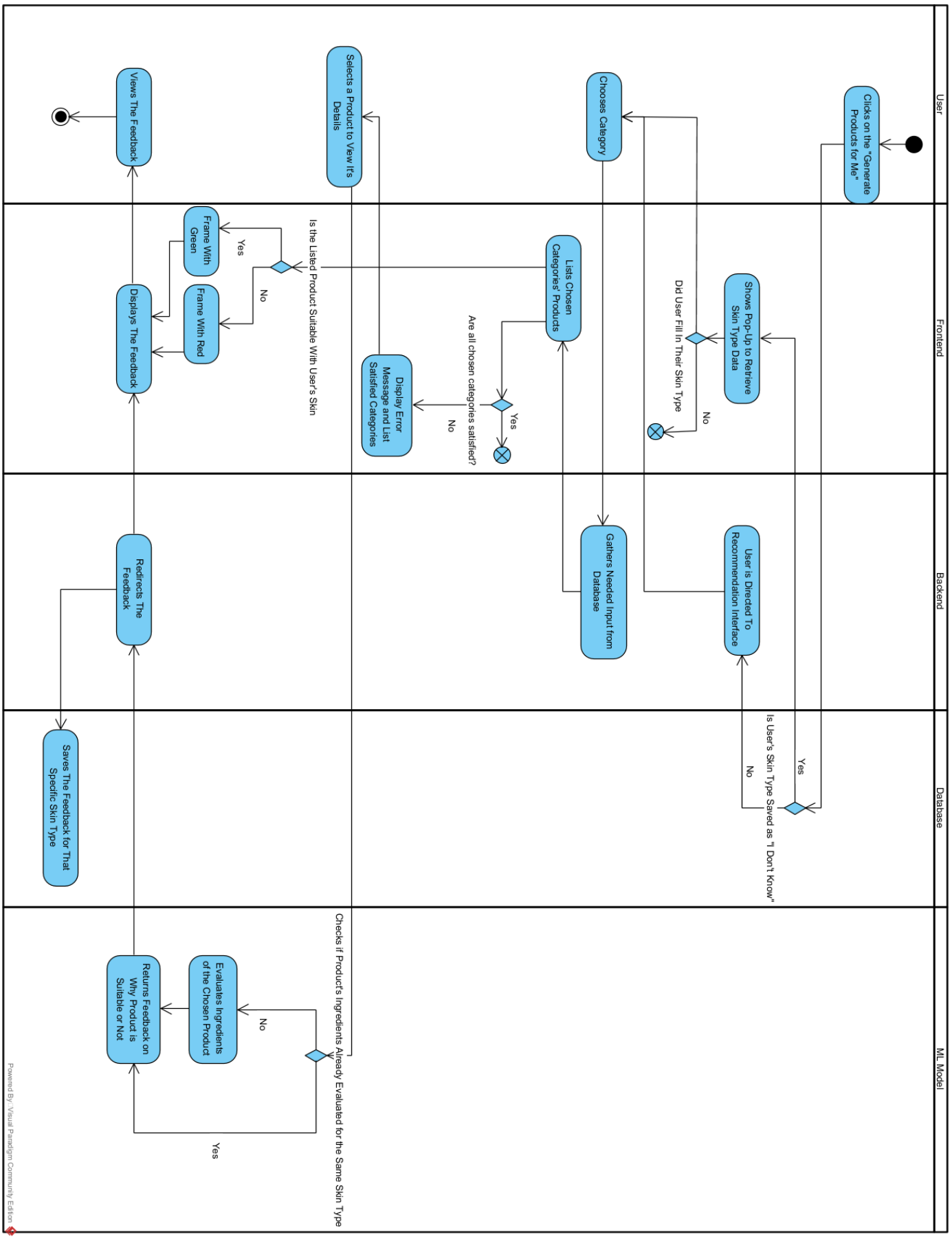

Figure 14 Generate Product Recommendations Activity Diagram

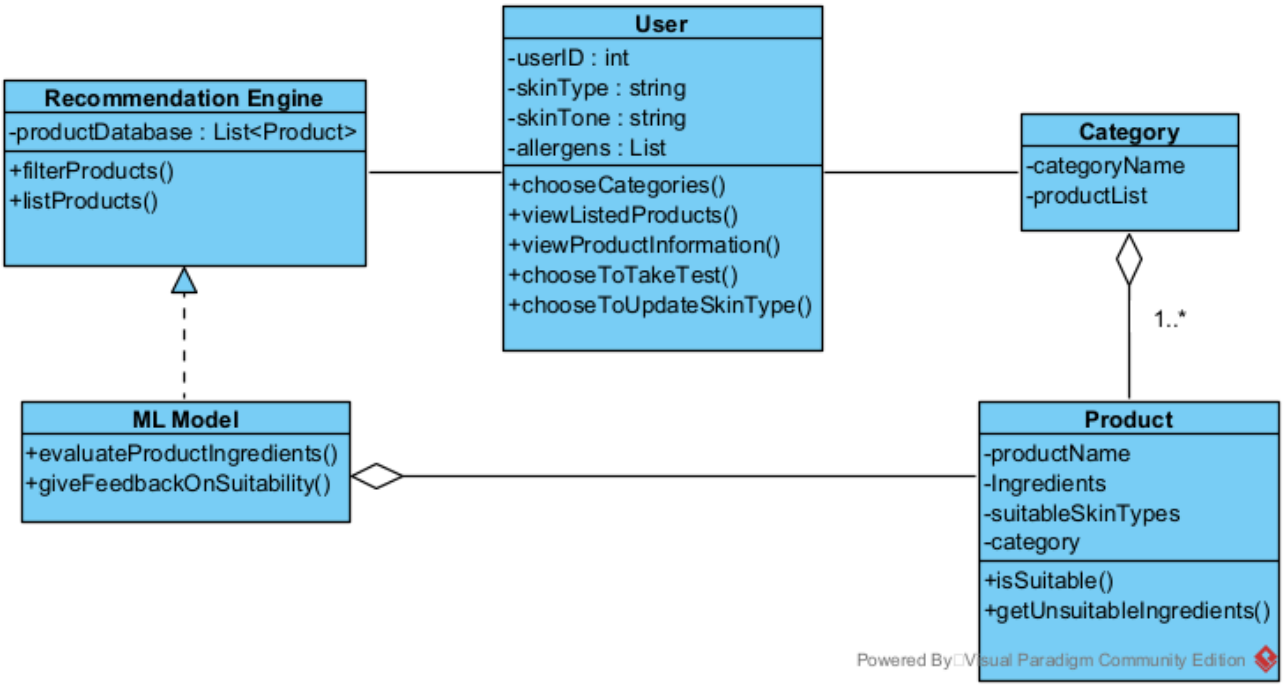

Figure 15 Generate Product Recommendations Class Diagram

Figure 15 outlines the structure and relationships of the system components responsible for generating product recommendations.

1. **Classes**:
    - **User**:
        - Represents the user of the system, with attributes like userID, skinType, skinTone, and allergens.
        - Contains methods to choose categories, view listed products, view product details, take the skin type test, and update skin type.
    - **Category**:
        - Represents product categories (e.g., cleanser, moisturizer).
        - Contains attributes such as categoryName and productList.
        - Has a one-to-many relationship with the **Product** class.
    - **Product**:
        - Represents individual products in the database.
        - Attributes include productName, ingredients, suitableSkinTypes, and category.
        - Methods include checking suitability (isSuitable) and retrieving unsuitable ingredients (getUnsuitableIngredients).
    - **Recommendation Engine**:
        - Manages the product database and provides filtered product lists based on user preferences.
        - Contains methods for filtering and listing products.
    - **ML Model**:
        - Evaluates product ingredients to determine suitability using machine learning algorithms.
        - Provides detailed feedback explaining why a product is suitable or unsuitable.
2. **Relationships**:
    - The **User** interacts with the **Recommendation Engine** to filter and view products.
    - The **Recommendation Engine** relies on the **ML Model** to evaluate product ingredients and provide feedback.
    - Each **Category** is associated with multiple **Product** objects.
3. **Key Functionality**:
    - The system evaluates products for suitability based on user preferences (skin type, tone, allergens) using the recommendation engine and ML model.
    - Feedback is saved in the database for future reference, ensuring continuous improvement of recommendations.

2.8. Update Ingredients of Skin Care Products'

The purpose of this function is to ensure that product data remains accurate and up-to-date by periodically refreshing ingredient information and re-evaluating product suitability. This enhances the reliability and relevance of recommendations provided to users.

Input

- Expiration Time: Each product in the database has an assigned expiration time for its ingredient data.
- Product ID: The unique identifier for the product whose ingredient data needs to be updated.

Process

Check Expiration:

- The system periodically checks all products in the database to determine if their expiration time has been reached.

Retrieve Updated Information:

- Query GPT to fetch the latest ingredient information for products whose data has expired.
- Replace the outdated ingredient list with the newly retrieved data.

Re-evaluate Product Suitability:

- Use the updated ingredient list as input to the machine learning model.
- Reclassify the product by determining its suitability for specific skin types.

Database Update:

Replace the outdated product details in the database with the updated information, including the new suitability classification.

Output

- A refreshed and updated product database.
- Accurate suitability classifications for all products, ensuring high-quality recommendations.

Benefits

- Maintains the accuracy and relevance of the product database over time.
- Improves user trust by providing reliable recommendations based on up-to-date information.

Figure 16 shows the system that identifies expired products, updates their ingredient data via ChatGPT, and reclassifies them using an ML model. The backend processes the updated information and saves it to the database. This ensures accurate and reliable product recommendations.

1. Database

1. Periodically Check Expiration Time:
   The database periodically checks if any product's expiration time has been reached.
2. Is Product Outdated?
   - If the product is not outdated, terminate the process for that product.
   - If outdated, add it to the list of expired products.
3. Add to List of Expired Products:
   The list of expired products is sent to the backend for further processing.

2. Backend

4. Receive Expired Products:
   The backend receives the list of expired products from the database.
5. Send Prompt for Every Product:
   For each expired product, a query is sent to ChatGPT to retrieve updated ingredient data.

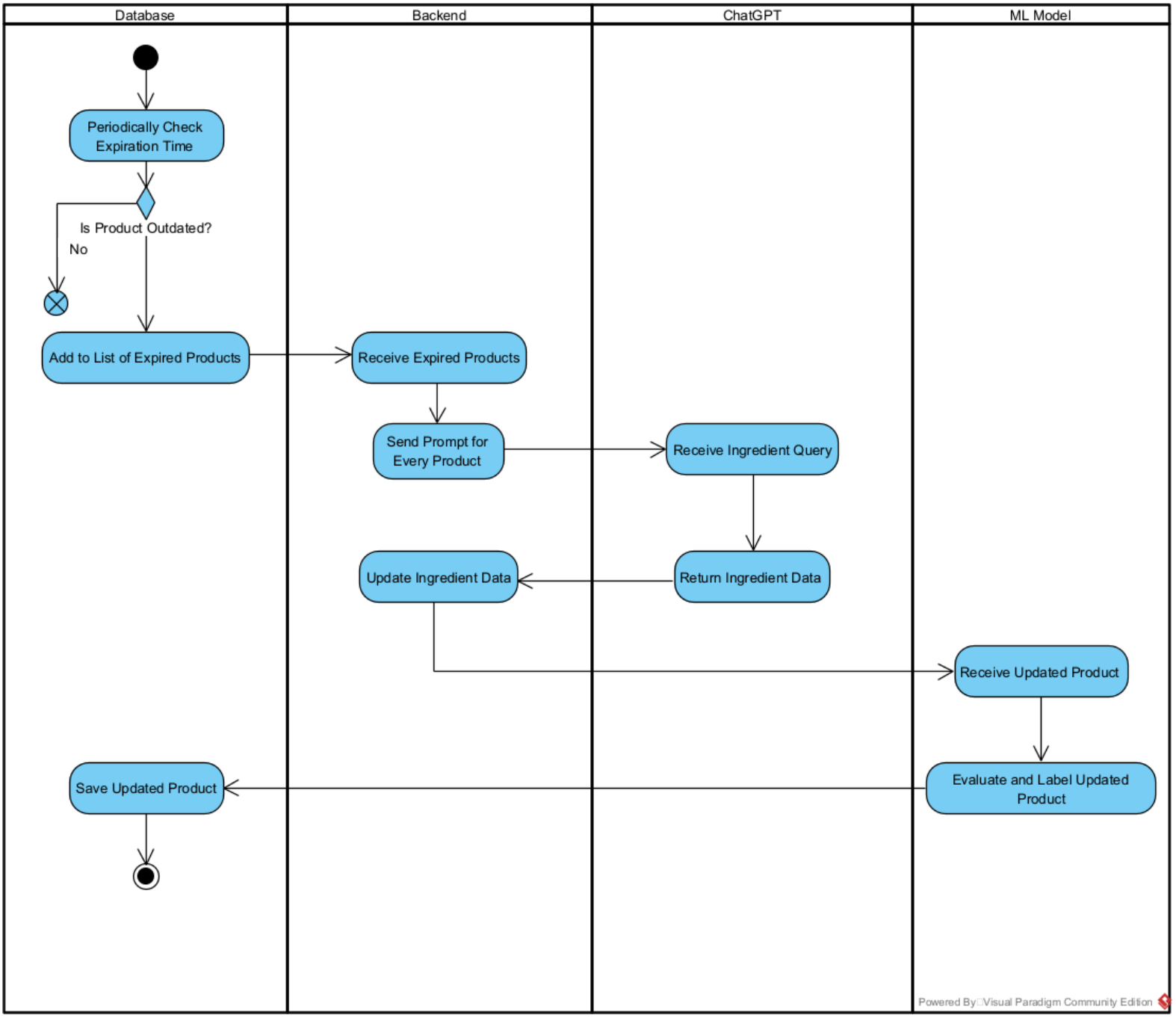

Figure 16 Update Ingredients of Skin Care Products' Activity Diagram

3. ChatGPT

6. Receive Ingredient Query:
   ChatGPT processes the query to retrieve the latest ingredient information for the product.
7. Return Ingredient Data:
   The updated ingredient information is sent back to the backend.

4. Backend (continued)

8. Update Ingredient Data:
   The backend updates the product's ingredient information with the data returned by ChatGPT.
9. Send Updated Product to ML Model:
   The backend sends the updated product information to the machine learning model for classification.

5. ML Model

10. Receive Updated Product:
    The ML Model receives the updated ingredient data.
11. Evaluate and Label Updated Product:
    The model analyzes the ingredients and assigns a suitability label based on skin type compatibility.
12. Return Classification:
    The labeled product is sent back to the backend.

6. Backend (Final)

13. Save Updated Product:
    The backend saves the updated product information, including the new ingredient list and suitability labels, back to the database.

End State

The process ensures that expired product data is refreshed, reclassified, and stored in the database, maintaining the accuracy and reliability of the system's recommendations.

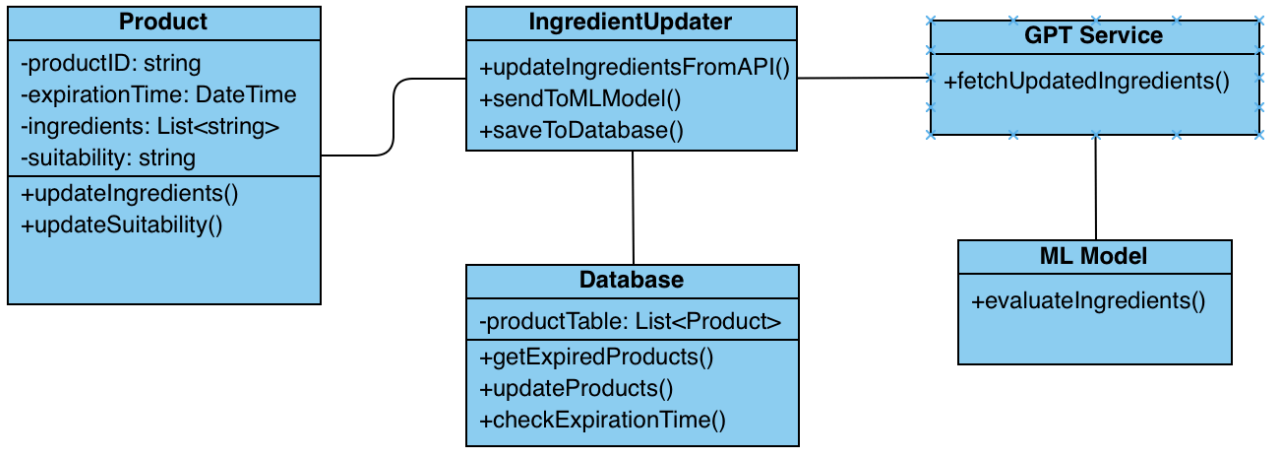

Figure 17 Update Ingredients of Skin Care Products' Class Diagram

This *Figure 17* Class Diagram represents the architecture and interactions within an ingredient update and suitability evaluation system for skincare products. The system ensures the product database remains accurate and relevant by periodically refreshing ingredient data using GPT APIs and re-evaluating product suitability with a machine learning model. It facilitates efficient updates to product information and suitability classifications.

1.Product

- **Attributes:**
  - productID: int – Unique identifier for the product.
  - expirationTime: Date – Expiration date of the product's ingredient data.
  - ingredients: List<String> – Current list of ingredients.
  - suitability: String – Suitability classification for specific skin types.
- **Methods:**
  - updateIngredients() – Updates the product's ingredient list.
  - updateSuitability() – Updates the suitability classification.

2.IngredientUpdater

- **Methods:**
  - updateIngredientsFromAPI() – Calls GPT API to retrieve updated ingredient data for a product.
  - sendToMLModel(productID: int, ingredients: List<String>): String – Sends the updated ingredient data to the ML model and retrieves suitability classification.
  - saveToDatabase(product: Product) – Saves the updated product details to the database.

3.GPTService

- **Methods:**
  - fetchUpdatedIngredients(): – Provides updated ingredient data for a product via GPT API.

4.MLModel

- **Methods:**
  - evaluateIngredients(): String – Evaluates the suitability of the ingredients for specific skin types and returns a suitability classification.

5.Database

- **Attributes:**
  - productTable: List<Product> – A list of all products in the database.
- **Methods:**
  - getExpiredProducts(): List<Product> – Fetches products with expired ingredient data.
  - updateProduct(product: Product) – Updates the product details in the database.
  - checkExpirationTime(): – Checks if the ingredient data has expired.

2.9. List All Products

This feature provides users with a comprehensive view of all skincare products available in the database. It also allows users to filter products by specific categories to quickly locate items of interest.

Purpose

The purpose of this function is to enhance user experience by:

- Offering a centralized location for users to view all products.
- Allowing filtering by product categories for targeted searches.

Input

- **User Action**:
  - View all products: No input required; all products are displayed.
  - Apply a filter: The user selects a category (moisturizer, cleanser, toner, serum, sunscreen) from a dropdown or filter panel.

Output

- A list of all products or filtered products based on the selected category.
- Each product is displayed in a card format containing:
  - Product Name
  - Category

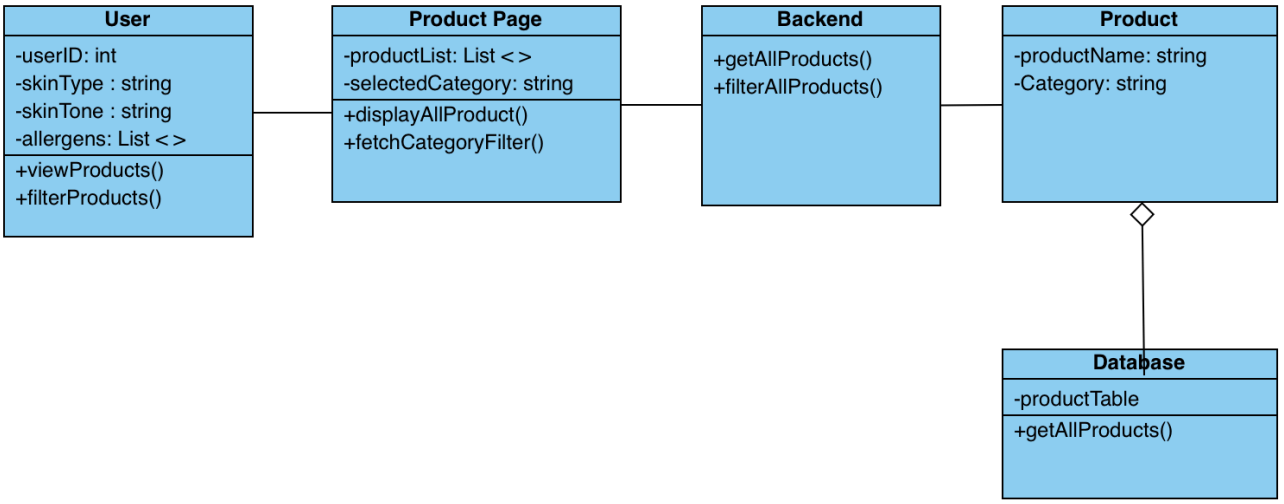

Figure 18 List All Products Class Diagram

This *Figure 18* Class Diagram represents the architecture and interactions within a product display system that shows all available skincare products on a single page. The system enables users to browse and filter products by category, skin type, and other attributes.

Classes and Attributes:

1. User:

- **Attributes:**
  - userID: int – A unique identifier for the user.
  - skinType: string – The user's skin type.
  - skinTone: string – The user's skin tone, relevant for products like sunscreen.
  - allergens: List<> – A list of allergens that the user should avoid.
- **Methods:**
  - viewProducts() – Enables the user to view all products available in the system.
  - filterProducts() – Filters products based on user attributes like skin type, tone, or allergens.

2. Product Page:

- **Attributes:**
  - productList: List<> – A list of all products displayed on the page.
  - selectedCategory: string – The category selected by the user (e.g., cleanser, toner).
- **Methods:**
  - displayAllProduct() – Displays all products available, irrespective of category or skin type.
  - fetchCategoryFilter() – Fetch filters the products by category and updates the display.

3.Backend:

- **Methods:**
  - getAllProducts() - Get all products from database with GET method.
  - filterAllProducst() - Filter all products by category.

4. Product:

- **Attributes:**
  - productName: string – The name of the skincare product.
  - Category: string – The category the product falls under.

5. Database:

- **Attributes:**
  - productTable – The main table storing product information.
- **Methods:**
  - getAllProducts() – Retrieves all products from the database and sends them to the product page for display.

Class Relationships:

- The **User** class interacts with the **Product Page** to view the list of products.
- The **Product Page** pulls product information from the **Database** and displays it to the user.
- The **Product** class holds information on individual items, including their name and categories.
- The **Backend** class got all products and filtered the products by category.
- The **Database** is responsible for storing all product data, which can be queried and displayed on the product page.

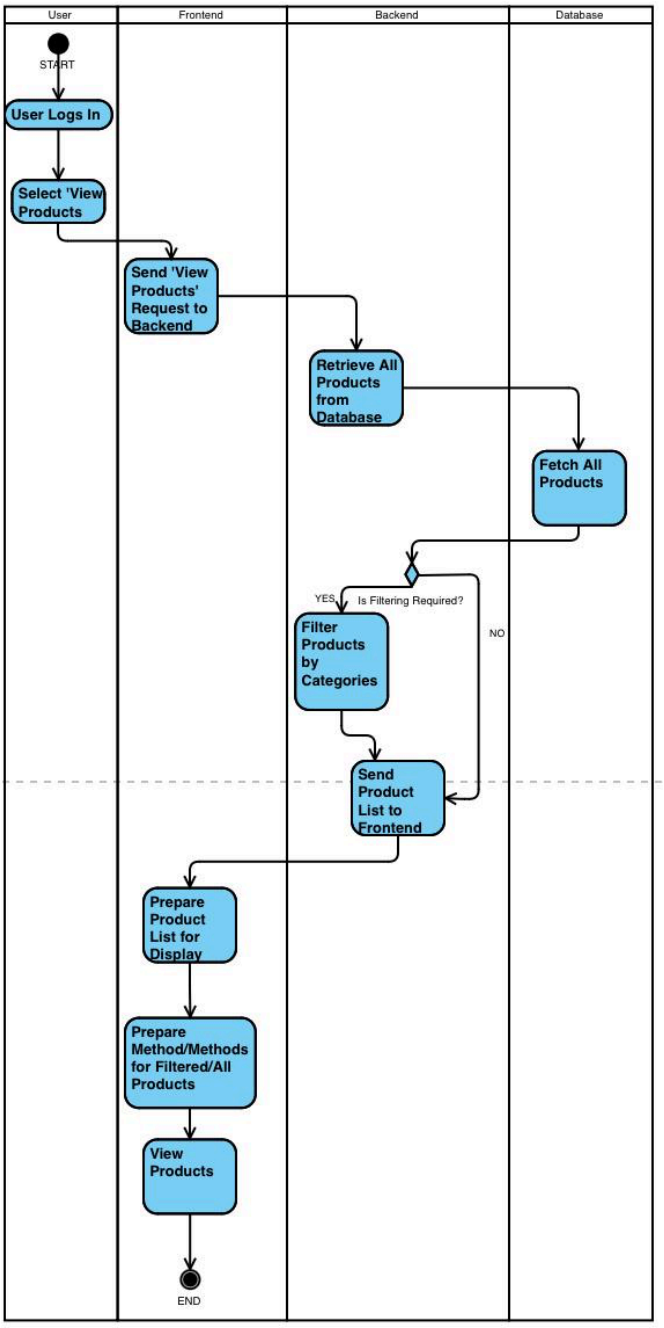

Figure 19 List All Products Activity Diagram

Figure 19 shows that the system allows users to view products by retrieving data from the database through the backend. Users can apply filters to refine the product list, which is then displayed on the frontend. This process ensures a seamless and interactive browsing experience.

1. User

- **User Logs In**:
  - The process begins with the user logging into the system.
- **Select "View Products"**:
  - After logging in, the user chooses the "View Products" option to see the available products.

2. Frontend

- **Send "View Products" Request to Backend**:
  - The frontend sends a request to the backend to retrieve the list of products. This is typically achieved via an API call.

3. Backend

- **Retrieve All Products from Database**:
  - The backend receives the request and queries the database to fetch all product information.

4. Database

- **Fetch All Products**:
  - The database fetches all product entries and sends them to the backend for further processing.

5. Backend

- **Is Filtering Required?**:
  - The backend checks whether the user has applied any filters (e.g., category, price range, etc.).
  - **If Filtering is Required**:
    - The backend filters the product list according to the selected criteria.
  - **If No Filtering is Required**:
    - The backend skips this step and proceeds with the full list.
- **Send Product List to Frontend**:
  - The final list of products, whether filtered or unfiltered, is sent back to the frontend.

6. Frontend

- **Prepare Product List for Display**:
  - The frontend prepares the received product list for user display. This includes formatting and arranging the products visually.
- **Prepare Methods for Filtered/All Products**:
  - The frontend ensures that filtering options are functional, allowing users to interact with the displayed list as needed.

7. User

- **View Products**:
  - Finally, the user views the list of products displayed on the frontend. They can further refine the list using filters if desired.

3. Non-Functional Requirements

3.1. Development Environment

Machine Learning

- **Languages:** Python
- **IDE:** PyCharm
- **Algorithms:** Random Forest, Logistic Regression(Softmax), SHAP

Frontend

- **Languages**: HTML, CSS, JavaScript,TypeScript
- **Frameworks/Libraries**:
    - React (Node.js for SSR)
    - Angular
    - Bootstrap

Backend

Backend will be written in Python. OpenAI's API is accessible via HTTPS requests, so we can use any programming language that supports making HTTP requests, including Python.

- **Languages**: Python
- **Frameworks**:
    - Flask
- **APIs**: REST, ChatGPT

Database

PostgreSQL will be the sole database used to manage structured data, including user profiles, product information, and skin type test results. Its robust architecture, scalability, and support for complex queries ensure data consistency, integrity, and high performance.

- **Database Management System (DBMS):** PostgreSQL

Backup Strategy:

- Weekly full database backups with daily incremental backups to ensure data security and recovery.

DevOps and Hosting

- GitHub

3.2. Security

- SSL for secure data transfer.
- Passwords stored with hashing (bcrypt).
- API keys stored securely (environment variables).
- Role-based access control (RBAC) to restrict sensitive data access.
- Data encryption at rest and during transmission to safeguard user information.

3.3. Scalability

Scalable database and API architecture to handle increased user data.

3.4. Testing

Unit, integration, and user acceptance tests for all features.

3.5. Data Privacy

Ensure user data is encrypted and stored securely.

4. References

1. Kumar, K., Sinha, V., Sharma, A., Monicashree, M., Vandana, M.L., Vijay Krishna, B.S.: AIassisted college recommendation system. In: Intelligent Sustainable Systems: Proceedings of ICISS 2022, pp. 141–150. Springer Nature Singapore, Singapore (2022) DOI:10.1007/978-981-99-9018-4_28
2. Saidah, S., Fuadah, Y.N., Alia, F., Ibrahim, N., Magdalena, R., Rizal, S.: Facial skin type classification based on microscopic images using convolutional neural network (CNN). In: Proceedings of the 1st International Conference on Electronics, Biomedical Engineering, and Health Informatics: ICEBEHI 2020, 8–9 Oct, Surabaya, Indonesia, pp. 75–83. Springer Singapore (2021) DOI: 10.1007/978-981-33-6926-9_7
3. Vinutha, M., Dayananda, R.B., Kamath, A.: Personalized skincare product recommendation system using content-based machine learning. In: 2024 4th International Conference on Intelligent Technologies (CONIT), pp. 1–9. IEEE, Bangalore, India (2024). DOI: 10.1109/CONIT61985.2024.10626458
4. Jadhav, S., Memane, D., Supekar, K., Shinde, S., & Jadhav, T. (2021). *A Personalized Skincare Recommendation System Using Machine Learning*. *CIENCIENGN*, 11(1). DOI: 10.52783/cienceng.v11i1.127
5. Lokesh, B., Devarakonda, A., Srinivas, G., & Naik, N. K. (2024). *Intelligent Facial Skin Care Recommendation System*. *AFJBS*, 6(Si2), 1822-1830.
   DOI: 10.33472/AFJBS.6.Si2.2024.1822-1830