YAŞAR UNIVERSITY
FACULTY OF ENGINEERING
DEPARTMENT OF COMPUTER ENGINEERING

COMP4920 Senior Design Project 2, Spring 2024
Advisor: Assoc. Prof. Dr. Ömer ÇETİN

GLOWG: Personalized Skin Care Powered by AI
Final Report

02.06.2025

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

PLAGIARISM STATEMENT

This report was written by the group members and in our own words, except for quotations from published and unpublished sources which are clearly indicated and acknowledged as such. We are conscious that the incorporation of material from other works or a paraphrase of such material without acknowledgement will be treated as plagiarism according to the University Regulations. The source of any picture, graph, map or other illustration is also indicated, as is the source, published or unpublished, of any material not resulting from our own experimentation, observation or specimen collecting.

Project Group Members:

Name, Lastname	Student Number	Signature	Date
Ceren Sude Yetim	21070001045		02.06.2025
İrem Demir	20070001029		02.06.2025
Ece Topuz	21070001057		02.06.2025
Gizem Tanış	20070001047		02.06.2025

Project Advisors:

Name, Lastname	Department	Signature	Date
Ömer Çetin	Computer Engineering		03.06.2025

ACKNOWLEDGEMENTS

We would like to express our sincere gratitude to all those who supported and contributed to the success of the *Glow Genie* project.

Firstly, we extend our deepest thanks to our project advisor, Assoc. Prof. Dr. Ömer ÇETİN, whose guidance, expertise, and valuable feedback played a crucial role in shaping the direction of this project. Their support was instrumental in overcoming challenges.

We would also like to acknowledge the collaborative efforts of our team members, İrem Demir, Ceren Sude Yetim, Ece Topuz, Gizem Tanış, whose dedication, hard work, and creative ideas were key to shaping the direction of this project. Each member's commitment and contribution were essential in achieving the project goals.

KEYWORDS

Skin Care
Product Suitability Evaluation
Product Recommendation System
Machine Learning
DeepSeek Integration
Dataset Preprocessing
Model Training
Ingredient-Based Classification
Data Analysis
Feature Engineering
Web-Based Application
User Interface Design
System Testing and Validation

ABSTRACT

The GlowGenie project, developed as part of the Senior Design Project 1 (COMP 4910) at Yaşar University, seeks to transform the skincare routine creation process by introducing a sophisticated web-based recommendation system. GlowGenie provides personalized skin care product suggestions tailored to users' unique skin types, tones, and preferences, addressing the challenge of selecting suitable products from an overwhelming market. By integrating advanced machine learning algorithms and GPT-based ingredient analysis, the project ensures that users receive scientifically informed and user-specific recommendations.

During the Fall 2024 semester, the project team concentrated on the foundational planning, research, and design phases. This included extensive research on skin types and product compatibility, designing a questionnaire to predict skin types, and creating comprehensive Requirements Specification and Design Specification Documents. Additionally, the team developed UML diagrams, database structures, and functional prototypes, focusing on a scalable, secure, and user-friendly system architecture.

GlowGenie introduces a novel approach to skincare by integrating a real-time database update system and a machine learning model to evaluate and classify products based on ingredient suitability. By providing insightful feedback on why a product suits or does not suit a user’s skin type, GlowGenie aims to educate users while improving their skincare journey. This project highlights the potential of AI-driven technology to enhance personalized healthcare experiences and simplify complex decision-making processes in the beauty and wellness industry.

ÖZET

GlowGenie projesi, Yaşar Üniversitesi'nde Senior Design Project 1 (COMP 4910) kapsamında geliştirilmiş olup, kişiye özel cilt bakım rutini oluşturma sürecini dönüştürmeyi amaçlayan, gelişmiş bir web tabanlı öneri sistemidir. GlowGenie, kullanıcıların cilt tipi, cilt tonu ve tercihlerine göre özel olarak uyarlanmış cilt bakım ürünü önerileri sunarak, pazardaki ürün çeşitliliği arasında uygun ürün seçimi sorununu çözmeyi hedefler. Proje, makine öğrenimi algoritmalarını ve GPT tabanlı içerik analizini entegre ederek, bilimsel verilere dayalı ve kullanıcı odaklı öneriler sunar.

2024 Güz dönemi boyunca proje ekibi, planlama, araştırma ve tasarım aşamalarına odaklanmıştır. Bu süreçte cilt tipleri ve ürün uygunluğu üzerine kapsamlı araştırmalar yapılmış, cilt tipini tahmin etmeye yönelik bir anket tasarlanmış ve kapsamlı Gereksinim Şartnamesi ile Tasarım Şartnamesi belgeleri hazırlanmıştır. Ayrıca, ekip UML diyagramları, veri tabanı yapıları ve işlevsel prototipler geliştirerek ölçeklenebilir, güvenli ve kullanıcı dostu bir sistem mimarisine odaklanmıştır.

GlowGenie, ürünlerin içerik uygunluğuna göre değerlendirilip sınıflandırılması için gerçek zamanlı bir veri tabanı güncelleme sistemi ve makine öğrenimi modeli entegre ederek yenilikçi bir yaklaşım sunmaktadır. Bir ürünün neden kullanıcı cilt tipi için uygun veya uygun olmadığını açıklayan geri bildirimler sağlayarak, kullanıcıları bilgilendirmeyi ve cilt bakım sürecini geliştirmeyi amaçlamaktadır. GlowGenie projesi, AI destekli teknolojinin kişiselleştirilmiş sağlık ve güzellik deneyimlerini geliştirme ve karmaşık karar alma süreçlerini basitleştirme potansiyelini gözler önüne sermektedir.

TABLE OF CONTENTS

PLAGIARISM STATEMENT ii
ACKNOWLEDGEMENTS iii
KEYWORDS iv
ABSTRACT v
ÖZET vi
TABLE OF CONTENTS vii
LIST OF FIGURES viii
LIST OF TABLES ix
LIST OF ACRONYMS/ABBREVIATIONS x
1. INTRODUCTION 1
1.1. Description of the Problem 1
1.2. Project Goal 2
1.3. Project Outputs/Deliverables 3
2. SURVEY OF RELATED WORK 4
3. REQUIREMENTS 5
4. DESIGN 5
4.1. High Level Design 5
4.2. Detailed Design 5
4.3. Realistic Restrictions and Conditions in the Design 7
5. IMPLEMENTATION AND TESTS 7
5.1. Implementation of the Product 8
5.2. Tests and Results of Tests 8
6. PROJECT MANAGEMENT 9
6.1. Project Plan 9
6.2. Project Effort/Manpower 10
6.3. Project Cost Analysis 11
7. CONCLUSIONS 12
7.1. Summary 12
7.2. Benefits of the Project 13
7.3. Future Work 14
REFERENCES 14
APPENDICES 15
APPENDIX A: REQUIREMENTS SPECIFICATION DOCUMENT 16
APPENDIX B: DESIGN SPECIFICATION DOCUMENT 59
APPENDIX C: PRODUCT MANUAL 73
APPENDIX D: PROJECT MANAGEMENT DOCUMENTS 85
APPENDIX D1: PROJECT PLAN 85
APPENDIX D3: PROJECT EFFORT LOGS FOR EACH TEAM MEMBER- 87
APPENDIX D3: PROJECT EFFORT LOG- CONSOLIDATED 97

LIST OF FIGURES

RSD Report.....13
Figure 1: Registration Use Case Diagram 7
Figure 2: Registration Activity Diagram..... 9
Figure 3: Registration Class Diagram.....10
Figure 4: LogIn Use Case Diagram..... 12
Figure 5: Log In Activity Diagram..... 14
Figure 6: Login Class Diagram.....16
Figure 7: Update Profile Information Class Diagram.....18
Figure 8: Update Profile Information Use Case Diagram.....20
Figure 9: Skin Type Test Use Case Diagram..... 23
Figure 10: Skin Type Test Activity Diagram..... 24
Figure 11: Skin Type Prediction Class Diagram.....25
Figure 12: Product Suitability Feedback System Activity Diagram.....28
Figure 13: Product Suitability Feedback Class Diagram..... 30
Figure 14: Generate Product Recommendations Activity Diagram..... 33
Figure 15: Generate Product Recommendations Class Diagram.....34
Figure 16: Update Ingredients of Skin Care Products' Activity Diagram.....36
Figure 17: Update Ingredients of Skin Care Products' Class Diagram..... 37
Figure 18: List All Products Class Diagram..... 39
Figure 19: List All Products Activity Diagram.....40
DSD Report.....56
Figure 20: Component Diagram of Glow Genie Software System..... 5
Figure 21: Registration Interface with error messages..... 7
Figure 22: Login Interface from GUI Demo.....8
Figure 23: Update Profile Interface with error messages from GUI Demo.....8
Figure 24: Skin type test and prediction interface from GUI Demo.....9
Figure 25: Product suitability interface with given feedback from GUI Demo.....10
Figure 26: Product recommendation interface from GUI Demo..... 11

LIST OF TABLES

Table 1: Project Plan 86
Table 2: Project Effort Log Gizem Tanış..... 88
Table 3: Project Effort Log Ece Topuz.....91
Table 4: Project Effort Log İrem Demir.....93
Table 5: Project Effort Log Ceren Sude Yetim..... 95
Table 6: Project Effort Log Consolidated…….....98

LIST OF ACRONYMS/ABBREVIATIONS

UML: Unified Modeling Language
ML: Machine Learning
API: Application Programming Interface
REST: Representational State Transfer
SQL: Structured Query Language

1. INTRODUCTION

1.1. Description of the Problem

The beauty and skincare industry has seen tremendous growth over the years, offering an abundance of products tailored to various skin types and concerns. However, this diversity has also brought significant challenges for consumers. Selecting the right skincare products, which match individual skin types and address specific needs, has become a daunting task for many. Key challenges include identifying one's skin type accurately, understanding product ingredients, and evaluating the compatibility of these products with their skin.

One of the primary hurdles lies in the inability of many users to accurately determine their skin type. Misjudging skin type can lead to the selection of unsuitable products, resulting in adverse effects such as dryness, excessive oil production, irritation, or acne. Beyond physical discomfort, these negative experiences contribute to financial losses and erode consumer trust in skincare routines and products.

Compounding this issue is the overwhelming variety of skincare products available on the market. Each product often boasts unique ingredients and claims, leaving users uncertain about which options will be most effective or safe. For individuals with specific allergies or sensitivities, this task becomes even more complex as they navigate ingredient lists that are often difficult to interpret without expert knowledge.

Modern lifestyles further exacerbate these challenges. Many individuals lack the time to conduct thorough research on skincare products or to stay informed about the latest formulations. Additionally, product ingredients are frequently updated by manufacturers, and keeping track of these changes can be a daunting task even for dedicated skincare enthusiasts. Traditional methods, such as relying on general product reviews or recommendations from non-specialized sources, are often insufficient to address these individual-specific needs.

Another significant gap exists in the lack of transparency regarding ingredient effects. Many existing platforms or recommendations fail to provide detailed explanations for why a product is deemed suitable or unsuitable for a particular skin type. This lack of clarity forces users to rely on trial-and-error methods, leading to frustration and wasted resources.

GlowGenie was conceptualized to address these pressing issues. The platform is designed to offer a personalized, AI-powered solution that simplifies the process of skincare product selection. By leveraging machine learning models, GlowGenie aims to help users accurately determine their skin type through a guided questionnaire. Once the user’s skin type is identified, the platform provides tailored product recommendations that factor in individual skin profiles, specific allergens, and preferences.

What sets GlowGenie apart is its dynamic approach to ingredient analysis. The platform periodically updates product ingredient databases using advanced AI tools like GPT, ensuring that recommendations remain relevant and reliable. Users receive detailed feedback on the suitability of product ingredients, with clear visual indicators highlighting compatible and incompatible elements. This level of transparency empowers users to make informed choices confidently.

Furthermore, GlowGenie aims to bridge the gap between technical complexity and user

experience by offering an intuitive interface. Users can filter products by categories, such as cleansers, moisturizers, or sunscreens, and view recommendations tailored to their unique needs. The ability to update profiles with changes in skin type, tone, or new allergens ensures that the system remains responsive to evolving user requirements.

For a more in-depth discussion of the challenges faced by skincare users and the comprehensive solutions GlowGenie proposes, please refer to the GlowGenie Requirements Specification Document (RSD). This document provides detailed insights into the underlying problems and the strategies employed to overcome them.

1.2. Project Goal

The primary goal of the GlowGenie project is to provide a cutting-edge, user-friendly, and AI-powered skincare platform that empowers individuals to make informed decisions about their skincare. By addressing the challenges outlined in the GlowGenie Requirements Specification Document (RSD) and implementing solutions detailed in the Detailed Software Design (DSD) document, the project seeks to transform the way users interact with skincare products and routines. Below are the detailed goals of the project:

1. **Accurate Skin Type Identification**:
   ○ Develop a robust AI-based skin type assessment tool to address the common issue of misidentifying skin types.
   ○ Use an interactive questionnaire designed to provide precise results, stored securely in the user's profile for continuous customization.
   ○ This functionality aligns with the RSD's emphasis on providing foundational solutions to help users begin their skincare journey effectively.

2. **Personalized Product Recommendations**:
   ○ Create a recommendation system that offers tailored product suggestions based on individual skin types, tones, allergies, and preferences.
   ○ Implement content-based filtering using machine learning models, as described in the DSD, to ensure high relevance and precision in recommendations. ○ Allow users to filter by product categories such as cleansers, moisturizers, sunscreens, and more, enhancing usability.

3. **Dynamic Ingredient Analysis and Updates**:
   ○ Integrate a dynamic product ingredient database powered by GPT-based AI tools to ensure the platform adapts to changes in formulations.
   ○ Provide users with transparent feedback on ingredient compatibility, including visual indicators for suitable or unsuitable products.
   ○ This feature aligns with the RSD's focus on addressing the evolving needs of users by maintaining up-to-date product information.

4. **Enhanced User Interaction and Customization**:
   ○ Develop an intuitive interface that simplifies navigation and makes complex skincare decisions accessible to non-expert users.
   ○ Enable users to update their profiles dynamically to reflect changes in their skin type, tone, or newly discovered allergens, as highlighted in the DSD.
5. **Compliance with Quality Standards**:
   ○ Ensure that the platform adheres to ISO/IEC 9001 software quality standards, as described in the DSD, emphasizing usability, security, and performance efficiency.
   ○ Use Unified Modeling Language (UML) diagrams for system design, ensuring clarity and

adherence to organizational standards.

6. **Seamless Integration of AI and User Data**:

   ○ Leverage machine learning models to continuously improve the accuracy of skin type assessments and product recommendations.
   ○ Utilize user data responsibly, with a strong emphasis on privacy and security.

By achieving these goals, GlowGenie aims to become a reliable and innovative platform that not only addresses the immediate concerns of its users but also evolves with their needs. The combination of state-of-the-art technology, transparent recommendations, and a personalized approach ensures that users can confidently navigate their skincare journeys. For a comprehensive overview of these functionalities and their technical implementations, refer to the DSD document.

1.3. Project Outputs/Deliverables

In COMP4910, we began by formalizing the scope and objectives of the GlowG project through the preparation of the **Project Assignment Form (PAF)**. Following this, we developed the **Requirements Specifications Document (RSD v1.0)**, which outlined the system's functional and non-functional requirements in a textual format. To ensure the document remained aligned with the project's evolution, we updated it as **RSD v2.0**, incorporating **UML diagrams** to visually represent the requirements. This version was also included as **Appendix A** in the **Final Report**, which summarized our methodologies, findings, and project progress.

The **Design Specifications Document (DSD v1.0)** was also created during this phase, providing a high-level system design using UML diagrams, which were included as **Appendix B** in the Final Report. To communicate the project's concept effectively, we prepared the **first drafts** of the **Project Poster** and **Project Presentation**, which highlighted the key features and objectives of GlowG. These drafts were integrated into a **preliminary version of the Project Website**, designed to showcase our work in an accessible format. Finally, **Peer Evaluation Forms** were completed by all team members to assess contributions and ensure transparency in the collaborative effort.

For COMP4920, the expected deliverables include the **Final Version of the Requirements Specifications Document**, which will reflect all updates and refinements based on the progress made. Similarly, the **Design Specifications Document** will be finalized to detail the complete system architecture. The **Final Report**, together with all appendices, will provide a comprehensive overview of the project's journey, from inception to completion.

We aim to deliver a **polished version of the Project Poster and Presentation**, catering to both academic and professional audiences. The **Project Website** will also be finalized, serving as a central hub to demonstrate GlowG's functionality, technical details, and potential applications. In addition, a **Project Manual** will be prepared to guide users and developers in utilizing and maintaining the system effectively.

Lastly, the **Project Source Code**, comprising the fully implemented GlowG system, will be delivered as a significant outcome. This code will reflect the culmination of our efforts, showcasing a robust and functional solution aligned with the project's objectives.

As part of COMP4920, we foresee completing and delivering our **AI-powered personalized**

skincare project, which was meticulously planned during COMP4910. This will mark the realization of our vision for GlowG, providing an innovative and user-centric solution.

2. SURVEY OF RELATED WORK

This section explores both academic research and commercial solutions in the field of personalized skincare systems. By reviewing relevant academic papers and analyzing successful commercial products, we aim to integrate the best practices and innovations into the development of the Glow Genie system, ensuring it provides effective, personalized skincare recommendations for users.

2.1 Academic Papers

Several academic papers have contributed to the development of personalized skincare systems by applying data mining techniques, machine learning algorithms, and dermatological expertise to classify skin types and recommend skincare products. Below are some key academic papers relevant to the field.

The academic papers we drew inspiration from have played a significant role in the development of personalized skincare systems. For instance, in the paper "A Personalized Skincare Recommendation System Using Machine Learning" by Supriya Jadhav and colleagues [1], the potential of machine learning algorithms for providing personalized skincare recommendations based on user profiles, such as skin type and concerns, was demonstrated. This approach, which analyzes user data to provide tailored skincare suggestions, inspired us to build a similar system that leverages user information to offer personalized recommendations. Additionally, the "Intelligent Facial Skin Care Recommendation System" by B. Lokesh and colleagues, which integrates Convolutional Neural Networks (CNN) [2] to classify skin types and offer personalized skincare advice based on user-uploaded skin images, influenced us to adopt individualized skincare solutions.

2.2 Commercial Solutions/Products

In addition to academic research, we also analyzed commercial solutions in the skincare industry to guide our system's development. These real-world examples provided practical insights into how personalized skincare can be delivered effectively.

Commercial solutions have also provided valuable insights for our system's development. ***Proactiv+***, for example, uses an online questionnaire to identify users' skin concerns and recommends a specific set of acne treatment products based on individual needs. Their approach, combining product knowledge with personalized skin assessments, motivated us to create a similar system focused on addressing users' specific skin concerns. ***Curology***, on the other hand, offers customized skincare treatments based on skin type and issues like acne and aging, using telemedicine consultations and dermatologist-reviewed photos of users' skin. Drawing from Curology's model, we aim to incorporate telemedicine features into our system, allowing for more precise skin assessments and personalized skincare formulas for our users.

3. REQUIREMENTS

The project began with the creation of a Project Assignment Form (PAF), which included details like a project code, title, team information, and a summary of the project or product. This was followed by the development of the first version of the Requirements Specification Document (RSD 1.0), which was written in a structured text format. Subsequently, an updated version, RSD 2.0, was generated. This version enhanced the initial requirements outlined in RSD 1.0 and presented them using the Unified Modeling Language (UML) notation.

The definitive requirements for our COMP4910 project are detailed in Appendix A, under the title "Requirements Specifications Document, version 2.0.

4. DESIGN

The design section outlines the architecture and technical framework for the Glow Genie project. It details the system's high-level design, which ensures modularity, scalability, and maintainability, as well as the conditions and restrictions that were considered to ensure the system meets the project's objectives while balancing feasibility and functionality.

4.1. High Level Design

The high-level design of the GlowGenie system utilizes a **Layered Architecture** to ensure modularity, maintainability, and scalability. This architecture separates the system into three key layers: the **Presentation Layer**, which handles user interactions through a dynamic and responsive React.js interface; the **Application Layer**, which processes business logic, integrates with machine learning models, and facilitates API communication; and the **Data Layer**, which manages persistent storage using PostgreSQL for efficient and structured data handling.

This architecture was selected because it promotes a clear separation of concerns, enabling ease of future development, adaptability for new features, and enhanced maintainability of the codebase. Detailed specifications for the high-level design can be found in **Appendix B: Design Specifications Document**, specifically in sections B.3.1 (**GLOWG Software System Architecture**).

4.2. Detailed Design

The GlowGenie detailed design can be distilled into four main areas:

1. **Backend Control (AppController)**
 The AppController is the single entry point for all API requests. It defines and routes endpoints for user registration, login, profile updates, skin testing, product compatibility checks, and recommendations. Rather than embedding any heavy logic, it offloads work to dedicated services (e.g., UserService, SkinTestService, RecommendationService, OpenAIIngredientAnalyzer). In practice, this means that AppController only needs to know which service method to invoke, keeping the controller lean and focused on validating inputs, managing sessions/authentication, and formatting responses. For example, when a user submits their skin questionnaire, AppController.take_skin_test() simply hands off the raw answers to the ML model and then stores the returned skin type. Likewise,

get_recommendations() merges the user's stored profile, ML filters, and GPT-based ingredient insights by calling the appropriate helper classes—nothing more, nothing less.

2. **Machine Learning Subsystem (Skin Type & Product Suitability)**

○ **ML-SkinTypePredictor**: Uses a linear regression model to translate questionnaire answers into one of the four skin types. The design is straightforward—load the training data, preprocess, fit a linear regression, save the model, and later load it for predictions. Since the input features are neatly structured (9 multiple-choice questions), there's minimal data engineering involved. When new survey data arrives, the predictor simply runs it through the stored regression coefficients and returns "Dry", "Oily", "Combination", or "Normal".

○ **ML-ProductSuitabilityModel**: Builds a Random Forest classifier over binary ingredient vectors to estimate how suitable a product is for each skin type. Once the model is trained, it outputs raw class scores that are converted via softmax into percentage-based suitability for all four skin types. SHAP values are computed afterward to explain which ingredients push a product toward or away from each skin category. In other words, this model not only tells you "80% chance this serum is best for Dry skin" but also pinpoints that "Alcohol Denat." is dragging suitability down.

3. **Frontend Components (React Classes)**

○ **RegistrationFormComponent / LoginComponent / ProfileUpdateComponent**: These three classes handle user account flows—registration, login, password resets, and profile edits (skin type, tone, allergens). They validate fields in real time, enforce rate limits on verification codes, and display errors without any sugarcoating ("Passwords must contain at least one uppercase letter," not "You might want to add an uppercase").

○ **SkinTypeTestComponent**: Renders the 9-question skin quiz, checks that no question is skipped, submits the answers, and immediately shows the returned skin type. There's no fluff: unanswered questions trigger "Please fill in the mandatory areas," and once a type is returned, the UI updates instantly.

○ **ProductSuitabilityCheckComponent**: Lets users search or select a product, then invokes the backend to fetch the ingredient list. It uses GPT-based analysis to compare each ingredient against the user's profile (tone, allergens, skin type), then colors compatible ingredients green and incompatible ones red. No ambiguous "maybe safe" messages—every ingredient is labeled clearly.

○ **ProductRecommendationComponent / AllProductsListComponent**: The former fetches personalized recommendations category by category (e.g., cleanser, serum, sunscreen) and renders each product card with its name, ingredients, and suitability cues. The latter simply retrieves and displays every product in a grid, with filter controls (by category or "only suitable" flags). Clicking "View More" or filtering re-queries the backend. If you pick a product from recommendations, it's automatically added to a favorites list or to the routine.

4. **Database Subsystem (PostgreSQL via pgAdmin 4)**

All user-related and product-related data live in PostgreSQL tables. The core tables are users (with foreign keys to skintypes and skintones), allergens/userallergens (many-to-many), products (with a category_id FK), ingredients/productingredients (many-to-many), and

suitabilityfeedback for tracking how users respond to recommended items. Each table adheres to normalization to avoid redundant data; composite keys manage relationships cleanly (for example, a product's multiple ingredients).

○ **Key operations** include creating/verifying users, storing quiz results, listing/filtering products, updating ingredient sets, computing and saving SHAP explanations, and managing password-reset tokens.

○ **Backups**: A weekly full dump plus daily incremental snapshots are stored encrypted in the cloud, and monthly restore drills ensure everything actually works if the worst happens.

In short, the detailed design keeps each module laser-focused: the controller only routes, ML classes only learn/predict/explain, frontend classes only render and validate, and the database only stores/retrieves. This separation prevents any single component from becoming a tangled mess—which is exactly why you won't find any business logic inside AppController or raw SQL scattered in React components.

The detailed design is provided in **Appendix B: Design Specifications Document, v2.0, section 4**.

4.3. Realistic Restrictions and Conditions in the Design

In the design of the **Glow Genie** project, several restrictions and conditions were considered to ensure the system is functional and feasible within the project's scope. These include the following:

Data Security: The current design does not include advanced security measures like data encryption or multi-factor authentication, as the system's primary focus is on providing personalized skincare recommendations. However, security measures such as user authentication and basic data privacy will be included in the future phases of development.

Security and Authentication: While basic security measures are implemented, there are limitations in advanced security protocols. For instance, the system employs standard password enforcement rather than multifactor authentication, a decision driven by the need to balance security with user convenience.

User Experience: As the system is designed to be user-friendly, it incorporates simple input methods and provides real-time feedback. However, it does not offer a complex user interface with multiple language support, as this is outside the current scope.

System Performance: The system is designed to handle a reasonable load, with a focus on ensuring quick response times for user input. Performance optimizations such as caching is not included at this stage but will be considered during later implementation phases.

5. IMPLEMENTATION AND TESTS

5.1. Implementation of the Product

The implementation follows the same modular, layered architecture outlined in the design:

- **Backend (Flask/Python)**

All request handling sits in `AppController`, which simply routes incoming HTTP calls to specialized service modules (e.g., `AuthService`, `RecommendationService`, `OpenAIRoutes`). In practice, that means your endpoint definitions (register, login, fetch products, run ingredient analysis) live in a central controller, but the actual work—database queries, ML inferences, email sending—happens in those downstream services. Environment-specific settings (database URLs, API keys, secret tokens) are kept in `.env` and `config.py`, so you never accidentally check secrets into version control.

- **Machine Learning Subsystem**

You have two standalone model classes:

1. **SkinTypePredictor** – trained offline with linear regression based on questionnaire answers; once the model file is in place, the Flask app just loads it at startup and calls `predict()` on new inputs.

2. **ProductSuitabilityModel** – a Random Forest classifier that's also trained ahead of time on binary ingredient vectors; at runtime it's used to score how compatible each product is with each skin type. SHAP-based explanations are generated on demand, but the heavy lifting happens outside of request time.

- **Frontend (React.js)**

Everything in `src/` issues secure REST calls to those Flask endpoints. Components validate inputs in real time (so you can't submit an empty email or a weak password without immediate feedback), and they render results—skin type, recommendation cards, suitability colors—straight out of the backend's JSON responses. The React app is completely decoupled from database logic; it never knows how data is stored, only how to call "/take-skin-test" or "/get-recommendations."

- **Folder Structure & Version Control**

The repo is split into `/backend`, `/ml_models`, and `/frontend`, each with its own `README.md`, unit tests, and config files. Whenever you need to tweak database schemas or add a new endpoint, you know exactly where to go. All branches—feature, testing, production—live on GitHub, and CI checks run on every pull request to make sure nothing breaks before merging.

In short, this implementation keeps each layer—API, ML, UI—completely isolated. If you ever wanted to swap Flask for FastAPI or replace React with Vue, it'd be a matter of writing new adapters, not rewriting your entire codebase.

More detailed information on this implementation can be found in **APPENDIX C, Section 2 of the Product Manual**.

5.2. Tests and Results of Tests

Testing was organized in layers, covering everything from individual APIs and ML routines to the React UI and database interactions:

- **Backend/API Tests:** Each REST endpoint (registration, login, profile updates, product listing, recommendations) was exercised via Swagger. We poked both "happy path" inputs and bogus data to make sure validation kicked in. Whenever an endpoint wrote or read from Postgres, we double-checked via pgAdmin 4 that rows were actually inserted, updated, or fetched as expected.

- **Machine Learning Tests:** The two model classes (SkinTypePredictor and ProductSuitabilityModel) were trained and evaluated offline in PyCharm against held-out test sets. Accuracy metrics were verified, and SHAP outputs were sanity-checked to confirm that the explanations made sense. Once those models were wired into Flask, we hit their endpoints in Swagger again to ensure runtime behavior matched the offline results.

- **Frontend/Integration Tests:** The React components—login, profile update, skin test, product pages—were manually tested in a browser. Using DevTools, we monitored HTTP requests to ensure the UI sent correct payloads and displayed responses (or error messages) properly. Layout checks, form validation, and error-handling flows were all verified under normal usage.

- **Database Verification:** Beyond unit tests, we manually ran SQL queries after key operations—like submitting a skin test or updating a profile—to confirm that the data layer reflected the frontend action. This guaranteed there were no hidden "Oh, it only looks like it worked" bugs.

- **Known Gaps:** Password reset and email confirmation code paths exist only as placeholders; they haven't been fully exercised yet. Likewise, any admin-only features or internal data management tools are deferred to later iterations.

In short, we ran Swagger, PTAs (pen-and-paper technical audits), and eyeballed the React logs until our eyes hurt—covering most visible features end-to-end. For a deeper dive into each test case, script, and result, see **Appendix C, Section 3**.

6. PROJECT MANAGEMENT

6.1. Project Plan

Here is the number of weeks allocated to each task (Weeks 1–15):

1. **Project Planning & Final Requirements Confirmation**
   Weeks: 1, 2, 3, 4, 14, 15 → **6 weeks**

2. **System Implementation Based on Literature Findings**
   Weeks: 5, 6, 7 → **3 weeks**

3. **DeepSeek API Integration & Optimization**
   Weeks: 6, 7 → **2 weeks**

4. **Data Collection & Preprocessing Pipeline Development**
   Weeks: 1, 2, 8, 9 → **4 weeks**

5. **Machine Learning Model Training & Integration**
   Weeks: 9, 10, 11, 12, 13, 14, 15 → **7 weeks**

6. **UI Design Implementation & Frontend Development**
   Weeks: 7, 8, 9, 10, 11, 12, 13, 14, 15 → **9 weeks**

7. **Database Modeling & PostgreSQL Deployment**
   Weeks: 12, 13, 14 → **3 weeks**

8. **Flask Backend Development & RESTful API Integration**
   Weeks: 5, 6, 7, 8, 9, 10, 11, 12, 14 → **9 weeks**

9. **Final Presentation, Poster Design & Deliverables Preparation**
   Weeks: 14, 15 → **2 weeks**

Our Project Plan Table is in Appendix D1.

6.2. Project Effort/Manpower

Team Member	WoMan-Days	WoMan-Hours (≈ Man-Days × 8h)
İrem Demir	20.38	163.04 h
Ece Topuz	19.25	154.00 h
Ceren Sude Yetim	18.50	148.00 h
Gizem Tanış	18.75	150.00 h
Total (Team)	76.88	615.04 h

You can consult Appendix D2 for the month-by-month breakdown and Appendix D3 for the consolidated monthly totals and overall manpower figures.

6.3. Project Cost Analysis

Araç	Marka / Sağlayıcı	Model / Sürüm	Özellikler	Maliyet (TL)
MacBook Air (Geliştirme)	Apple	M1 (2022)	• Apple M1 çip• 8 GB birleştirilmiş RAM• 256 GB SSD	25.000 TL (≈ 1.350 USD)
Windows 11 Laptop	Acer	Swift 5 (Intel i7-1165G7)	• Intel Core i7-1165G7 (2.8 GHz)• 16 GB RAM• 1 TB SSD• Windows 11 64-bit	15.000 TL (≈ 800 USD)
PostgreSQL 15	PostgreSQL Global Dev.	15	• Açık kaynak, ilişkisel veritabanı yönetim sistemi	0 TL (Açık kaynak)
pgAdmin 4	pgAdmin Development	6.x	• Grafiksel PostgreSQL yönetim arayüzü	0 TL (Açık kaynak)
Python 3.10.6	Python Software Foundation	3.10.6	• Dinamik tipli programlama dili	0 TL (Açık kaynak)
Flask 2.2.2	Pallets Projects	2.2.2	• Hafif Python web çerçevesi (RESTful API için)	0 TL (Açık kaynak)
React.js	Meta (Facebook)	18.x	• Ön yüz geliştirme kütüphanesi	0 TL (Açık kaynak)
DeepSeek API (İçerik Analizi)	DeepSeek Inc.	Standard Abonelik	• Ürün içeriği çekme	0 TL (Ücretsiz)
Visual Studio Code	Microsoft	1.x	• Kod düzenleyici ve IDE	0 TL (Açık kaynak)
PyCharm Community	JetBrains	2022.3	• Python geliştirme ortamı (ML geliştirme için)	0 TL (Açık kaynak)
pip / Kütüphaneler	Python Paket Yöneticisi	–	• scikit-learn, SHAP, imbalanced-learn, vb.	0 TL (Açık kaynak)

Toplam Donanım Maliyeti (yaklaşık): 40.000 TL
Toplam Yazılım/Maaş Maliyeti (açık kaynak + araçlar): 0 TL

7. CONCLUSIONS

7.1. Summary

The Glow Genie project, initiated as part of the Senior Design Project, focuses on providing personalized skincare recommendations using DeepSeek API to help individuals choose suitable skincare products for their skin types. Below is a revised summary of what has been accomplished so far. We have delivered everything we originally promised; the only change was that DeepSeek is used solely for pulling and analyzing product ingredients (instead of ChatGPT).

1. **Planning:**
    - Identified the need for a personalized, AI-driven skincare system.
    - Outlined scope, objectives, and deliverables.
    - Completed detailed project planning, including timelines and responsibilities.
2. **Requirements:**
    - Developed a Requirements Specification Document (RSD) capturing all functional and non-functional needs.
    - Ensured alignment between stakeholder expectations and technical feasibility.
3. **Research and Analysis:**
    - Surveyed existing skincare recommendation platforms.
    - Evaluated various machine learning algorithms to inform our recommendation-engine design.
    - Researched how to integrate the DeepSeek API for ingredient retrieval and filtering; this replaces any earlier plan to use ChatGPT for the same purpose.
4. **System Design and Architecture:**
    - Produced a scalable, secure architecture that cleanly separates frontend, backend, ML, and database layers.
    - Confirmed that DeepSeek's API can be invoked by our backend's ingredient-analysis service without disrupting the overall design.
5. **Frontend and Backend Planning:**

- Defined component structure and data flows for the React-based UI.

- Mapped out RESTful endpoints in Flask to call DeepSeek for ingredient details; the actual recommendation logic remains in our ML services.

6. **Design Phase:**

    - Created a Design Specification Document (DSD) detailing technology choices (Flask, React.js, PostgreSQL, Python ML libraries) and security considerations.

    - Documented the switch from ChatGPT to DeepSeek for ingredient lookup; all other design decisions remain unchanged.
7. **Documentation:**

    - Compiled all project documentation (RSD, initial DSD drafts, planning artifacts).

    - Ensured that the shift to DeepSeek for ingredient retrieval is clearly noted, while recommendation still uses our trained ML models.
8. **Machine Learning Model Planning:**

- Identified and preprocessed a product dataset with binary ingredient features and known skin-type labels.

- Selected **Linear Regression** for predicting user skin type (nine-question survey) and **Random Forest** for classifying product suitability based on ingredients.

- Incorporated **SHAP** for post-hoc explainability to highlight key ingredients.

- Planned offline training, validation, and serialization of models for fast Flask inference, with periodic retraining as new data arrives.

7.2. Benefits of the Project

The Glow Genie project provides significant advantages not only for individual users but also for the environment. By offering personalized skincare recommendations, it helps users make better choices for their skin while minimizing environmental impact. These benefits align with the project's goal to enhance both user satisfaction and sustainability in the skincare industry.

Benefits for Users:

Glow Genie offers users a personalized skincare experience by recommending products specifically suited to their skin types. This ensures that users can make informed decisions, enhancing the effectiveness of their skincare routines while avoiding harmful products.

● **Personalized Skincare:** Glow Genie provides tailored skincare product recommendations based on individual skin types, ensuring users select products that are safe and effective for

their skin.
● **Informed Decision-Making:** The system helps users avoid harmful products and reduces the risk of skin irritation, enhancing overall skincare routines.
● **Cost Savings:** By suggesting only suitable products, it prevents unnecessary purchases, saving users money and time spent on trial and error.

Benefits for the Environment:

By promoting tailored product recommendations, Glow Genie helps reduce unnecessary product waste, contributing to a more sustainable skincare industry. The system encourages responsible consumption, supporting a shift toward more sustainable choices in the market.

● **Reduced Product Waste:** By helping users choose products that match their skin types, Glow Genie minimizes product returns and unused products, contributing to less waste.
● **Sustainable Consumption:** Encourages mindful consumption by reducing the likelihood of users purchasing products that do not fit their needs, promoting sustainability in the skincare industry.

7.3. Future Work

Machine Learning Algorithms: Enhance the recommendation engine by incorporating more advanced machine learning models. Optimize these models to provide more personalized and accurate recommendations, while enabling continuous learning from user feedback and data.

User Interface and Experience Enhancements: Refine the user interface based on user feedback to ensure it is intuitive, visually appealing, and mobile-responsive. Improve user experience (UX) through interactive elements and ensure ease of navigation and clarity in product recommendations.

REFERENCES

1. 1. 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
2. 2. 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
3. 3. 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
4. 4. 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
5. 5. 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

APPENDICES

APPENDIX A: REQUIREMENTS SPECIFICATION DOCUMENT

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

Revision History 1
Table of Contents 2
1. Introduction to GlowGenie 3
2. Functional Requirements 4
2.1. Main User Interface and Functions 4
2.2. Enter Registration Information 6
2.3. Log In 11
2.4. Update Profile 17
2.5. Skin Type Test 21
2.6. Product Suitability Feedback 26
2.7. Generate Product Recommendations 31
2.8.Update Ingredients of Skincare Products' 35
2.9. List All Products 38
3. Non-Functional Requirements 42
3.1. Development Environment 42
3.2. Security 42
3.3. Scalability 42
3.4. Testing 43
3.5. Data Privacy 43

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.

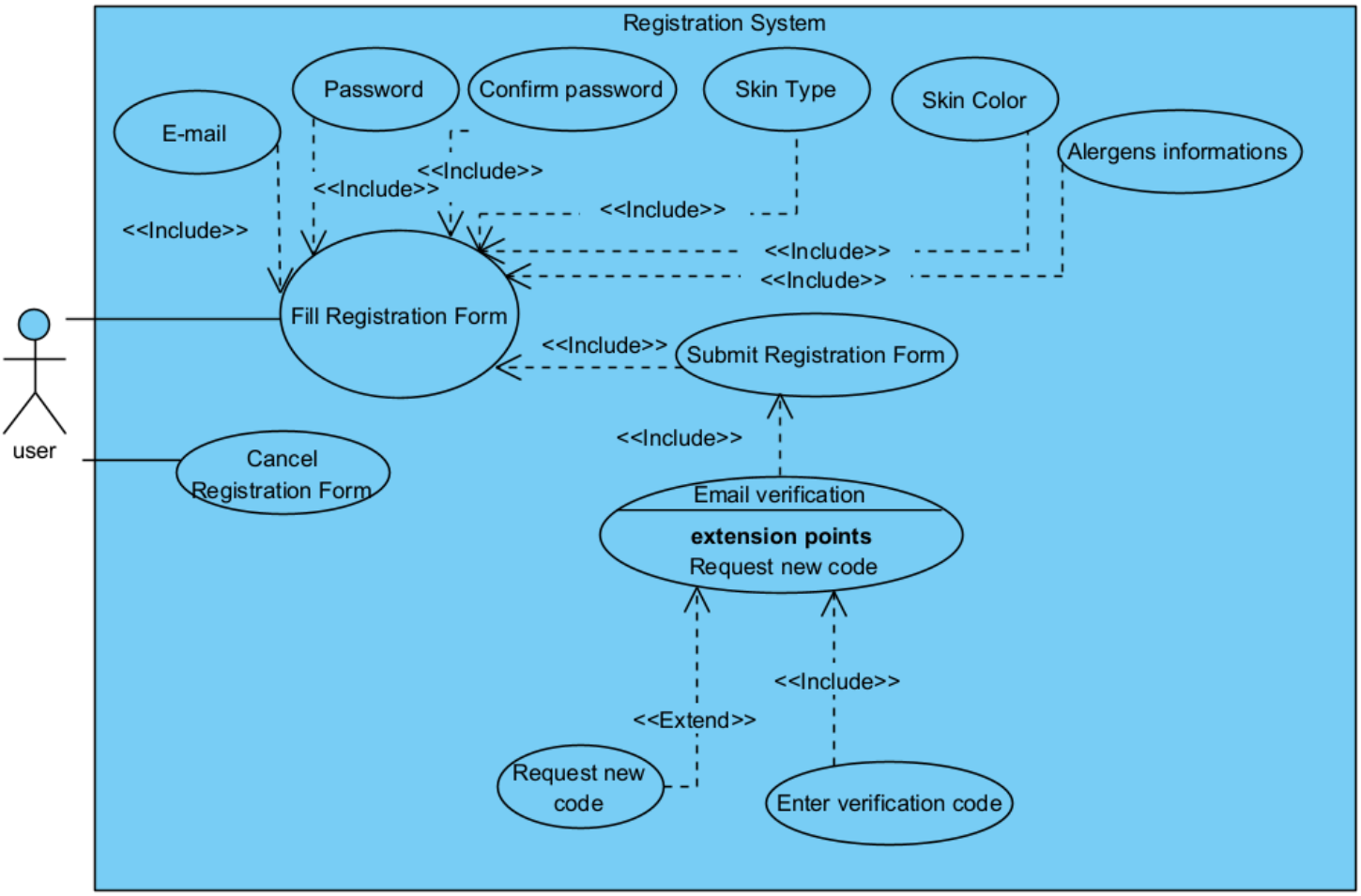

Figure 1 Registration Use Case Diagram

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.

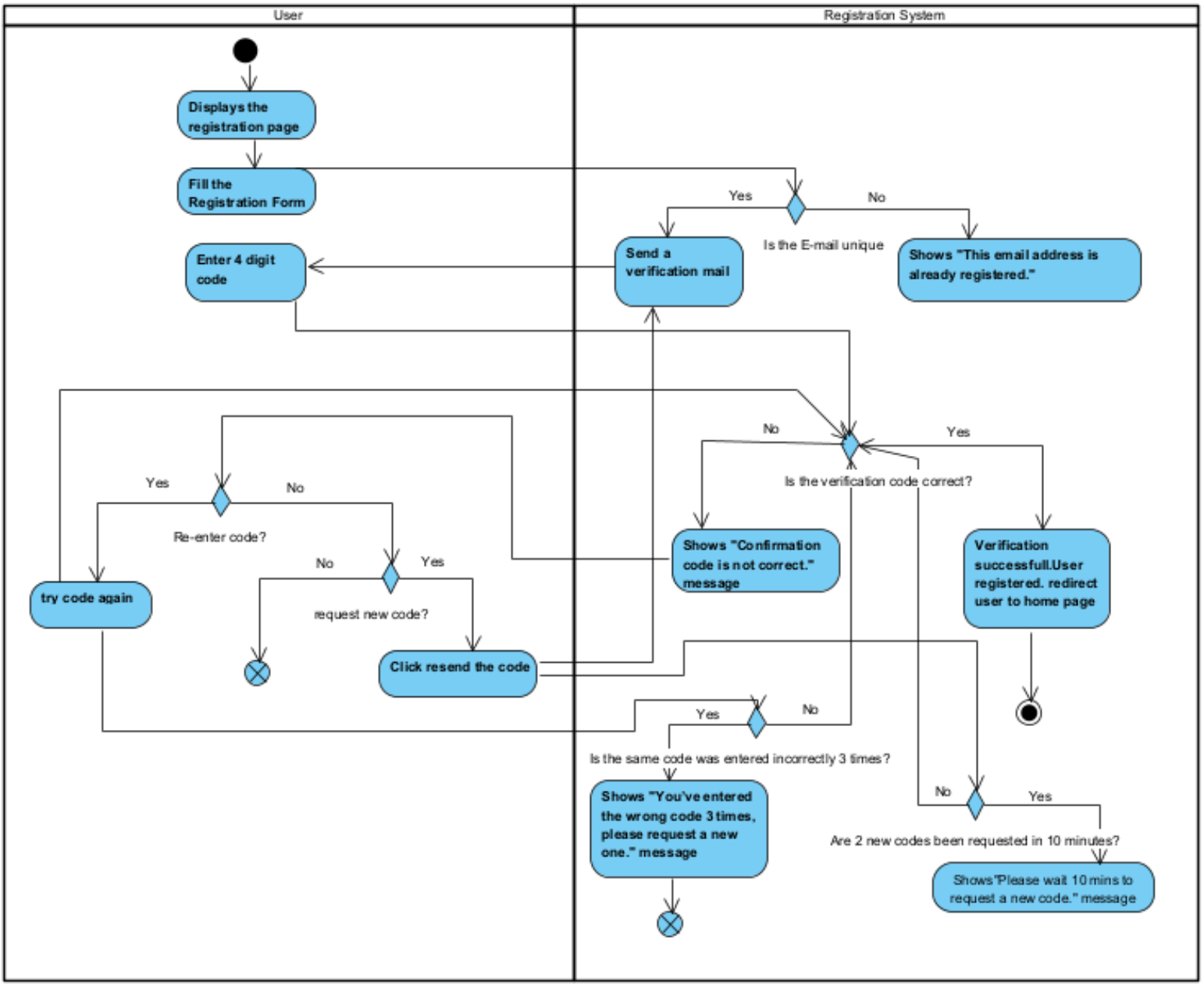

Figure 2 Registration Activity Diagram

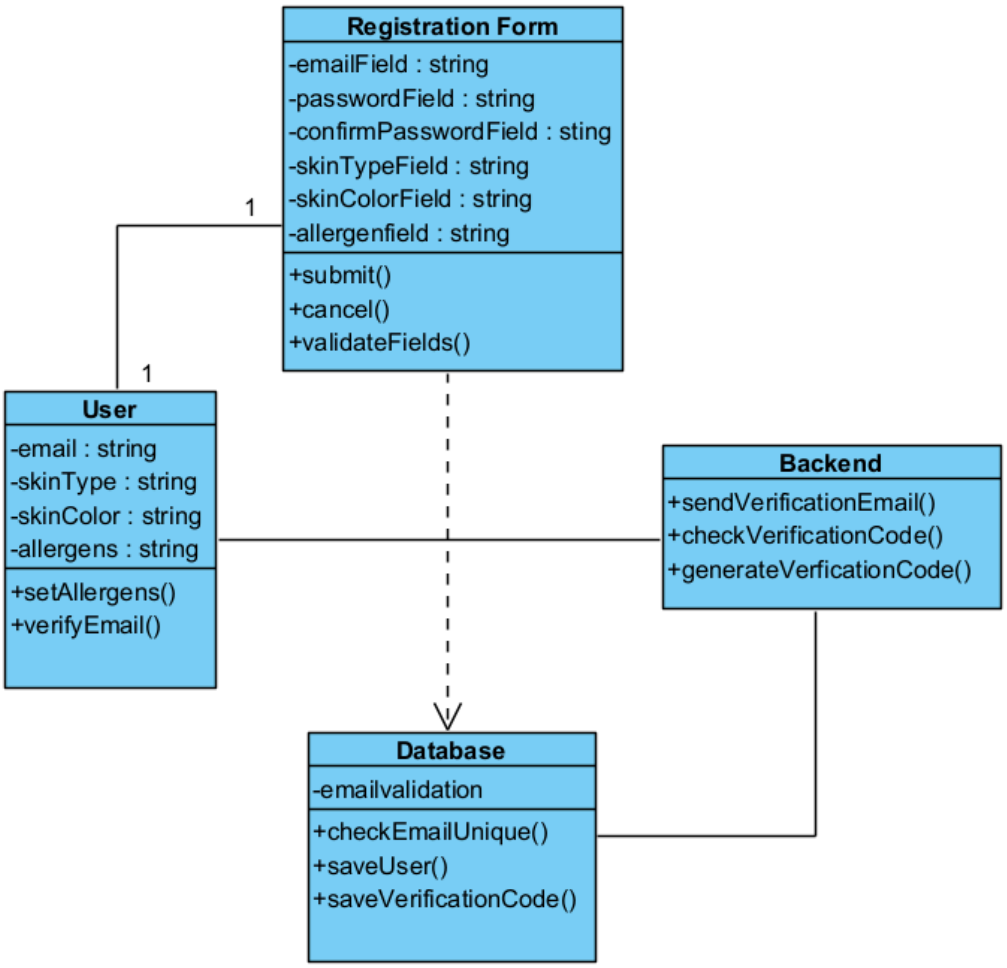

Figure 3 Registration Class Diagram

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.

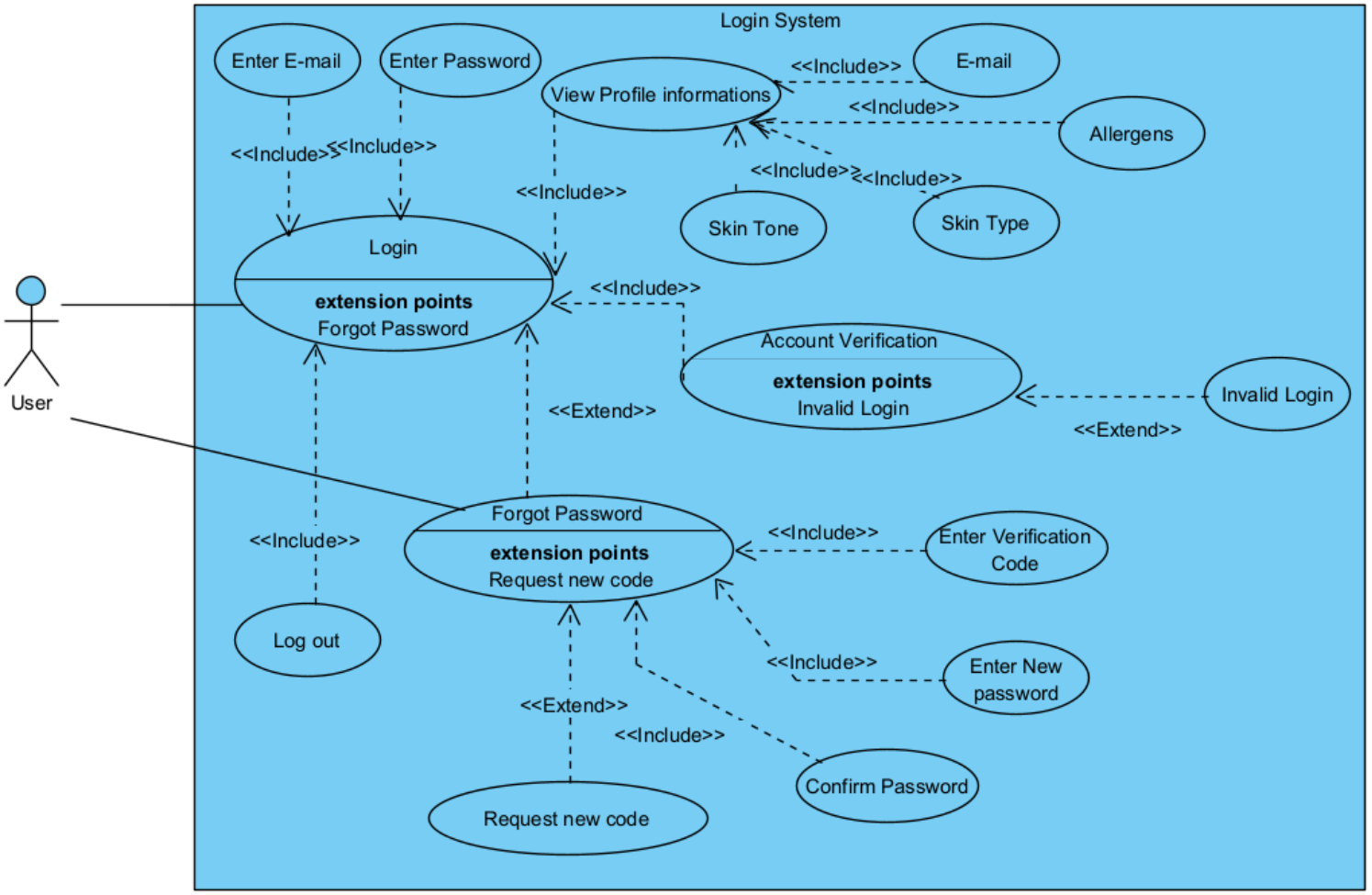

Figure 4 LogIn Use Case Diagram

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.

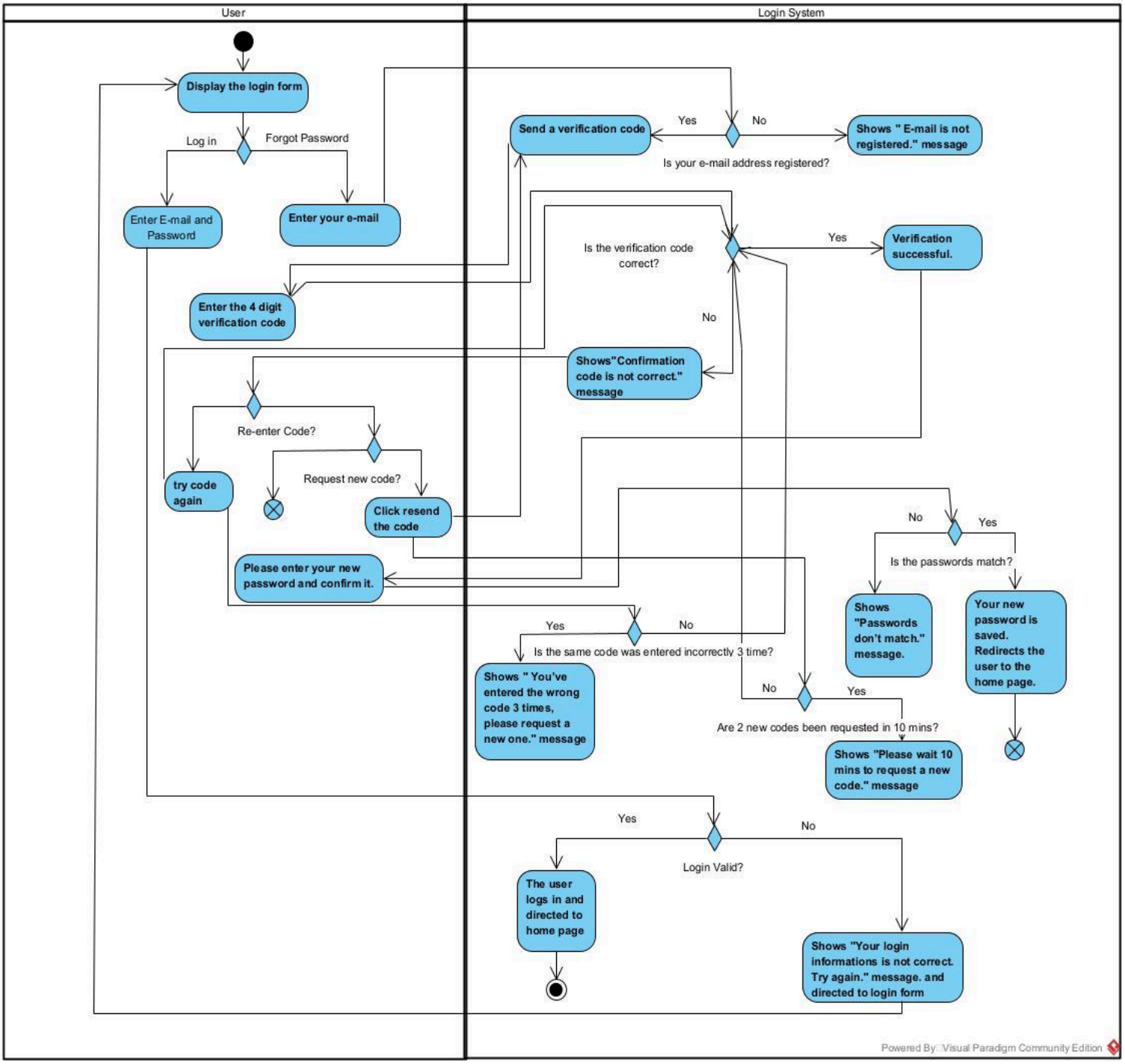

Figure 5 Log In Activity Diagram

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.

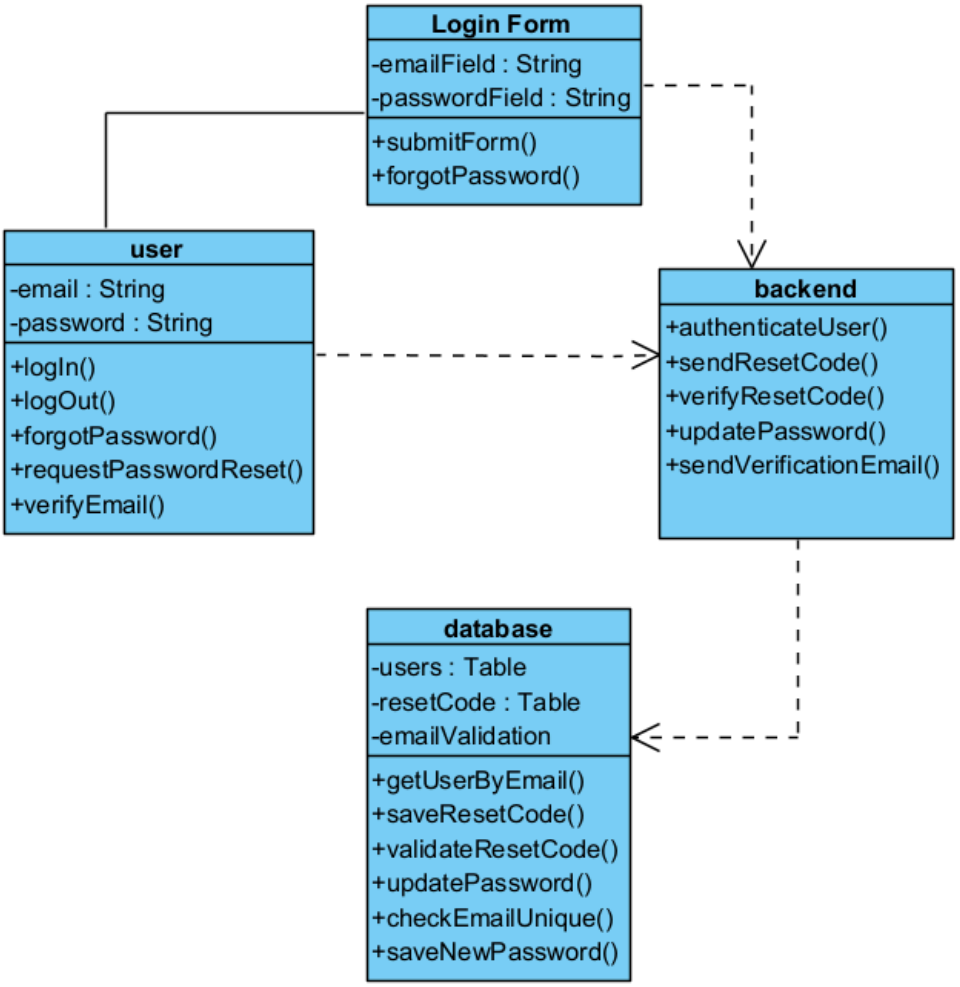

Figure 6 Log In Class Diagram

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.

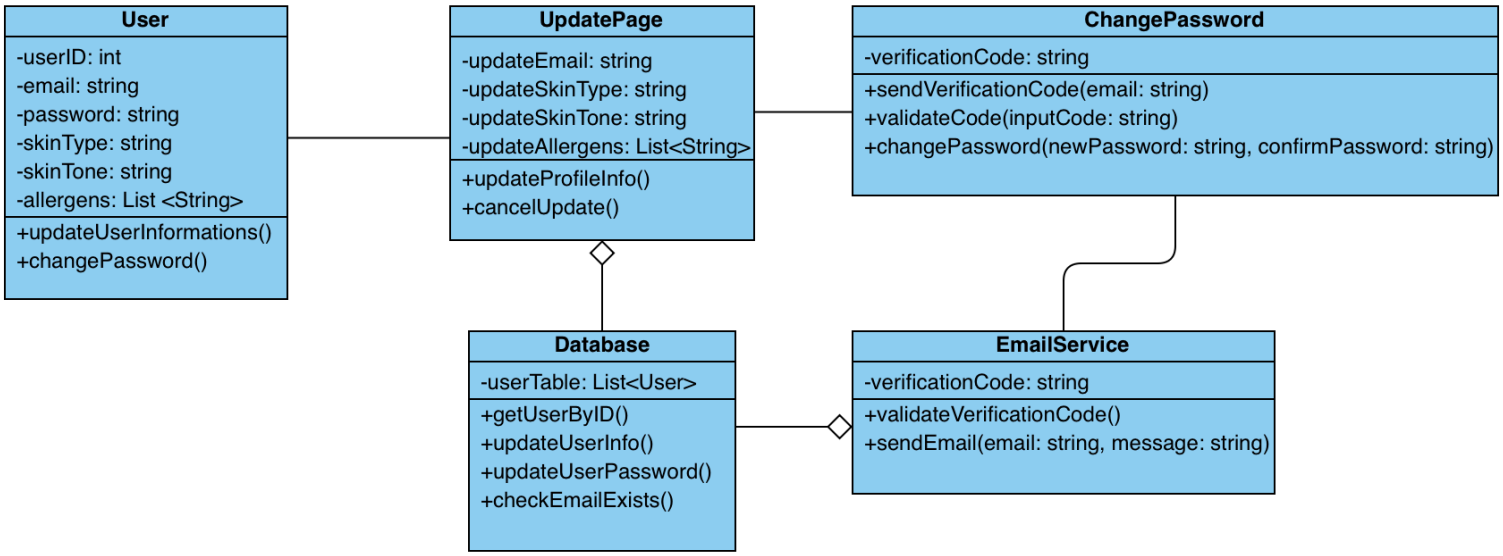

Figure 7 Update Profile Information Class Diagram

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.

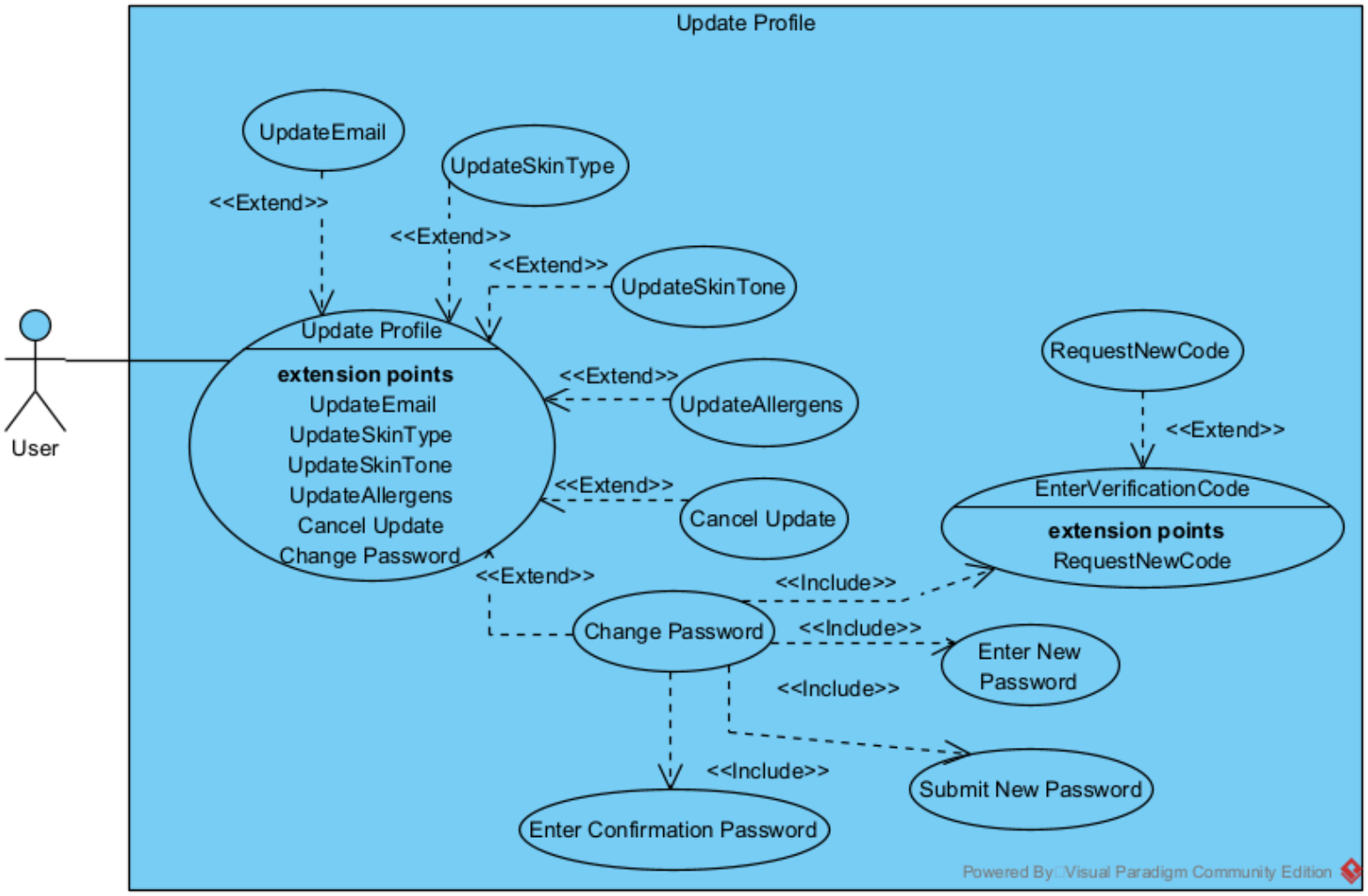

Figure 8 Update Profile Information Use Case Diagram

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.

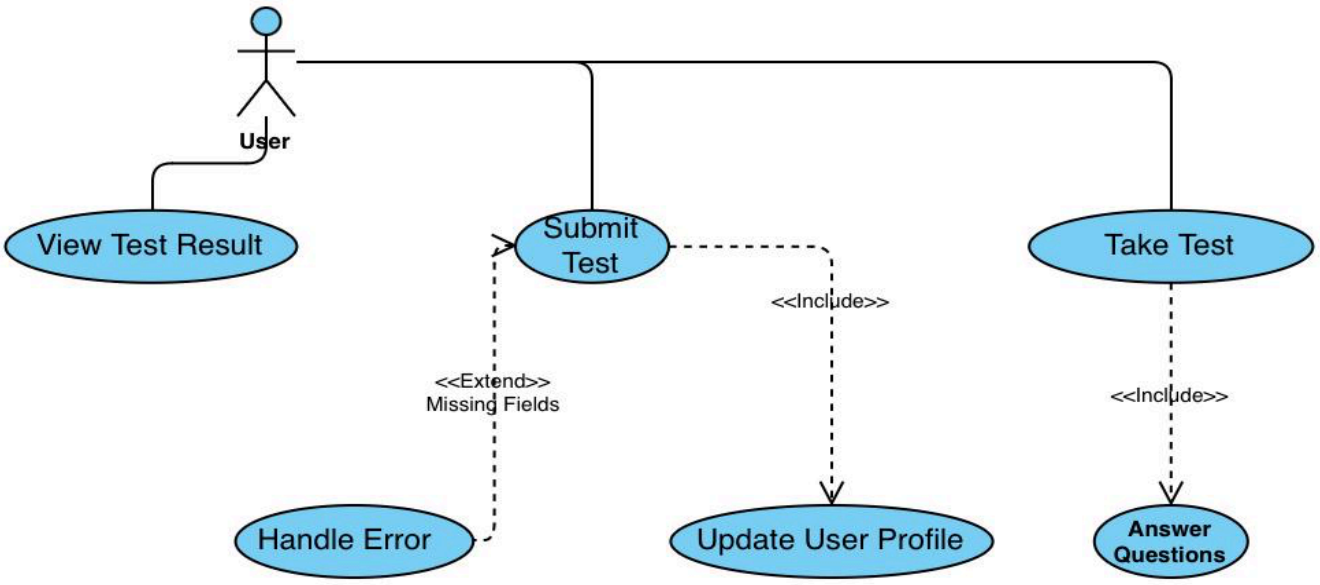

Figure 9 Skin Type Test Use Case Diagram

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.

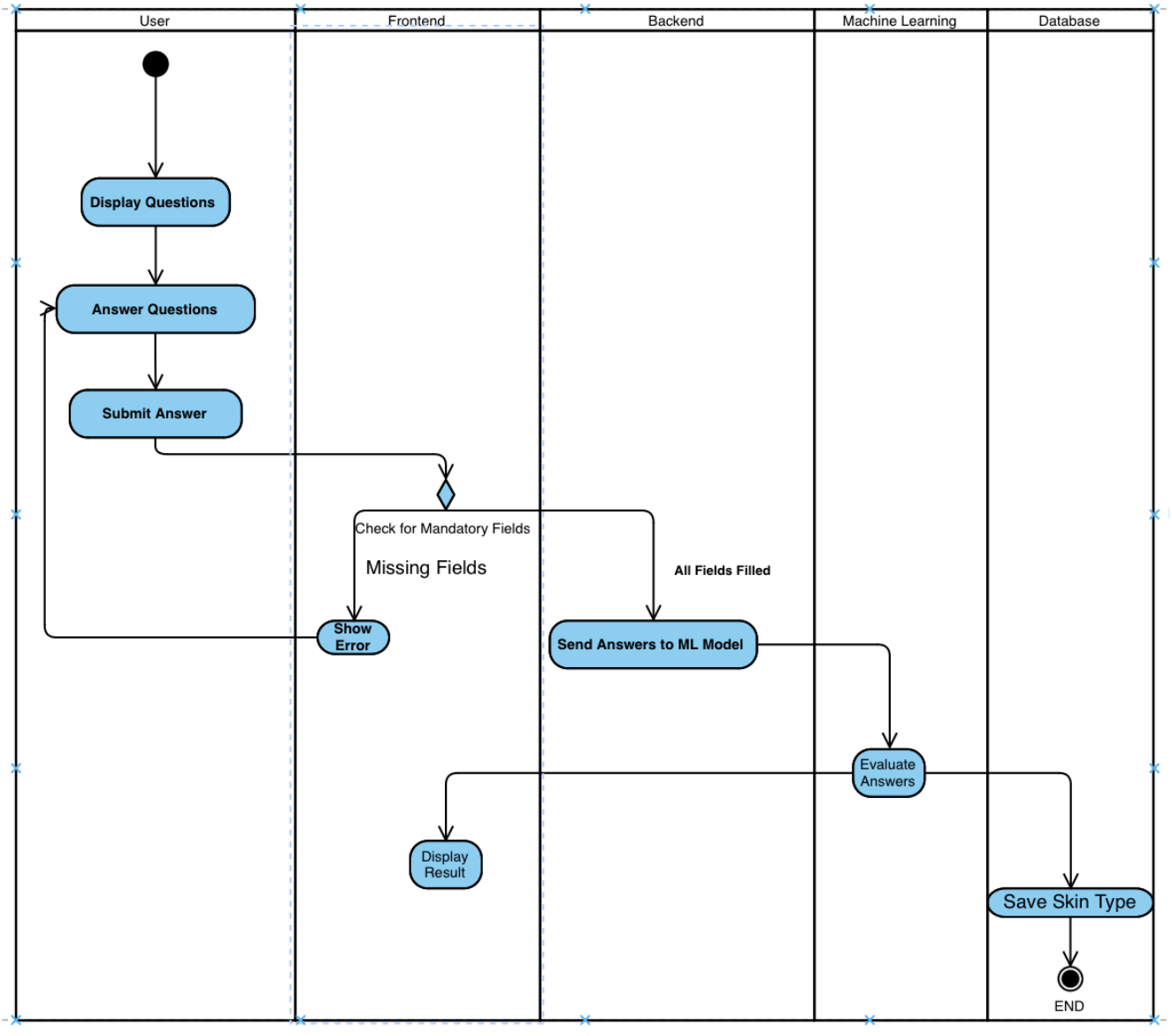

Figure 10 Skin Type Test Activity Diagram

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**).

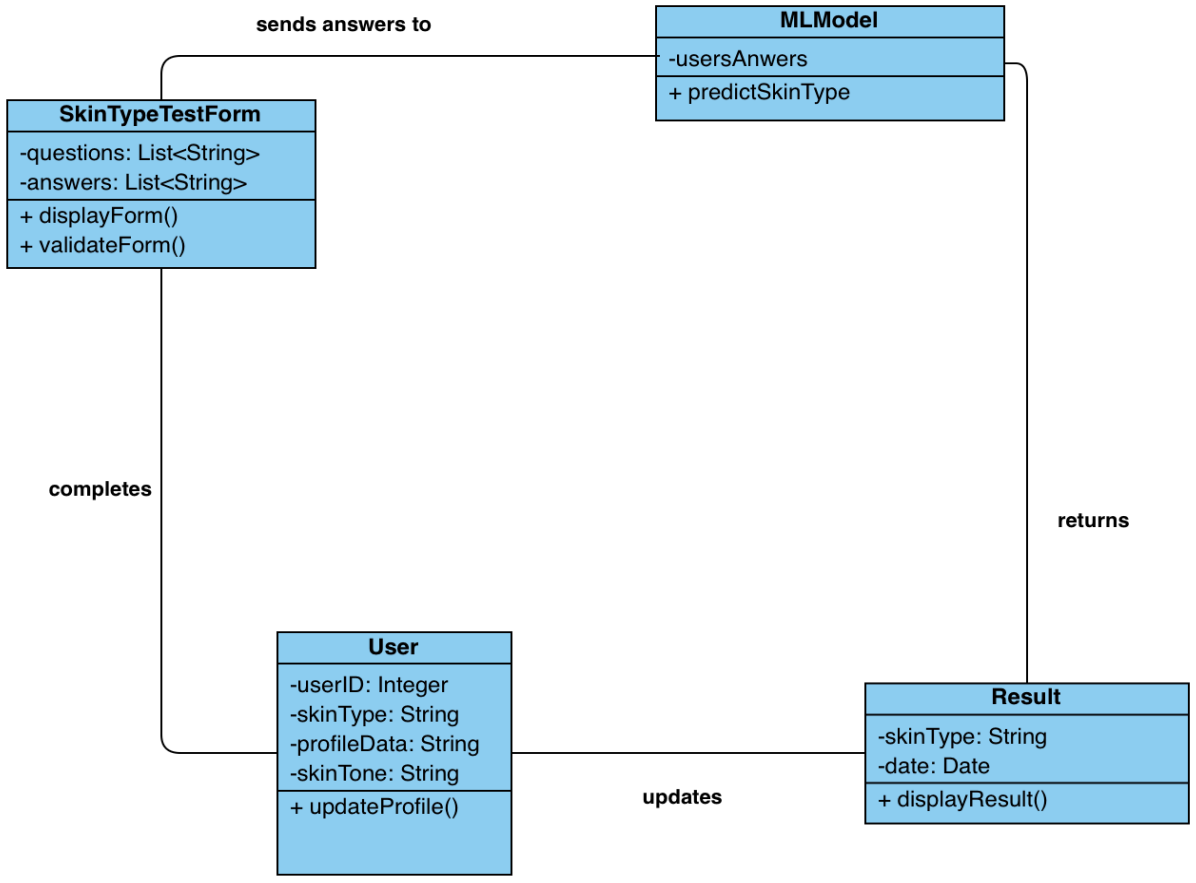

Figure 11 Skin Type Prediction Class Diagram

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.

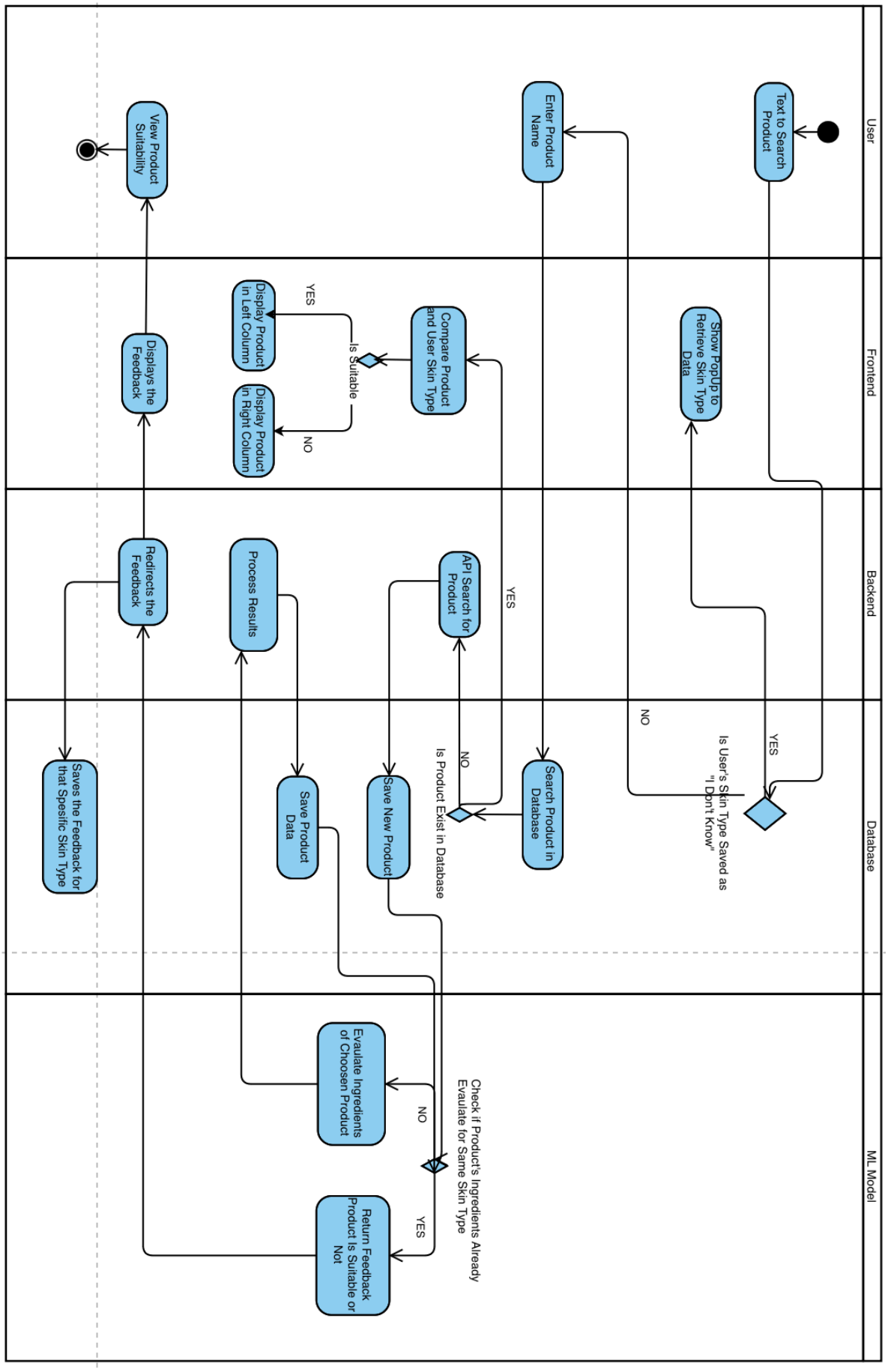

Figure 12 Product Suitability Feedback System Activity Diagram

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.

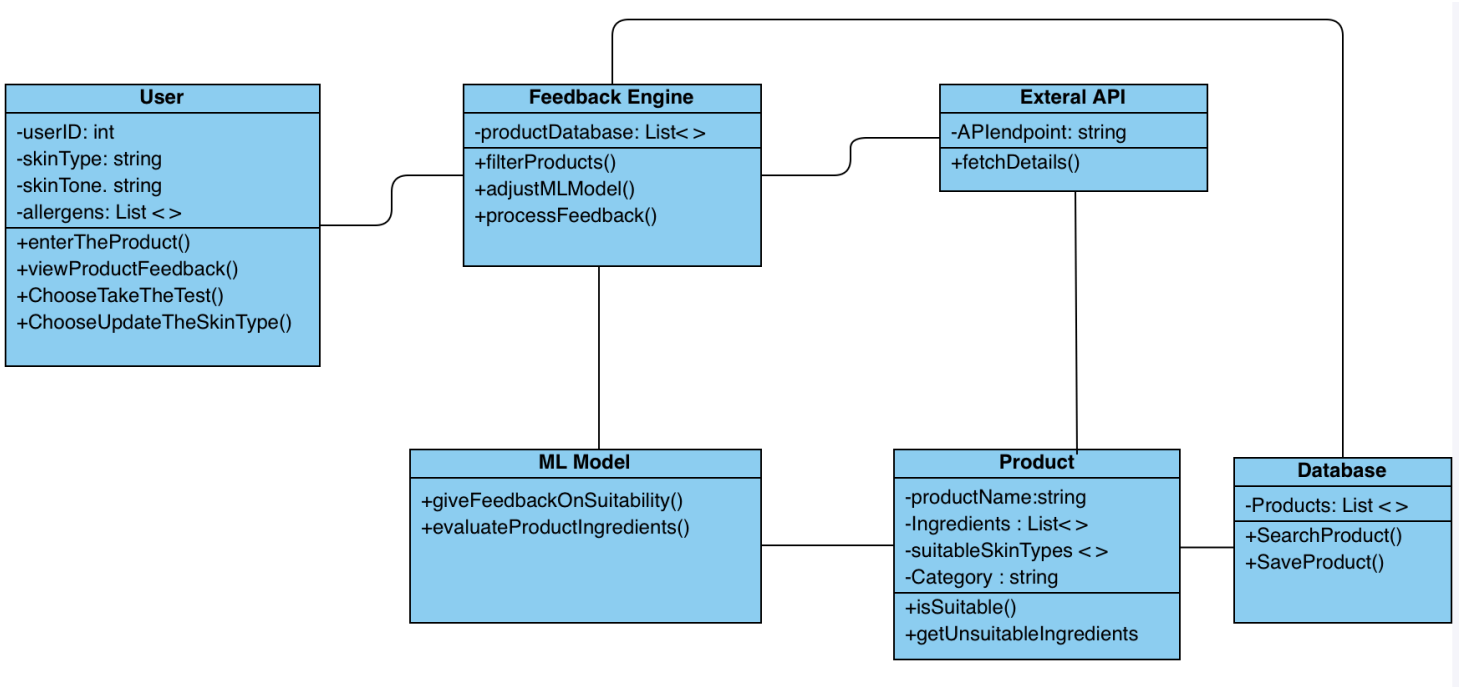

Figure 13 Product Suitability Feedback Class Diagram

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.

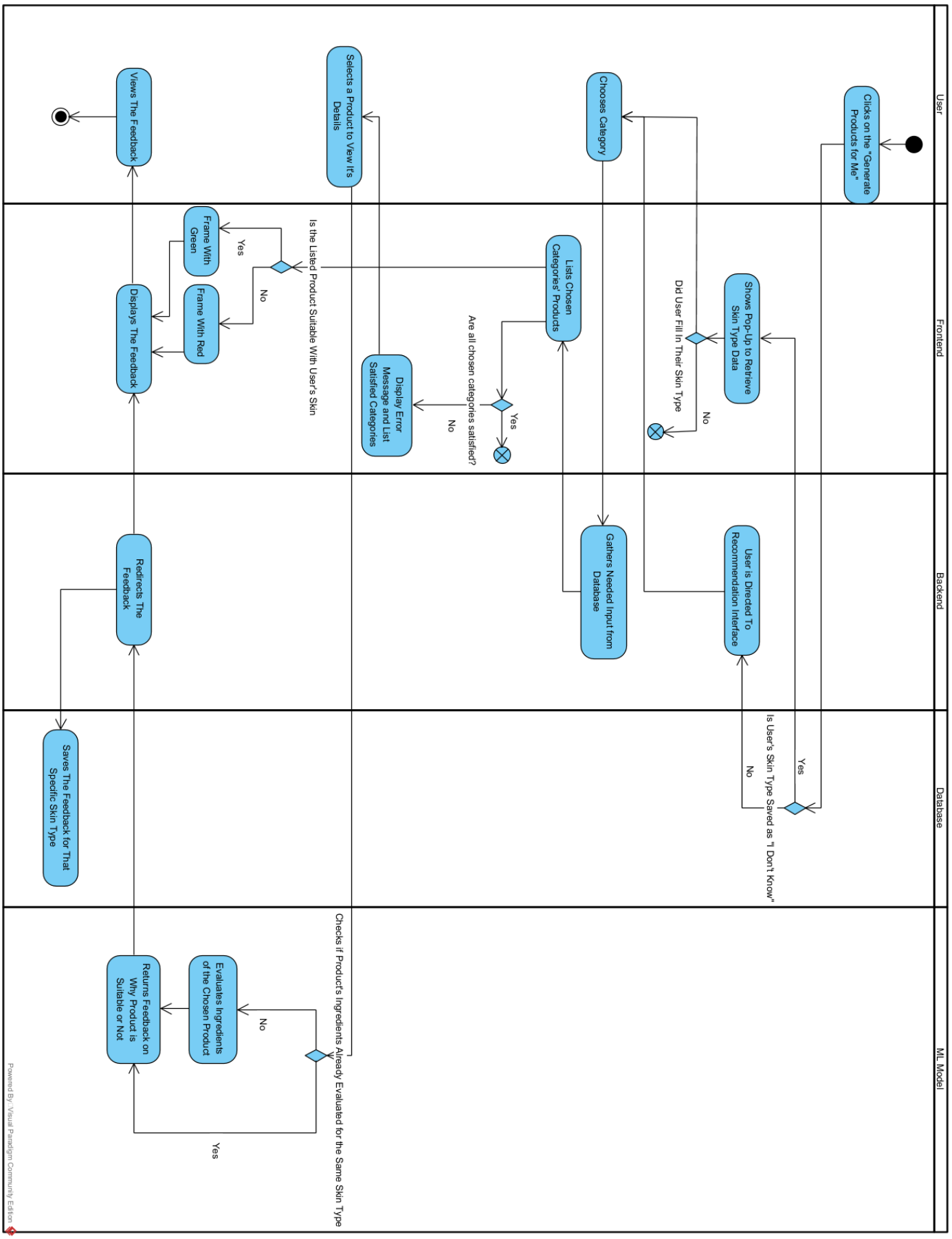

Figure 14 Generate Product Recommendations Activity Diagram

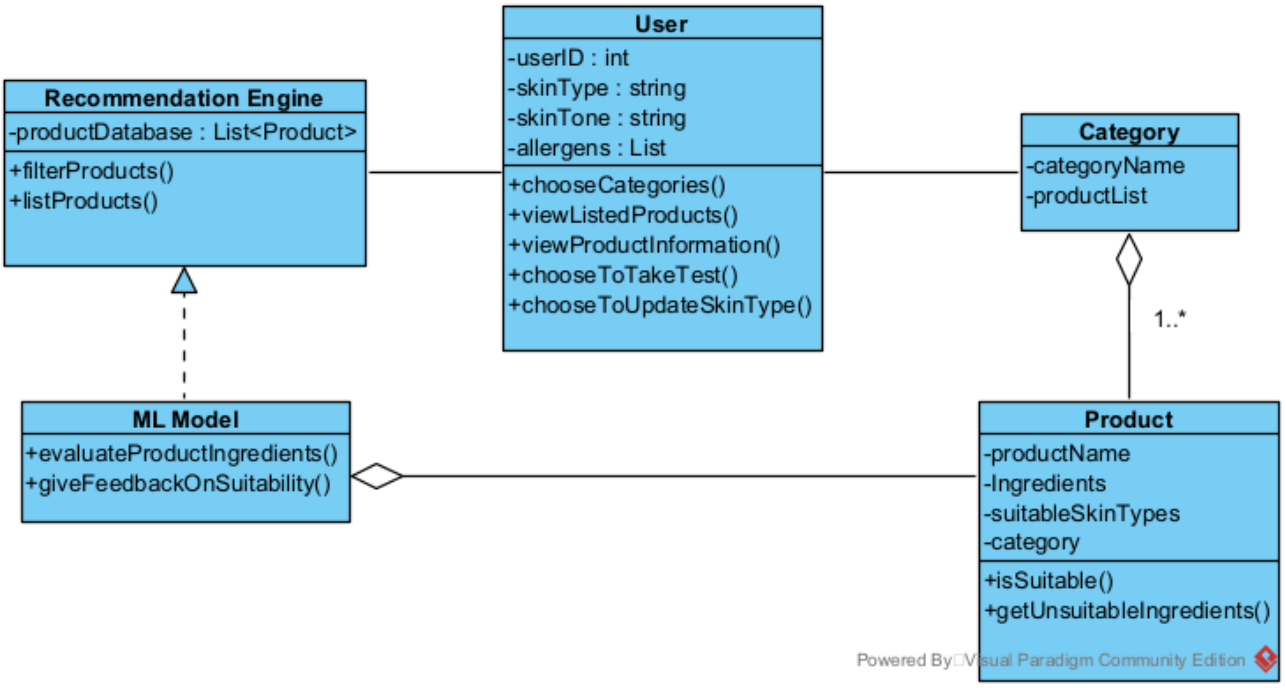

Figure 15 Generate Product Recommendations Class Diagram

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.

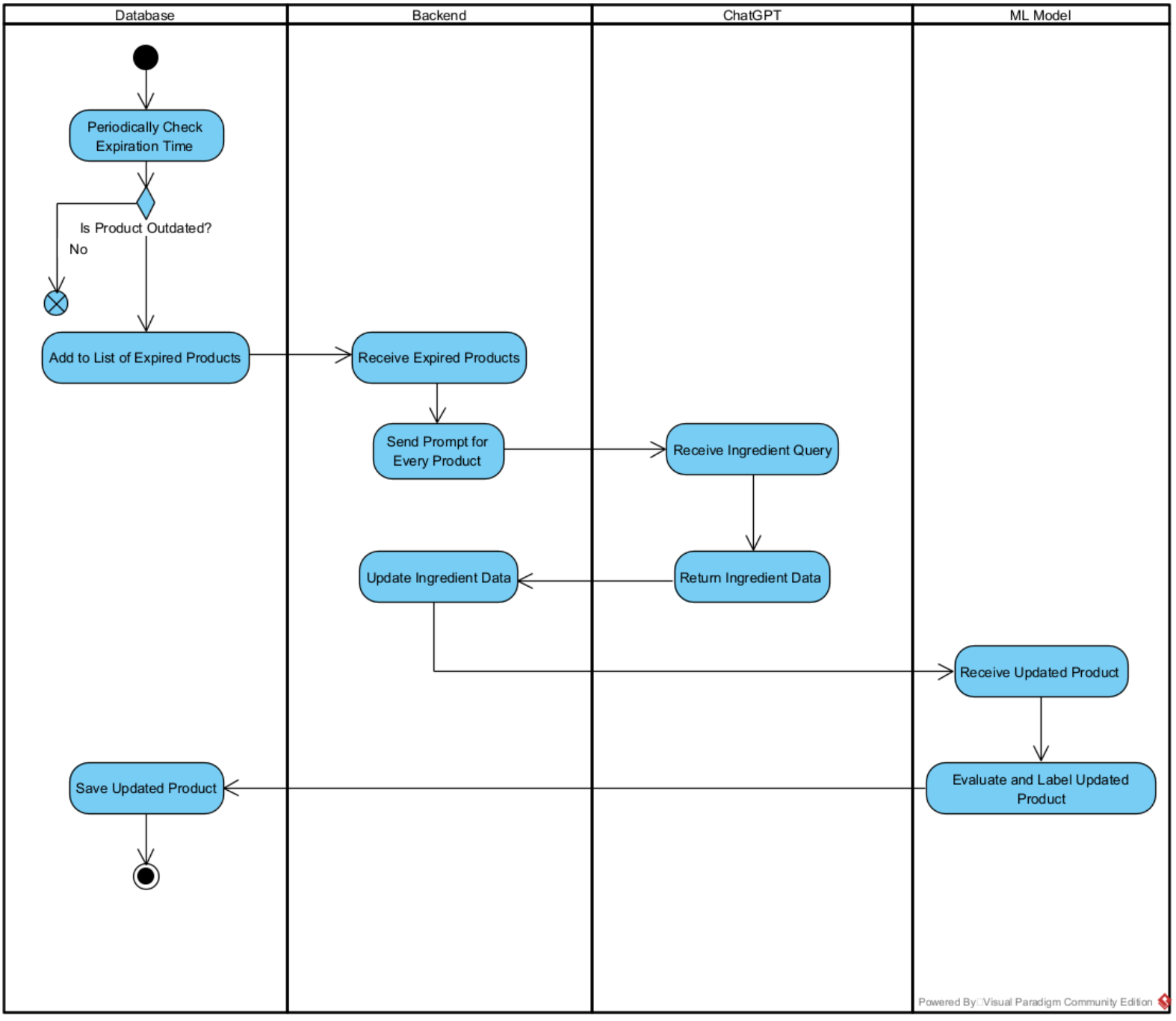

Figure 16 Update Ingredients of Skin Care Products' Activity Diagram

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.

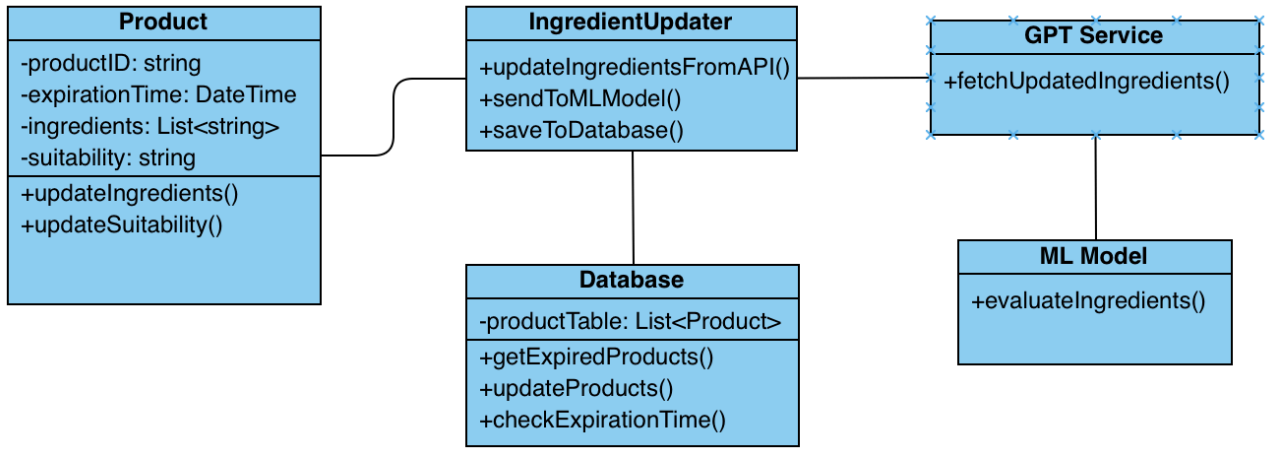

Figure 17 Update Ingredients of Skin Care Products' Class Diagram

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

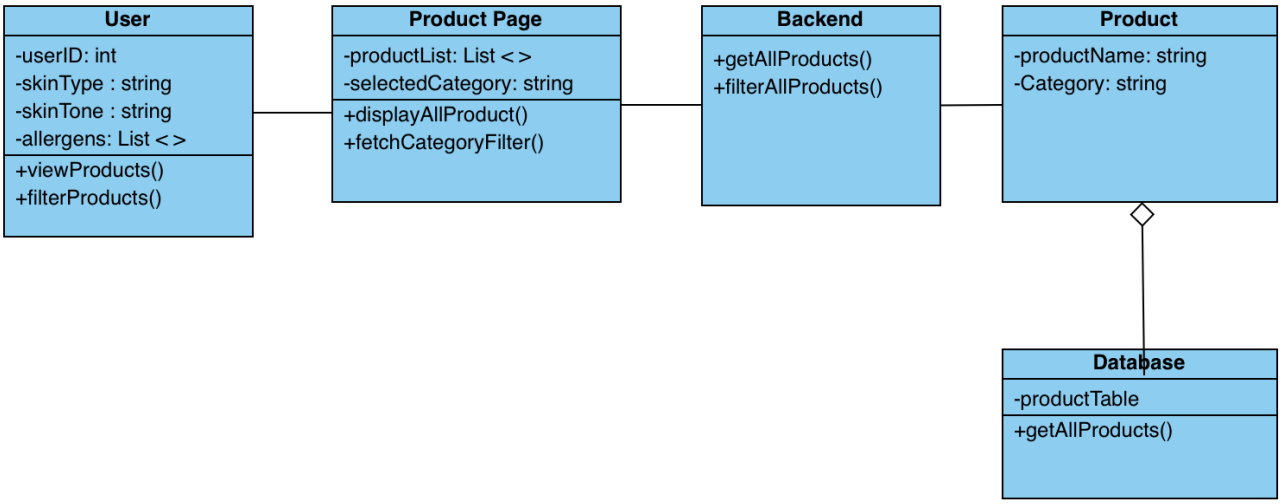

Figure 18 List All Products Class Diagram

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.

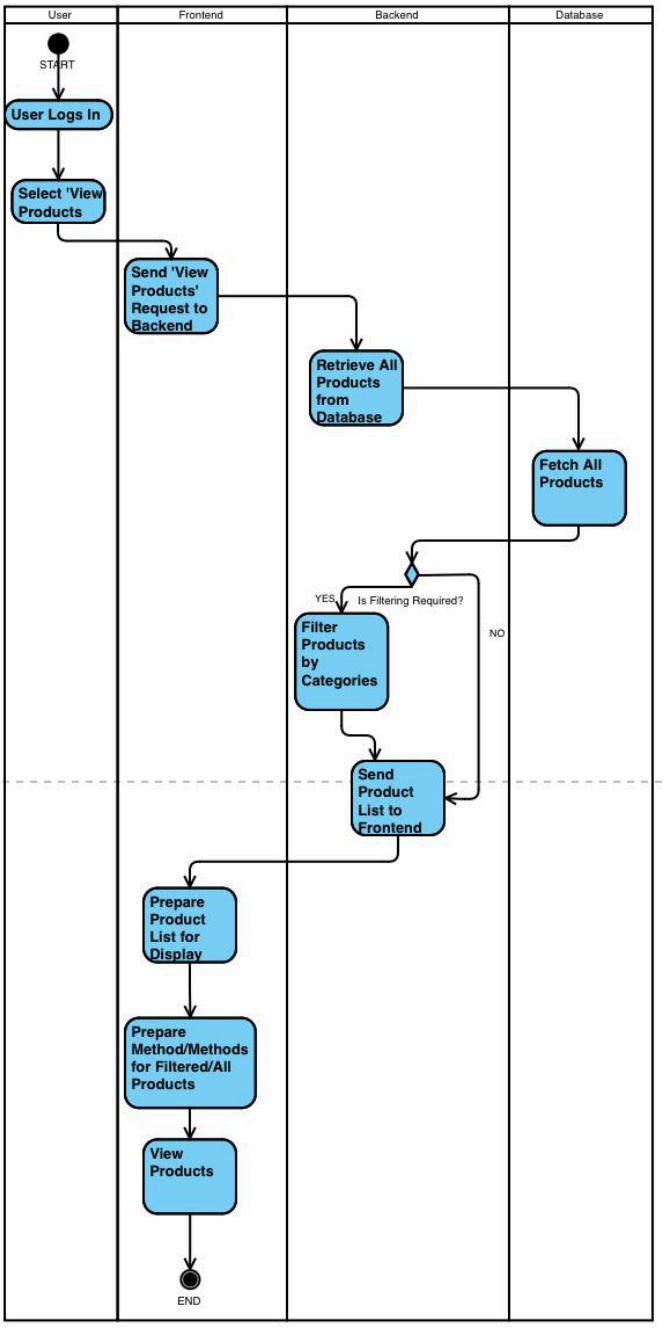

Figure 19 List All Products Activity Diagram

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.

APPENDIX B: DESIGN SPECIFICATIONS DOCUMENT

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

GLOWG: Personalized Skincare Powered by AI
High Level Design
Design Specifications Document

Revision 1.0
20.01.2025

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

Revision History

Revision	Date	Explanation
1.0	20.01.2025	Initial high level design

Table of Contents

Revision History....2
Table of Contents....3
1. Introduction....4
2. GLOWG System Design....5
3. GLOWG Software Subsystem Design....5
3.1. GLOWG Software System Architecture....5
3.2. GLOWG Software System Structure....7
3.3. GLOWG Software System Environment....12
4. GLOWG Software System Detailed Design:....13
5. Testing Design....13

1. Introduction

This section provides an overview of the purpose, main functionalities, and design methodology of the GlowGenie application, including the system's goals, core features, and adherence to quality standards.

Purpose:

The purpose of this project is to develop the **GlowGenie Application** based on the foundation laid in the **GlowGenie Requirements Specification Document (RSD)**. GlowGenie is designed to provide a user-friendly, AI-powered skincare solution to address challenges in determining skin types, evaluating product suitability, and providing personalized recommendations. This Detailed Software Design (DSD) document outlines the design and implementation details of the GlowGenie application, utilizing **Python for backend, React.js for frontend**, and **PostgreSQL** for database management.

Main Functions of GlowGenie System:

1. **User Account Operations:**
    - Login, registration, and logout functionalities.
    - Ability to update user profiles, including skin type, skin tone, and allergens.
    - Password reset functionality with secure email-based verification.
2. **Skin Type Test:**
    - An AI-based interactive questionnaire to determine the user's skin type (Dry, Oily, Combination, Normal).
    - Results are saved in the user's profile for future use.
3. **Product Suitability Feedback:**
    - Analyze skincare products to determine their compatibility with the user's skin type, tone, and allergens.
    - Provide ingredient-based feedback with visual indicators (green/red framing) for suitability.
4. **Generate Product Recommendations:**
    - Suggest personalized products tailored to the user's preferences and skin profile.
    - Allow filtering by product categories like cleansers, moisturizers, sunscreens, toners, and serums.
5. **Update Product Ingredients:**
    - Periodically update product ingredient data via GPT integration.
    - Re-evaluate product suitability based on updated information.
6. **List All Products:**
    - Display a comprehensive list of products with filtering options by category.
    - Provide detailed information for each product, including ingredients and suitability feedback.

Design Basis and Methodology

The design of GlowGenie is based on the **GlowGenie Requirements Specification Document (RSD), Revision 2.0**, in the file `GlowGenie-RSD-2025-01-12.doc`[6]. This DSD document serves as a continuation of the RSD, detailing the system's architectural choices, structural breakdown, and implementation specifics. The notation used to describe the design of the GlowGenie Application is primarily **UML**.

The design methodology adheres to **ISO/IEC 9001 software quality standards**, ensuring a structured and reliable process for developing the GlowGenie application. This standard emphasizes **usability**, **performance efficiency**, and **security**, aligning with internationally recognized best practices for quality management systems. To ensure clarity and consistency, **Unified Modeling Language (UML)** is extensively used throughout the document, detailing the architecture, component interactions, and class designs. This approach guarantees that the system meets user requirements while maintaining high quality and scalability.

2. GLOWG System Design

The GLOWG system design focuses entirely on the **software subsystem**, as no hardware components are required for the development and implementation of this project. This section provides an overview of the system, identifying its components and their interactions, and includes a UML Component Diagram as seen below on *Figure 20* to visualize the relationships among the components including **Skin Type Test UI**, **User Management UI, Product Catalog UI, Recommend Product UI, User, Product, Product Evaluation Model**, **Skin Type Model, Persistence** and **Database**. These components work together to provide a personalized and user-friendly experience for evaluating skincare products. The design decisions ensure scalability, maintainability, and extensibility, making the system adaptable to future requirements.

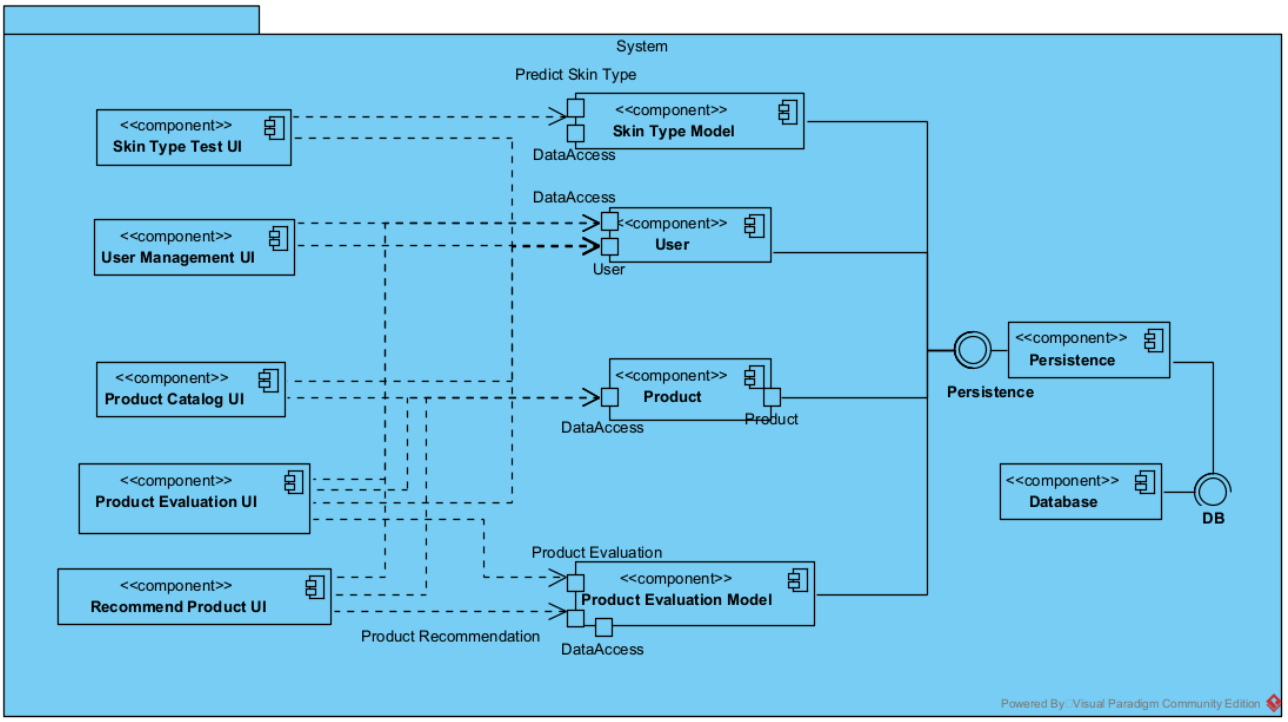

Figure 20 Component Diagram of Glow Genie Software System

3. GLOWG Software Subsystem Design

Since the GLOWG project is a software-only system, this section focuses on the design of the **software subsystem**, which encompasses the entire system. Below is the detailed explanation of the architectural style chosen for the GLOWG system, along with a justification of the design decisions.

3.1. GLOWG Software System Architecture

The GlowGenie system employs a **Layered Architecture**, a widely used architectural style in software engineering. It structures the application into three primary layers, ensuring modularity, scalability, and separation of concerns.

Architecture Layers:

1. **Presentation Layer**:
    - **Role**: Responsible for the user interface and interaction.
    - **Technology**:

    - Developed using React.js (with Node.js for Server-Side Rendering) and Angular for building dynamic and responsive user interfaces. Styled with Bootstrap for consistent and responsive design.
    - **Features**:
        - Handles user interactions (e.g., login, product filtering, and test submissions).
        - Displays data from the backend using RESTful APIs.
2. **Application Layer**:
    - **Role**: Serves as the bridge between the presentation and data layers.
    - **Technology**: Implemented in Python, using Flask, RESTful API and GPT API.
    - **Features**:
        - Processes user inputs and applies business logic.
        - Implements AI-powered features, such as skin type analysis, compatibility feedback, and product ingredient evaluation via GPT API.
        - Facilitates secure and seamless communication between the frontend, database, and external APIs.
3. **Data Layer**:
    - **Role**: Manages persistent data storage and retrieval.
    - **Technology**: Uses **PostgreSQL** for relational database management.
    - **Features**:
        - Stores user profiles, product information, test results, and recommendation data.
        - Ensures data integrity and security through proper indexing, constraints, and role-based access.

Justification for Layered Architecture:

- **Scalability**: Each layer can be scaled independently to handle increased traffic or data.
- **Maintainability**: The separation of concerns allows developers to modify one layer without affecting others.
- **Security**: Sensitive data is handled securely at the application and data layers, reducing exposure risks.
- **Flexibility**: New features can be added or existing features updated within their respective layers.

Design Decisions and Justification

The design decisions for the GLOWG system are guided by the need for scalability, modularity, and user-centric design. Below is a detailed justification for the selected architectural style:

1. **Layered Architecture:**
    - **Why Chosen:** Layered architecture ensures clear separation of responsibilities, making the system easier to extend, debug, and maintain. It also allows independent development of each layer by different teams.
    - **Benefits:** Facilitates the integration of new features (e.g., additional ML models or external APIs) without impacting other layers.
2. **Integration of Machine Learning Models:**
    - **Why Chosen:** Machine learning provides the ability to classify and evaluate product suitability dynamically based on user attributes and product data. It enhances the system's intelligence and adaptability.
    - **Benefits:** Provides transparent and explainable feedback to users, ensuring trust and personalization.
3. **Use of External APIs:**
    - **Why Chosen:** Leveraging external APIs enables the system to access up-to-date product data and enhance analysis without duplicating existing data sources.
    - **Benefits:** Reduces the need for extensive data entry and maintenance, while ensuring accuracy and comprehensiveness.
4. **Scalable Database Design:**
    - **Why Chosen:** PostgreSQL was selected for its support of complex queries and scalability. It handles the relational data of users, products, and feedback effectively.

- **Benefits:** Ensures efficient data storage and retrieval, supporting future growth in the number of users and products.

5. **Frontend Technology (React.js):**
   - **Why Chosen:** React.js was selected for its ability to build highly interactive user interfaces with reusable components.
   - **Benefits:** Provides a responsive and dynamic user experience, which is essential for attracting and retaining users.

3.2. GLOWG Software System Structure

The system structure of GlowGenie is organized into several primary components, each fulfilling specific roles within a layered architecture. These components have been designed with a focus on modularity, scalability, and maintainability, ensuring that the system is adaptable to future enhancements. The core components include the **Frontend**, **Backend**, **Machine Learning Models**, and **Database** packages, with the integration of external APIs handled within the backend.

Key Components and Their Roles:

Frontend Package:

- **Responsibilities:** Handles user interactions, including skin type tests, product browsing, and feedback display.
- **Features:**
  - User-friendly interface for personalized skincare recommendations.
  - Real-time filtering of products based on categories, skin types, and preferences.
  - Visual cues for product suitability (e.g., green/red framing).
- **Sub-components:**
  - **User Management UI:** Manages registration as seen in *Figure 21*, login as seen in *Figure 22* below, profile updates *Figure 23*, and skin type test interactions.
  - **Product Catalog UI:** Displays products, allows filtering, and shows detailed product information.

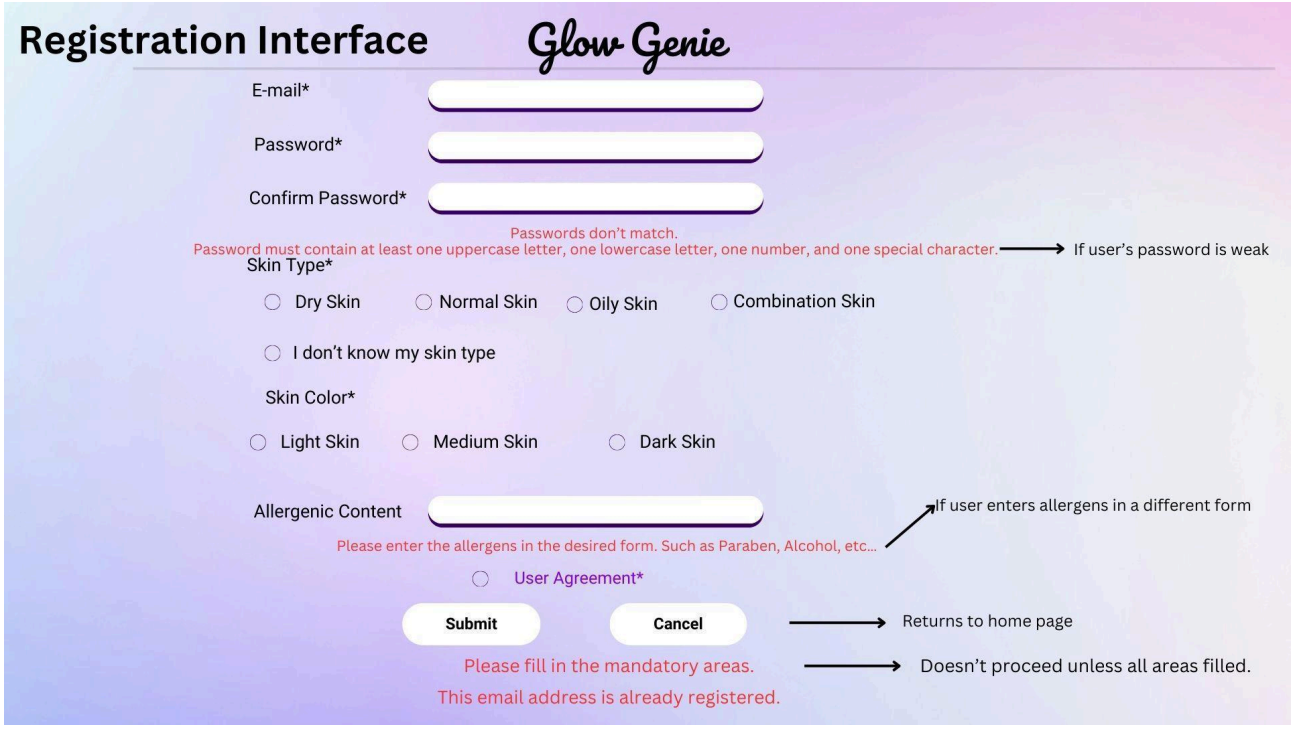

Figure 21 Registration Interface with error messages

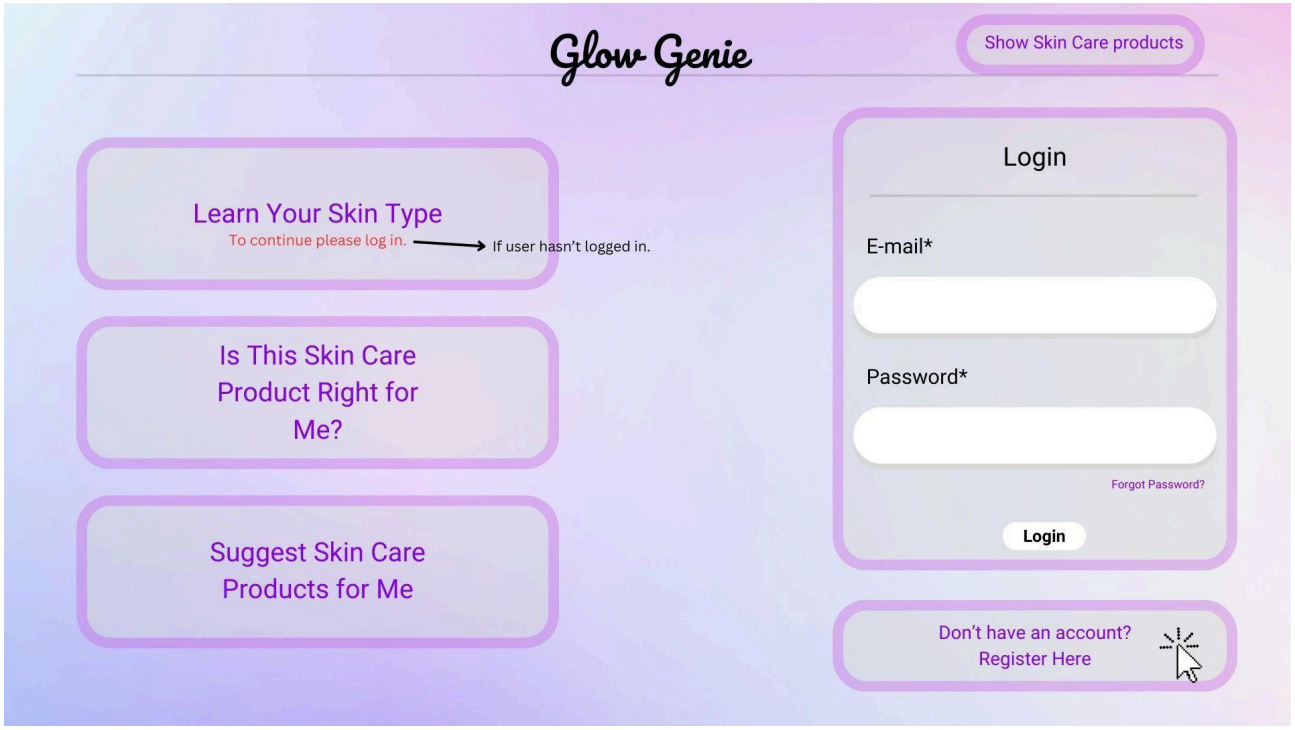

Figure 22 Login Interface from GUI Demo

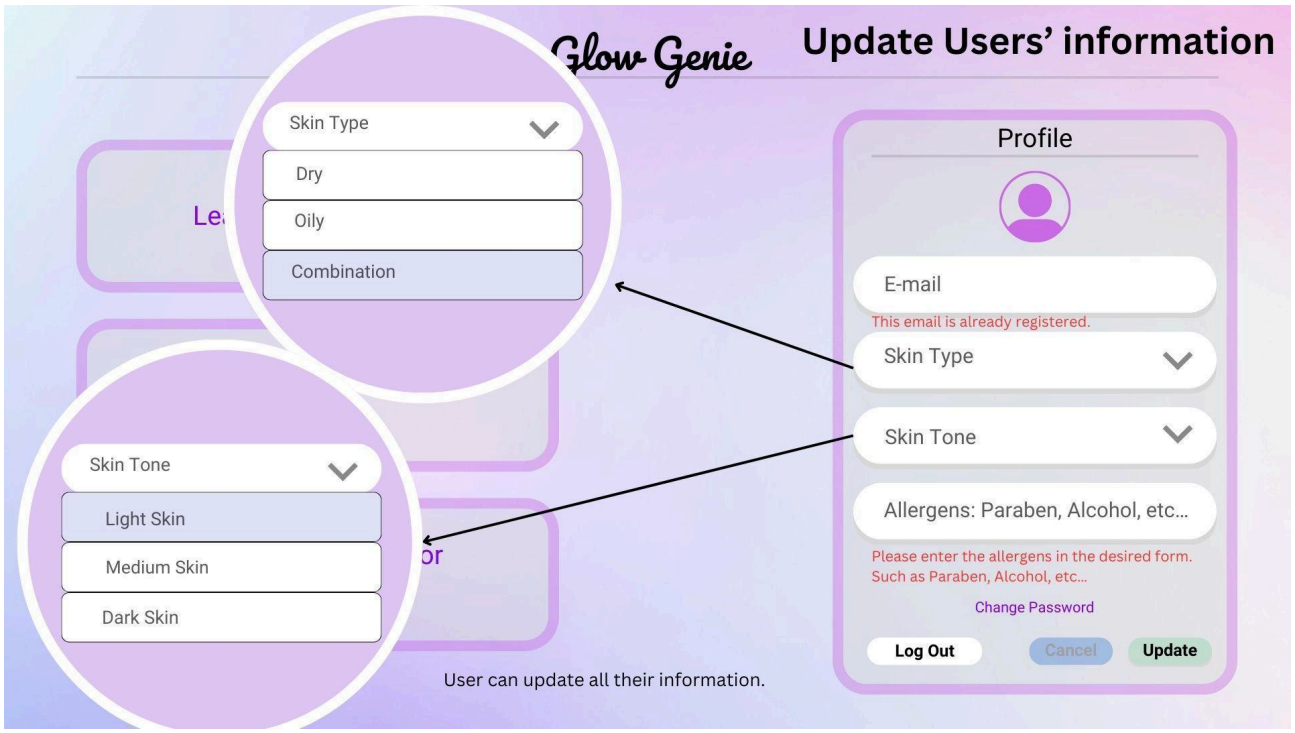

Figure 23 Update Profile Interface with error messages from GUI Demo

Backend Package:

- **Responsibilities:** Manages core business logic, API services, and interactions with the database and external APIs.
- **Features:**
  - Secure user authentication and profile management.
  - Product data management, including ingredient-based filtering.
  - Integration with external APIs for ingredient data (e.g., GPT APIs).
- **Sub-components:**
  - **User Service:** Implements logic for user registration, authentication, and profile management.
  - **Product Service:** Handles product categorization, filtering, and compatibility analysis.
  - **GPT API Integration:** Manages communication with external APIs to fetch updated ingredient data.
  - **Logging & Security:** Ensures secure operations and logs key activities for debugging and monitoring.

Machine Learning Models Package
Responsibilities:
Manages product compatibility evaluations and updates, ensuring accurate skin type classification and suitability labeling for skincare products based on AI models.

Features:

- **Skin Type Prediction:** Determines users' skin types through a questionnaire.

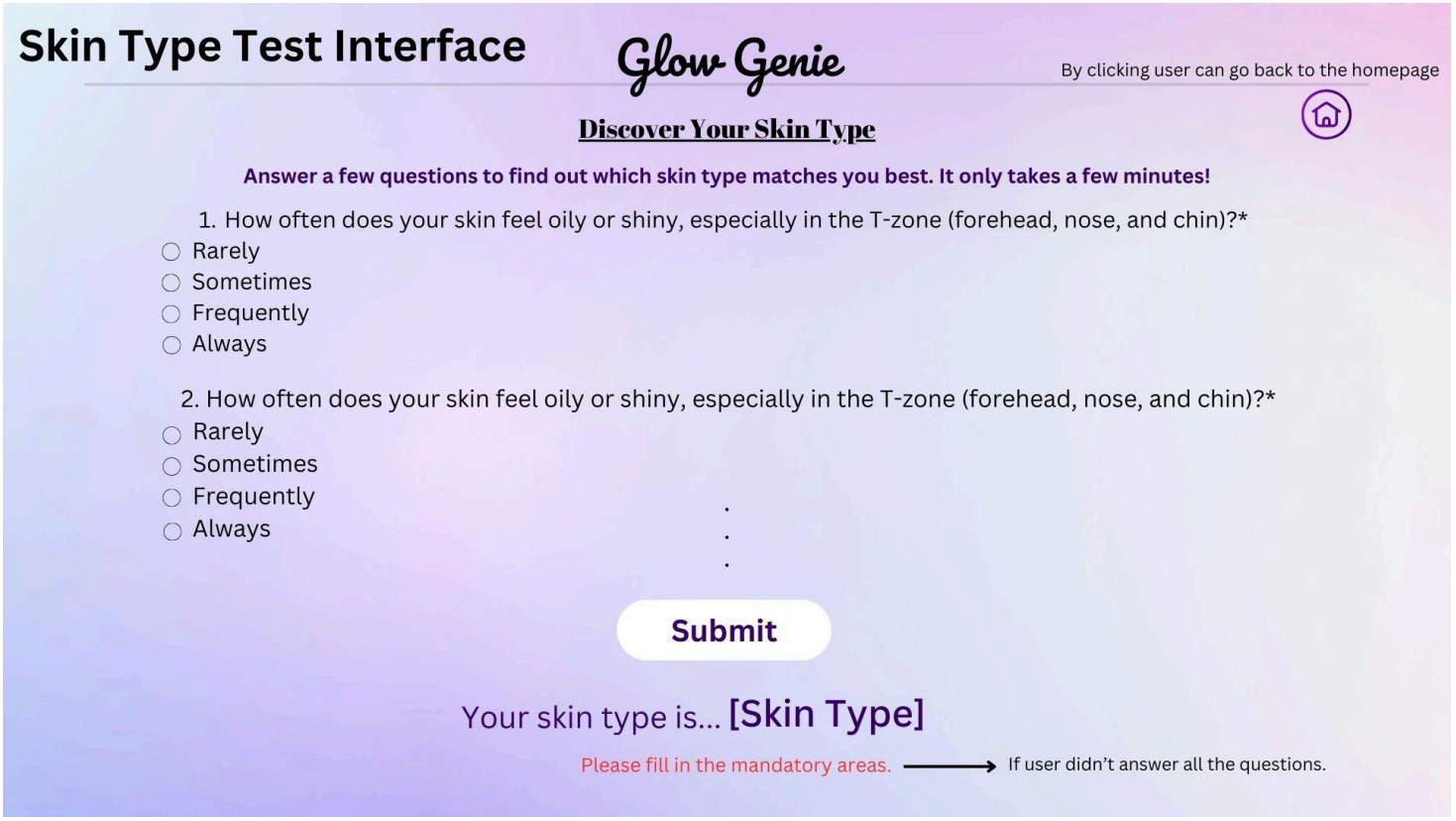

Figure 24 Skin type test and prediction interface from GUI Demo

- **Product Suitability Evaluation:** Classifies products based on ingredient data and predicts their compatibility with various skin types.
- **Ingredient Analysis Feedback:** Identifies specific ingredients contributing to a product's suitability or unsuitability for a given skin type as seen in *Figure 25*.

- **Retraining and Updating Models:** Keeps models up-to-date by incorporating new user data, product information, and feedback.

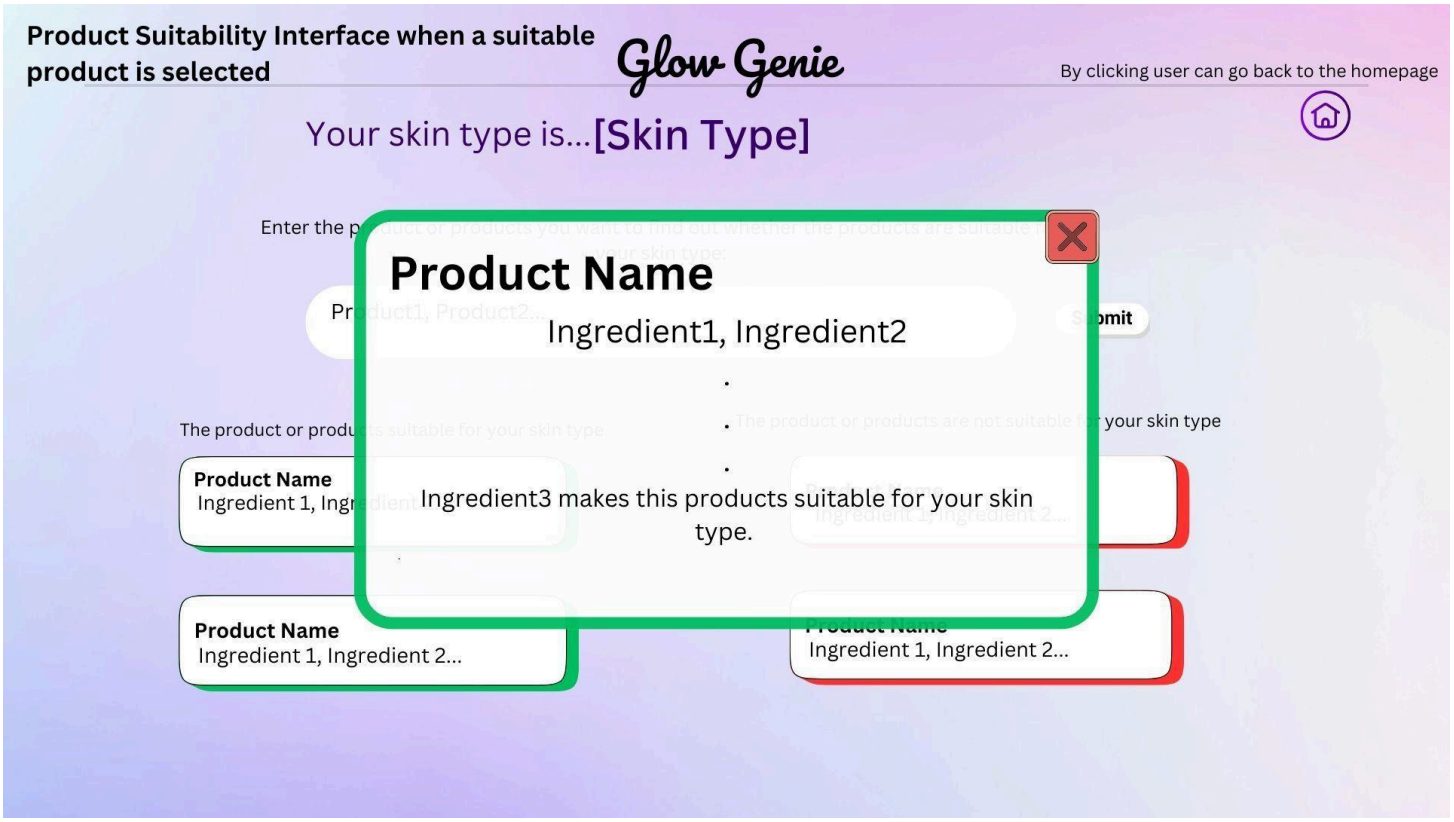

Figure 25 Product suitability interface with given feedback from GUI Demo

Sub-components:

1. **Skin Type Test:**
   - AI-based questionnaire designed to predict users' skin types using a pre-trained model.
2. **AI Model Integration:**
   - Implements the logic for integrating machine learning models to evaluate and label products for skin type suitability.
3. **Model Retraining Service:**
   - Handles retraining of the skin type prediction and product suitability models, incorporating updated ingredient and user feedback data to improve accuracy.
4. **Recommendation Engine:**
   - Generates personalized skincare recommendations based on the compatibility between user skin types and labeled product data seen in *Figure 26*.
   - Provides ingredient-based feedback to users for transparency and informed decision-making.

Skin Type Prediction Model

1. **Input Collection:**
    - The user completes a skin type test, where each question represents a feature in the model (e.g., frequency of dryness, oiliness, sensitivity, etc.).
    - At first, all questions will directly contribute as features, ensuring a straightforward and comprehensive representation of the user's skin characteristics.
2. **Model Processing:**
    - The responses are processed and fed into the pre-trained skin type prediction model.
    - The model is trained to classify skin types (e.g., dry, oily, combination, sensitive) based on labeled training data.
3. **Prediction:**
    - The model outputs a single skin type as the prediction for the user.
    - The result is saved in the user's profile and used for product recommendations.

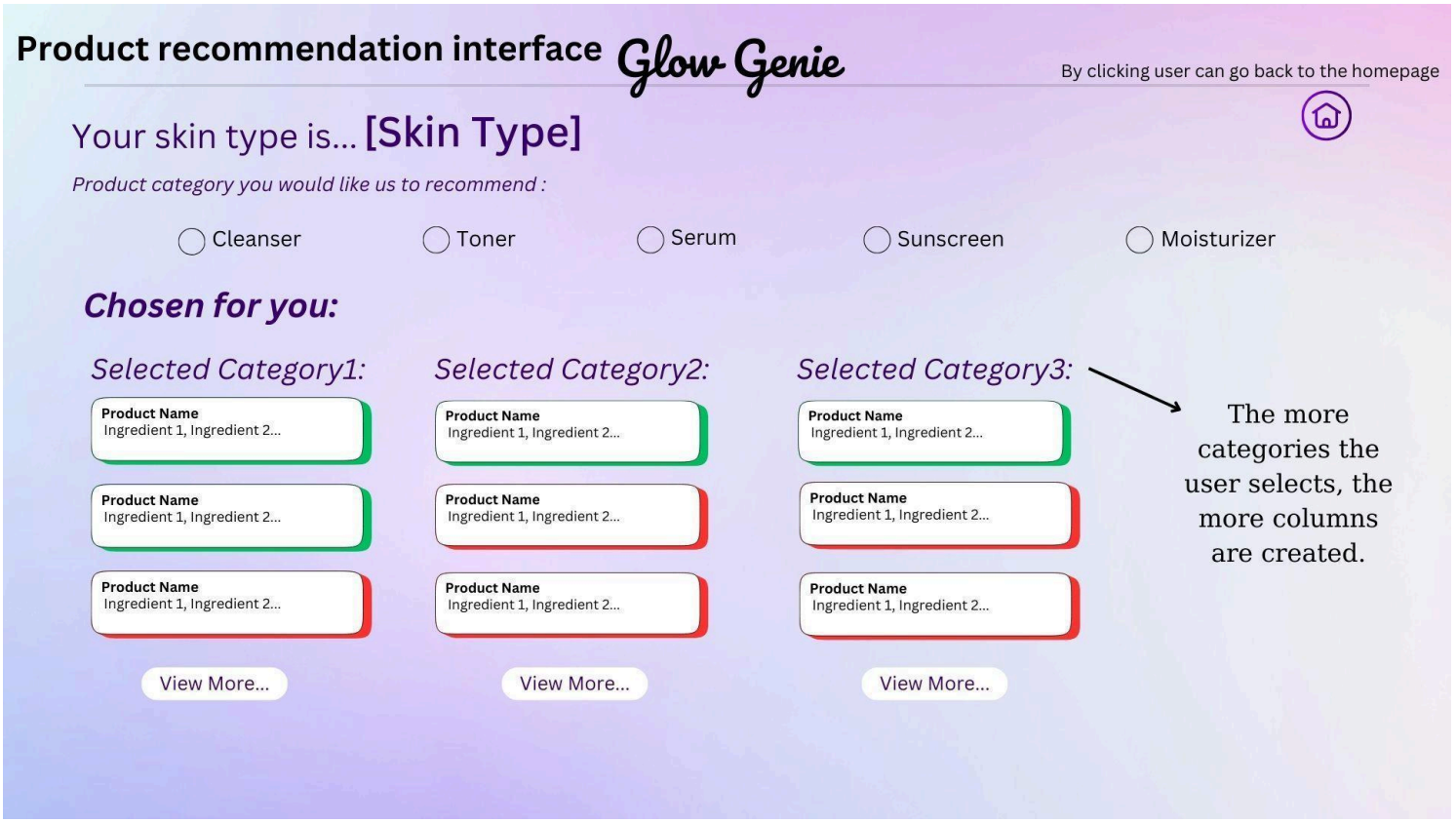

Figure 26 Product recommendation interface from GUI Demo

Evaluating and Labeling Products for Skin Types Model

1. **Product Data Preparation:**
    - Ingredient data for each product is retrieved from the database.
    - This data serves as input to the model for suitability evaluation.
    - No additional conversion into feature vectors is considered as Random Forest can directly handle structured input.
2. **Model Processing:**
    - A Random Forest model is used to predict product suitability.
    - The model evaluates how suitable a product is for each skin type (e.g., 85% suitable for dry skin, 20% for oily skin, etc.).
    - Softmax is applied to the output to provide a percentage for each skin type.

3. **Labeling:**
    - The product is labeled for each skin type as either **suitable** or **unsuitable** based on the highest suitability percentage or a threshold.
    - Using SHAP (SHapley Additive exPlanations), the model identifies which ingredients contributed to the suitability or unsuitability of the product for each skin type.
    - The reasons for labeling (i.e., the specific ingredients responsible for suitability or unsuitability) are saved to the database alongside the product.
4. **Database Update:**
    - The product data is updated with its suitability percentages, suitability labels for each skin type, and SHAP analysis results (ingredient-level explanations).

Database Package:

- **Responsibilities:** Manages user data, product information, and feedback in a structured database.
- **Features:**
    - Persistent storage of user profiles, products, and recommendations.
    - Efficient data retrieval and updates.
- **Sub-components:**
    - **User Repository:** Handles CRUD operations for user-related data.
    - **Product Repository:** Manages product-related data in the database.
    - **Recommendation Repository:** Stores and retrieves generated recommendations and feedback.

3.3. GLOWG Software System Environment

The GLOWG software subsystem is designed to operate in a robust and scalable environment, leveraging modern hardware, system software, and middleware to ensure optimal performance and reliability. This section provides a detailed description of the target environment, programming tools, and supporting software.

3.3.1 System Software Environment

The system software environment comprises the backend, frontend, database, and middleware technologies that collectively run the GLOWG application.

1. **Backend**:
    - **Programming Language**: Python 3.10.
    - **Frameworks**:
        - Flask: Lightweight web framework for RESTful API implementation.
    - **APIs**: Integration with GPT APIs for ingredient data updates and external AI enhancements.

    Machine Learning Integration:

    - **Scikit-learn:** For implementing the Random Forest model used for product suitability evaluation.
    - **Numpy:** For numerical computations and efficient matrix operations.
    - **Pandas:** For data manipulation and preprocessing, particularly for handling datasets of ingredients and skin type responses.
    - **Softmax (via Scikit-learn/Numpy):** To compute suitability percentages for products across different skin types.
    - **SHAP (SHapley Additive exPlanations):** For generating interpretability insights for product suitability and storing ingredient-based explanations in the database.

2. **Frontend**:
    - **Programming Language**: JavaScript/TypeScript, HTML, CSS
    - **Frameworks**:
        - React.js 18: For building responsive and dynamic user interfaces.
        - Node.js: For server-side rendering and improved performance.
    - **Styling**: Bootstrap 5 for consistent and responsive UI design.
3. **Database**:
    - **Database Management System**: PostgreSQL 15.
    - **Features**:
        - Support for advanced queries and indexing for faster data retrieval.

3.3.2 Development Tools

The development environment for the GLOWG application uses the following tools and platforms:

1. **Integrated Development Environment (IDE)**:
    - Visual Studio Code for coding, debugging, and integrating frontend and backend services.
2. **Version Control**:
    - Git for source code management and collaborative development.
    - GitHub as the central repository for version control and issue tracking.
3. **Testing Tools**:
    - **Postman**: For API testing and verification.
    - **Selenium**: For automated testing of the user interface.
    - **Pytest**: For unit and integration testing in the backend.

3.3.3 Justification for Environment Choices

1. **Modern Frameworks and Tools**:
    - React.js and Flask ensure efficient development and user-friendly interfaces.
    - PostgreSQL provides robust data management capabilities.
2. **Middleware**:

- **Authentication**:
    - **JWT** (jsonwebtoken)
- **Body Parsing**:
    - **express.json()** and **express.urlencoded()** (Built-in in Express)
- **Input Validation**:
    - **express-validator**
- **Error Handling**:
    - Custom middleware in Express
- **Caching**:
    - **cache-manager** (in-memory, Redis, etc.)

4. GLOWG Software System Detailed Design:

Detailed design will be carried out in the context of the COMP 4920 course next semester.

5. Testing Design

We will use **unit testing**, **integration testing**, and **user acceptance testing (UAT)** to ensure the application is reliable and user-friendly. Unit testing will validate individual components, integration testing will ensure seamless interactions between modules, and UAT will confirm the application meets user expectations. Together, these methods ensure functionality, reliability, and user satisfaction.

APPENDIX C: PRODUCT MANUAL

COMP4920 Senior Design Project II, Spring 2024
Advisor: Assoc. Prof. Dr. Ömer ÇETİN

GLOWG: Personalized Skincare Powered by AI
Product Manual

Revision 1.0
04.05.2025

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

Revision History

Revision	Date	Explanation
1.0	04.05.2025	Initial Product Manual

Table of Contents

Revision History 2
Table of Contents 3
1. Introduction 4
2. GlowGenie Software Subsystem Implementation 4
2.1. Source Code and Executable Organization 4
2.2. Software DevelopmentTools 5
2.3. Hardware and System Software Platform 6
3. GlowGenie Software Subsystem Testing 6
4. GlowGenie Installation, Configuration and Operation 7
4.1. Backend Installation 7
4.2. Frontend Installation 8
5.3. Database Setup and Configuration 8
4.4. Operation Flow 9
4.5. Developer and Maintenance Notes 9
References 10

1. Introduction

This document presents the Product Manual for **GlowGenie**, an AI-powered skincare recommendation system developed within the scope of the **COMP4910 Senior Design Project**. The primary objective of this manual is to comprehensively document the final implementation of the software, including its architectural structure, development environment, testing strategy, installation procedure, and operational guidelines.

GlowGenie is designed to offer users a personalized skincare experience by utilizing artificial intelligence to identify their skin type, evaluate cosmetic product ingredients, and generate suitable product recommendations. The platform gathers user input through a structured questionnaire, analyzes it with machine learning models, and assesses ingredient compatibility using natural language processing models such as DeepSeek. Based on this analysis, the system delivers customized product suggestions aligned with the user's skin characteristics and possible sensitivities.

This document builds upon the prior **Requirements Specification Document (RSD)** and **Detailed Software Design Document (DSD)**. While the RSD and DSD focus on the conceptual and architectural design of the system, this manual presents a detailed overview of the implemented features and operational logic of the GlowGenie application.

The application enables secure user registration and login, real-time profile updates, AI-based skin analysis, and compatibility checks for skincare product ingredients. It supports users in navigating the product catalog through category-based filtering and enables informed decision-making by identifying which ingredients are suitable or potentially harmful. The system also provides mechanisms for password recovery and user verification via email-based confirmation.

In the sections that follow, the software subsystem, development tools, testing procedures, and installation steps will be discussed in detail. This manual serves as a reference for developers, testers, and stakeholders evaluating or extending the GlowGenie system.

2. GlowGenie Software Subsystem Implementation

2.1. Source Code and Executable Organization

The source code of the GlowGenie system is organized into a modular and layered structure, separating concerns across the backend, machine learning models, and frontend components. The backend is developed using Python with the Flask framework and implements a RESTful API that handles HTTP requests and responses. Core functionalities such as user authentication, profile management, product retrieval, and ingredient evaluation are encapsulated within dedicated service modules (e.g., AuthService, RecommendationService, OpenAIRoutes). Each endpoint is routed through a centralized controller (AppController) that delegates tasks to these underlying services.

The machine learning subsystem is composed of two main model classes: SkinTypePredictor, which classifies users' skin types based on questionnaire inputs using a regression model, and ProductSuitabilityModel, which utilizes a random forest algorithm to determine product compatibility based on binary-encoded ingredient data. These models are trained offline and loaded during runtime for inference.

The frontend is implemented using React.js and communicates with the backend through secure API calls. It presents user-friendly interfaces for skin type testing, product browsing, feedback submission, and personalized recommendations. All forms and inputs are validated both client-side and server-side to ensure data integrity and security.

The codebase is divided into clearly defined folders such as /backend, /ml_models, and /frontend, each containing relevant modules, configuration files, and static assets. Database interactions are abstracted within service layers, ensuring a clean separation between application logic and data persistence. Configuration settings, such as database connection strings, API keys, and environment-specific variables, are stored in .env and config.py files to facilitate portability and deployment flexibility.

All code is maintained in a version-controlled repository on GitHub, with branches used for development, testing, and production integration. While the design largely adheres to the structure proposed in the DSD document, minor adjustments were made during implementation to improve modularity and maintainability. The source code includes properly structured comments and inline documentation, written in accordance with professional software engineering practices.

2.2. Software DevelopmentTools

The implementation of the GlowGenie application utilized a comprehensive set of software tools distributed across macOS and Windows environments. This distributed setup was adopted to align each development task with the most compatible and efficient platform. Tools and frameworks were carefully selected based on their suitability for modular full-stack development, AI integration, and real-time user interaction.

The backend and database subsystems were developed on **macOS Ventura**, using **Python 3.10.6** and the **Flask 2.2.2** microframework, both installed via the official python.org and pip. The Flask backend served as a RESTful API layer, facilitating communication between the user interface, the PostgreSQL database, and integrated machine learning modules. A virtual environment was configured using Python's built-in venv module to ensure dependency isolation and reproducibility. Backend code was written and managed using **Visual Studio Code 1.87.2** for macOS, with relevant extensions for Python and database interaction.

The database layer was implemented using **PostgreSQL 15**, also on macOS, and managed with **pgAdmin 4**, which enabled schema design, data inspection, and administrative control over stored procedures and backup operations. The backend and database were hosted locally on the same macOS machine to ensure low-latency and secure data exchange during development.

The machine learning components were developed in a **Windows-based environment**, using **Jupyter Notebook 6.5.4** for model prototyping. **scikit-learn 1.2.2** was used to implement the SkinTypePredictor and ProductSuitabilityModel classes, and **SHAP 0.41.0** was incorporated to provide interpretability into feature influence on model outcomes. Trained models were exported via joblib and transferred to the backend system on macOS for deployment.

Frontend development was carried out in a separate **Windows environment** using **React 18.2.0**, initialized with create-react-app. HTTP communication with the backend API was handled using **Axios 1.3.4**, and interface styling was achieved through **Bootstrap 5.2.3**. The frontend environment was managed using **Node.js 18.15.0** and **npm 9.5.0**, both installed from nodejs.org. Visual Studio Code (Windows edition) served as the primary IDE for interface development, and browser-based testing was conducted locally over specified ports.

API endpoints were tested and monitored using **Swagger 0.9.7.1**, while version control was maintained throughout the project using **Git 2.42.0**. All source code was hosted on **GitHub** within a private repository, and branching strategies were adopted to organize feature development, testing, and stable release workflows.

Although no custom-built third-party tools were developed, all core service modules — such as UserService, RecommendationService, and OpenAIIngredientAnalyzer — were implemented from scratch by the team, following principles of modularity, reusability, and separation of concerns.

This platform-aware toolchain enabled efficient collaboration across systems while maintaining a cohesive and testable architecture throughout development and integration phases.

2.3. Hardware and System Software Platform

The development and testing of the GlowGenie system were carried out on both macOS and Windows platforms, with different subsystems implemented on each based on performance needs and developer environment compatibility. This hybrid approach allowed for optimized development across all components of the system, including backend logic, database operations, machine learning models, and frontend interfaces. The backend and database subsystems were developed on macOS Ventura 13.x, running on a MacBook Air with an Apple M1 chip, 8 GB of unified memory, and 256 GB SSD storage. The backend was implemented using Python 3.10.6 and Flask 2.2.2, and was hosted locally for testing purposes. PostgreSQL 15 was installed natively on macOS and managed via pgAdmin 4, which provided a graphical interface for schema design, data queries, and administrative operations. The macOS Terminal, using the zsh shell, was used for environment activation, package installation via pip, and execution of backend services. This configuration ensured smooth integration between the backend logic and the local database service. The frontend and machine learning components were developed and tested on a Windows 11 environment running on a laptop equipped with an 11th Gen Intel Core i7-1165G7 processor (2.80 GHz), 16 GB RAM, and 1 TB SSD storage, using a 64-bit architecture. React 18.2.0 and supporting libraries such as Axios and Bootstrap were used to build and test the user interface. The machine learning models — including SkinTypePredictor and ProductSuitabilityModel — were developed in PyCharm using scikit-learn 1.2.2, and model interpretability was supported by SHAP 0.41.0. The trained models were exported using joblib and transferred to the backend on macOS for integration and runtime prediction. All code development was performed using Visual Studio Code 1.87.2 on both platforms, with relevant extensions installed for Python, JavaScript/React, and SQL. API development and endpoint validation were performed using Postman 10.19.5, ensuring accurate testing across environments. The project's hardware and software setup enabled distributed development while maintaining consistent behavior across platforms. The system remains platform-independent at the architectural level, and can be deployed on cloud-based services or containerized environments such as Docker in future versions.

3. GlowGenie Software Subsystem Testing

Throughout the development of the GlowGenie application, a systematic and layered testing approach was followed to validate both individual modules and the complete system workflow. Testing activities were carried out across backend services, machine learning models, frontend components, and database interactions to ensure that all functionalities operated as expected.

The core backend endpoints—such as user registration, login, logout, profile updates, product listing, and personalized recommendation generation—were tested using **Swagger**, where each API response was verified for both valid and invalid input scenarios. These endpoints were connected to

a PostgreSQL database running locally, and backend–database integration was validated through successful data insertions, updates, and queries. For instance, the profile update endpoint was tested by submitting user inputs from the frontend interface and confirming changes via database queries in **pgAdmin 4**.

The machine learning models (SkinTypePredictor and ProductSuitabilityModel) were evaluated in an offline environment using **Py Charm**, where prediction accuracy was measured using test datasets. The trained models were later integrated into the Flask backend and accessed through API endpoints, which were also tested via **Swagger** to confirm their runtime behavior. Predictions were confirmed to match expected outcomes, and SHAP values were analyzed to verify model interpretability.

The user interface developed in React was manually tested in a browser environment. Frontend forms for login, profile updates, and skin test submission were evaluated for layout integrity, input handling, and communication with the backend. Browser developer tools were used to observe network requests and verify that the frontend correctly displayed backend responses and handled errors gracefully.

Some functionalities, such as the **password reset and email confirmation system**, were partially implemented but not fully tested due to time constraints. These modules currently have placeholder logic in place and are intended for further testing in future versions. Additionally, features related to administrative controls or internal data management were not implemented in this version of the project and are scheduled for later phases.

During testing, logs and error traces were monitored directly from the terminal and development environment to capture any anomalies. Overall, the majority of critical features—especially those visible to the end-user—were tested and confirmed to work correctly under normal usage conditions.

4. GlowGenie Installation, Configuration and Operation

This section outlines the complete installation procedure, configuration steps, and operational flow of the GlowGenie system, which consists of a Python-based backend server, a React-based frontend client, a PostgreSQL relational database, and integrated machine learning components. The system is designed to run on a local development environment but remains flexible for future deployment to remote or containerized platforms.

4.1. Backend Installation

The backend was developed using **Python 3.10.6** and the **Flask** framework. To install and run the backend:

1.Clone the Project Repository

```
git clone https://github.com/your-org/glowgenie.git
cd glowgenie/backend
```

2.Set up Virtual Environment

Create and activate a virtual environment to isolate dependencies:

```
python3 -m venv venv
source venv/bin/activate  # macOS/Linux
.\venv\Scripts\activate    # Windows
```

3.Install Backend Dependencies

```
pip install -r requirements.txt
```

4.Prepare the Environment Variables

Create a .env file in the root of the backend/ folder with the following content:

```
FLASK_APP=app.py
FLASK_ENV=development
SECRET_KEY=your-secret-key
DATABASE_URL=postgresql://username:password@localhost:5432/glowgenie
```

5.Start the Backend Server

After setting up the environment and database (see 5.3), run:

```
flask run
```

The backend will start at http://127.0.0.1:5000.

4.2. Frontend Installation

The frontend was developed using **React 18.2.0**, **Node.js 18.15.0**, and **npm 9.5.0**.

1. **Navigate to the Frontend Directory**

   ```
   cd ../frontend
   ```

2. **Install Node.js Dependencies**

   ```
   npm install
   ```

3. **Start the React Development Server**

   ```
   npm start
   ```

The UI will be accessible at http://localhost:3000. It communicates with the backend through RESTful API calls defined in Axios-based service modules.

4. **Update Axios Base URLs if Needed**

In frontend/src/config/api.js or equivalent, ensure the backend URL matches:

```
const BASE_URL = "http://localhost:5000";
```

4.3. Database Setup and Configuration

The application uses **PostgreSQL 15** to manage user data, skin test results, product information, and recommendation records.

1. **Install PostgreSQL (via postgresql.org)**
2. **Create a Database**
   Open **pgAdmin 4**, create a new database named `glowgenie`.
3. **Run Schema Scripts**

   Use SQL tab in pgAdmin to run provided schema creation script:

-- Example: users table

```
CREATE TABLE users (
   user_id SERIAL PRIMARY KEY,
   email VARCHAR(255) UNIQUE NOT NULL,
   password VARCHAR(255) NOT NULL,
  skin_type_id INT,
  skin_tone_id INT
);
```

4. **Connect to the Database**

   Ensure that the connection string in .env matches:

   postgresql://postgres:yourpassword@localhost:5432/glowgenie

4.4. Operation Flow

Once the backend, frontend, and database are all running correctly:

- The user launches the UI at localhost:3000
- They register a new account and log in
- The system directs them to a skin test form
- Their responses are analyzed by the integrated ML model (SkinTypePredictor)
- Their skin type is stored, and product recommendations are fetched via /recommend endpoint
- Users can view ingredient compatibility for any product and filter by category
- Their actions, preferences, and feedback are stored in the database
- Incompatible products (due to allergens or skin mismatch) are flagged in real time

Throughout usage, form validation, input sanitization, and error handling ensure a smooth and secure interaction. If users enter invalid data, they receive contextual feedback both from frontend and backend validators.

4.5. Developer and Maintenance Notes

- Ensure .env and database credentials are correct before running flask run
- Check PostgreSQL service is active and listening on default port (5432)
- Use browser dev tools (Console & Network tab) to debug frontend–backend communication
- Backend logs (terminal) will display request traces and error stacks
- Database records can be inspected and modified via pgAdmin 4

The system is currently optimized for local development but remains structurally compatible with future cloud deployment (e.g., Render, Heroku) or Docker-based containerization.

References

1. Python Software Foundation. (2023). *Python 3.10.6 Documentation*. Retrieved from https://docs.python.org/3.10/
2. Flask Documentation. (2023). *Flask 2.2.x*. Retrieved from https://flask.palletsprojects.com/
3. PostgreSQL Global Development Group. (2023). *PostgreSQL 15 Documentation*. Retrieved from https://www.postgresql.org/docs/15/
4. pgAdmin Team. (2023). *pgAdmin 4 Documentation*. Retrieved from https://www.pgadmin.org/docs/pgadmin4/latest/
5. React Contributors. (2023). *React 18 Documentation*. Retrieved from https://reactjs.org/docs/getting-started.html
6. Node.js Foundation. (2023). *Node.js 18 Documentation*. Retrieved from https://nodejs.org/en/docs/
7. scikit-learn Developers. (2023). *scikit-learn 1.2.2 Documentation*. Retrieved from https://scikit-learn.org/stable/
8. Lundberg, S. M., & Lee, S. I. (2017). *A Unified Approach to Interpreting Model Predictions*. In Proceedings of the 31st International Conference on Neural Information Processing Systems (NIPS'17). SHAP Library Documentation: https://shap.readthedocs.io/
9. OpenAI. (2023). *DeepSeek API Documentation*. Retrieved from https://api-docs.deepseek.com/
10. Git Documentation. (2023). *Git 2.42 User Manual*. Retrieved from https://git-scm.com/docs
11. Visual Studio Code. (2023). *VS Code Docs*. Retrieved from https://code.visualstudio.com/docs
12. Swagger API Platform. *Swagger API Testing*. Retrieved from https://swagger.io/
13. GlowGenie-RSD-2025-01-12.doc (Requirements Specification Document).
14. GlowGenie-DSD-2025-05-01.doc (Detailed Software Design Document).

APPENDIX D: PROJECT MANAGEMENT DOCUMENTS

APPENDIX D1: PROJECT PLAN

COMP 4910 GLOWG: GlowGenie, Project Plan, 02.06.2025, v1.0																
Task No	Task Name	Weeks														Any Additional Notes
		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
1	Project Planning & Final Requirements Confirmation	X	X	X	X										X	X	
2	System Implementation Based on Literature Findings					X	X	X									
3	DeepSeek API Integration & Optimization					X	X										
4	Data Collection & Preprocessing Pipeline Development	X	X							X	X						
5	Machine Learning Model Training & Integration										X	X	X	X	X		
6	UI Design Implementation & Frontend Development					X	X	X	X	X	X	X	X	X	X	X	
7	Database Modeling & PostgreSQL Deployment		X	X									X	X	X		
8	Flask Backend Development & RESTful API Integration					X	X	X	X	X	X	X	X		X	X	
9	Final Presentation, Poster Design & Deliverables Preparation														X	X	

APPENDIX D2: PROJECT EFFORT LOGS FOR EACH TEAM MEMBER-

COMP 4910/4920 Project Effort Log, Project Code: GLOWG, 02.06.2025, v1.0 Team Member: Gizem Tanış			
Week	**Dates**	**Work Done in Some Detail**	**Total Hours Spent**
Week 1	24/02/2025 - 02/03/2025	Project topic review, initial planning, team meeting, GitHub repository setup	3
Week 2	03/03/2025 - 09/03/2025	I worked on designing the database schema using PostgreSQL and created the Entity-Relationship (ER) diagram to define the structure of tables and their relationships.	7
Week 3	10/03/2025 - 16/03/2025	I implemented the tables in PostgreSQL using pgAdmin and inserted test data. I also started researching skincare products to identify which ones are suitable for specific skin types.	7
Week 4	17/03/2025 - 23/03/2025	I identified the suitable products for different skin types and prepared the product dataset to be used in the project.	8
Week 5	24/03/2025 - 30/03/2025	I tested the user interface and gathered feedback to suggest small improvements for a better user experience.	10

Week 6	31/03/2025 - 06/04/2025	I helped fix some frontend issues in the React.js part and joined my teammates in small tests to ensure proper functionality.	11
Week 7	07/04/2025 - 13/04/2025	I created relationships between database tables using foreign keys and tested sample data flow with the React frontend.	12
Week 8	14/04/2025 - 20/04/2025	I contributed to the second version of the DSD (Detailed Software Design) document by preparing the sections related to the database structure.	14
Week 9	21/04/2025 - 27/04/2025	I checked the database connections and added clear error messages to handle potential connection issues.	10
Week 10	28/04/2025 - 04/05/2025	I added validation rules to user input forms to prevent invalid data entries.	11
Week 11	05/05/2025 - 11/05/2025	I worked on the Product Manual report and conducted detailed research while preparing the content.	10
Week 12	12/05/2025 - 18/05/2025	I brainstormed ideas, conducted research, and designed a suitable poster for our capstone project presentation.	10

Week 13	19/05/2025 - 25/05/2025	We participated in the “GBYF” competition. During this process, I worked on the project presentation, content preparation, and visual design.	15
Week 14	26/05/2025 - 01/06/2025	I prepared individual datasets for each product to compare its ingredients with others and organized the matching details carefully.	12
Week 15	02/06/2025 - 08/06/2025	I checked for missing or incorrect data and made necessary adjustments by identifying what needed to be added or fixed.	10
Week 17	09/05/2025 - 15/06/2025		
Week 18	16/06/2025 - 22/05/2025		
Total Effort in Man-Hours			**150,00**
Total Effort in Man-Days			**18,75**
Notes:			
1. This table shows the team member project effort. One Man-Day is Eight Man-Hours. 2. Each team member must fill out the form periodically (preferably at the end of the week of any work done). 3. Each filled-out table must be emailed at the end of each month to a selected Project Member (cc to Project Advisor), who will produce a consolidated table.			
Form: GLOWG-Project Effort Log-2025-06-02-v1.0-Tanış.xlsx			

COMP 4910/4920 Project Effort Log, Project Code: GLOWG, 02.06.2025, v1.0 Team Member: Ece Topuz			
Week	**Dates**	**Work Done in Some Detail**	**Total Hours Spent**
Week 1	24/02/2025 - 02/03/2025	Sharing tasks within the team, initial meetings. Clarifying project goals and user requirements. Created Github repository	3
Week 2	03/03/2025 - 09/03/2025	Planning user flows and page transitions on paper. Creating basic user scenarios and workflows.	8
Week 3	10/03/2025 - 16/03/2025	Setting up the project structure: Starting a React project, planning the folder structure, installing key dependencies.	6
Week 4	17/03/2025 - 23/03/2025	Creating the basic structures of the authorization (Auth) pages: Preparation of Register, Login, Forgot Password screens according to the component structure.	10
Week 5	24/03/2025 - 30/03/2025	Adding styles to Auth components: responsive form structures, using Bootstrap, basic validation processes.	10
Week 6	31/03/2025 - 06/04/2025	Creation of Navbar component, preparation of menu transitions, responsive design. React Router installation for page redirects.	11
Week 7	07/04/2025 - 13/04/2025	Developing LoginForm and OptionsPanel components. Ensuring that these components behave dynamically with user interaction.	12

Week 8	14/04/2025 - 20/04/2025	Development of ProfileCard and SkinTypePopup components. Dynamicization of content according to data received from the user. Preparation of DSD document v.2.	15
Week 9	21/04/2025 - 27/04/2025	Product detail page development: Card structures, visual layouts, in-page layouts.	8
Week 10	28/04/2025 - 04/05/2025	Product suitability analysis module (RightForMe) development: Filtering logic to work with user skin type and content matching.	12
Week 11	05/05/2025 - 11/05/2025	Creating the SkinTypeTest page: Obtaining skin test data, local state and form management.	11
Week 12	12/05/2025 - 18/05/2025	Modal and popup management: Opening and closing of SkinTypePopup and RightForMe modals, transition animations and button behaviors.	9
Week 13	19/05/2025 - 25/05/2025	Preparation of Product Manual. Interface checks for GBYF competition, demo presentation rehearsal.	15
Week 14	26/05/2025 - 01/06/2025	Full integration of version control system: Git branching strategy (main/dev/feature), commit rules, sharing via GitHub and responsive adjustments throughout the application, UI improvements, final touches to the design. Testing of page transitions.	14
Week 15	02/06/2025 - 08/06/2025	Testing all user flows: form controls, popups, redirects. Detection and fixing of errors.	10
Week 17	09/05/2025 - 15/06/2025		
Week 18	16/06/2025 - 22/05/2025		
Total Effort in Man-Hours			**154,00**

Total Effort in Man-Days			**19,25**
Notes:			
1. This table shows the team member project effort. One Man-Day is Eight Man-Hours. **2. Each team member must fill out the form periodically (preferably at the end of the week of any work done).** **3. Each filled-out table must be emailed at the end of each month to a selected Project Member (cc to Project Advisor), who will produce a consolidated table.**			
Form: GLOWG-Project Effort Log-2025-06-02-v1.0-Topuz.xlsx			

COMP 4910/4920 Project Effort Log, Project Code: GLOWG, 02.06.2025, v1.0
Team Member: İrem Demir

Week	Dates	Work Done in Some Detail	Total Hours Spent
Week 1	24/02/2025 - 02/03/2025	Created the GitHub repository and set up the initial structure for the project.	3
Week 2	03/03/2025 - 09/03/2025	Initialized the Flask backend project and installed all necessary libraries and dependencies.	6
Week 3	10/03/2025 - 16/03/2025	Started backend development and planned the core architecture of the system.	8
Week 4	17/03/2025 - 23/03/2025	Researched how to integrate OpenAI with the Flask backend and React frontend.	8
Week 5	24/03/2025 - 30/03/2025	Explored the DeepSeek API and successfully connected it to the backend using a free API key.	10
Week 6	31/03/2025 - 06/04/2025	Implemented CRUD operations to manage and retrieve user information from the database to Flask.	14
Week 7	07/04/2025 - 13/04/2025	Created and connected category-related tables such as skin type, skin tone, category, products using CRUD operations.	14

Week 8	14/04/2025 - 20/04/2025	Integrated the backend with the React frontend to enable seamless communication between both sides, and started preparing the DSD v2 documentation.	17
Week 9	21/04/2025 - 27/04/2025	Developed login and registration functionalities and ensured proper user flow.	12
Week 10	28/04/2025 - 04/05/2025	Implemented JWT-based authentication for secure login sessions, and the product manual was completed.	12
Week 11	05/05/2025 - 11/05/2025	Established a connection between the backend and the ML model, and integrated DeepSeek API to assess product content and store safety results in the database.	10
Week 12	12/05/2025 - 18/05/2025	Enhanced product filtering functionality by dynamically retrieving skin type and tone options from the backend and integrating them into the register form.	12
Week 13	19/05/2025 - 25/05/2025	Developed and styled email and password verification popups with countdown and resend features for both registration and password reset flows. Preparing Product Manual.	13
Week 14	26/05/2025 - 01/06/2025	Built the /predict API for product suitability analysis using the ML model, integrated it into the React frontend for real-time user feedback, and prepared the project presentation, poster, and final deliverables.	12
Week 15	02/06/2025 - 08/06/2025	Implemented token-based profile retrieval for logged-in users and finalized the skin type test feature with secure backend communication.	12
Week 17	09/05/2025 - 15/06/2025		

Week 18	16/06/2025 - 22/05/2025		
Total Effort in Man-Hours			**163,00**
Total Effort in Man-Days			**20,38**
Notes:			
1. This table shows the team member project effort. One Man-Day is Eight Man-Hours. 2. Each team member must fill out the form periodically (preferably at the end of the week of any work done). 3. Each filled-out table must be emailed at the end of each month to a selected Project Member (cc to Project Advisor), who will produce a consolidated table.			
Form: GLOWG-Project Effort Log-2025-06-02-v1.0-Demir.xlsx			

COMP 4910/4920 Project Effort Log, Project Code: GLOWG, 02.06.2025, v1.0
Team Member: Ceren Sude Yetim

Week	Dates	Work Done in Some Detail	Total Hours Spent
Week 1	24/02/2025 - 02/03/2025	Helped create and structure the GitHub repository (initial commits, folders), made some arrangements with the team.	3
Week 2	03/03/2025 - 09/03/2025	Collected raw product–ingredient data from sources and got the needed data for skin type test from the survey answers.	7
Week 3	10/03/2025 - 16/03/2025	Cleaned and preprocessed the dataset (handled missing values, normalization).	8
Week 4	17/03/2025 - 23/03/2025	Performed feature engineering and finalized the dataset schema.	12
Week 5	24/03/2025 - 30/03/2025	Began coding the SkinTypePredictor (model definition, data pipeline).	10
Week 6	31/03/2025 - 06/04/2025	Trained and validated SkinTypePredictor offline (accuracy checks, refinements).	12
Week 7	07/04/2025 - 13/04/2025	Advanced SkinTypePredictor training; tested API endpoints via Flasgger.	12

Week 8	14/04/2025 - 20/04/2025	Started coding the ProductSuitabilityModel class (data ingestion, basic logic).	10
Week 9	21/04/2025 - 27/04/2025	Continued ProductSuitabilityModel development and performed initial training.	12
Week 10	28/04/2025 - 04/05/2025	Advanced ProductSuitabilityModel training; tested API endpoints via Flasgger.	10
Week 11	05/05/2025 - 11/05/2025	Integrated SHAP for ProductSuitabilityModel and ran interpretability analyses.	11
Week 12	12/05/2025 - 18/05/2025	Updated data as needed and applied final refinements to both models.	9
Week 13	19/05/2025 - 25/05/2025	Advanced ProductSuitabilityModel training; tested API endpoints via Flasgger.	10
Week 14	26/05/2025 - 01/06/2025	Fixed remaining issues in the ProductSuitabilityModel and assisted with backend tasks.	10
Week 15	02/06/2025 - 08/06/2025	Worked on the final deliverables and on the ProductSuitability Model.	12
Week 17	09/05/2025 - 15/06/2025		
Week 18	16/06/2025 - 22/05/2025		
Total Effort in Man-Hours			**148,00**
Total Effort in Man-Days			**18,50**

Notes:

1. This table shows the team member project effort. One Man-Day is Eight Man-Hours.
2. Each team member must fill out the form periodically (preferably at the end of the week of any work done).
3. Each filled-out table must be emailed at the end of each month to a selected Project Member (cc to Project Advisor), who will produce a consolidated table.

Form: GLOWG-Project Effort Log-2025-06-02-v1.0-Yetim.xlsx			

APPENDIX D3: PROJECT EFFORT LOG- CONSOLIDATED

COMP 4910/4920 Project Effort Log, Project Code: GLOWG, 02.06.2025, v1.0									
Week	Dates	İrem Demir		Ece Topuz		Ceren Sude Yetim		Gizem Tanış		Total Weekly Effort in Man-Hours
		Work Done	Total Hours Spent	Work Done	Total Hours Spent	Work Done	Total Hours Spent	Work Done	Total Hours Spent	
Week 1	24/02/2025 - 02/03/2025	Created the GitHub repository and set up the initial structure for the project.	3	Sharing tasks within the team, initial meetings. Clarifying project goals and user requirements. Created Github repository	3	Helped create and structure the GitHub repository (initial commits, folders), made some arrangements with the team.	3	Project topic review, initial planning, team meeting, GitHub repository setup	3	12,00
Week 2	03/03/2025 - 09/03/2025	Initialized the Flask backend project and installed all necessary libraries and dependencies.	6	Planning user flows and page transitions on paper. Creating basic user scenarios and workflows.	8	Collected raw product–ingredient data from sources and got the needed data for skin type test from the survey answers.	7	I worked on designing the database schema using PostgreSQL and created the Entity-Relationship (ER) diagram to define the structure of tables and their relationships.	7	28,00

Week 3	10/03/2025 - 16/03/2025	Started backend development and planned the core architecture of the system.	8	Setting up the project structure: Starting a React project, planning the folder structure, installing key dependencies.	6	Cleaned and preprocessed the dataset (handled missing values, normalization).	8	I implemented the tables in PostgreSQL using pgAdmin and inserted test data. I also started researching skincare products to identify which ones are suitable for specific skin types.	7	29,00
Week 4	17/03/2025 - 23/03/2025	Researched how to integrate OpenAI with the Flask backend and React frontend.	8	Creating the basic structures of the authorization (Auth) pages: Preparation of Register, Login, Forgot Password screens according to the component structure.	10	Performed feature engineering and finalized the dataset schema.	12	I identified the suitable products for different skin types and prepared the product dataset to be used in the project.	8	38,00

Week 5	24/03/2025 - 30/03/2025	Explored the DeepSeek API and successfully connected it to the backend using a free API key.	10	Adding styles to Auth components: responsive form structures, using Bootstrap, basic validation processes.	10	Began coding the SkinTypePredictor (model definition, data pipeline).	10	I tested the user interface and gathered feedback to suggest small improvements for a better user experience.	10	40,00
Week 6	31/03/2025 - 06/04/2025	Implemented CRUD operations to manage and retrieve user information from the database to Flask.	14	Creation of Navbar component, preparation of menu transitions, responsive design. React Router installation for page redirects.	11	Trained and validated SkinTypePredictor offline (accuracy checks, refinements).	12	I helped fix some frontend issues in the React.js part and joined my teammates in small tests to ensure proper functionality.	11	48,00
Week 7	07/04/2025 - 13/04/2025	Created and connected category-related tables such as skin type, skin tone, category, products using CRUD operations.	14	Developing LoginForm and OptionsPanel components. Ensuring that these components behave dynamically with user interaction.	12	Advanced SkinTypePredictor training; tested API endpoints via Flasgger.	12	I created relationships between database tables using foreign keys and tested sample data flow with the React frontend.	12	50,00

Week 8	14/04/2025 - 20/04/2025	Integrated the backend with the React frontend to enable seamless communication between both sides, and started preparing the DSD v2 documentation.	17	Development of ProfileCard and SkinTypePopup components. Dynamicization of content according to data received from the user. Preparation of DSD document v.2.	15	Started coding the ProductSuitabilityModel class (data ingestion, basic logic).	10	I contributed to the second version of the DSD (Detailed Software Design) document by preparing the sections related to the database structure.	14	56,00
Week 9	21/04/2025 - 27/04/2025	Developed login and registration functionalities and ensured proper user flow.	12	Product detail page development: Card structures, visual layouts, in-page layouts.	8	Continued ProductSuitabilityModel development and performed initial training.	12	I checked the database connections and added clear error messages to handle potential connection issues.	10	42,00
Week 10	28/04/2025 - 04/05/2025	Implemented JWT-based authentication for secure login sessions, and the product manual was completed.	12	Product suitability analysis module (RightForMe) development: Filtering logic to work with user skin type and content matching.	12	Advanced ProductSuitabilityModel training; tested API endpoints via Flasgger.	10	I added validation rules to user input forms to prevent invalid data entries.	11	45,00

Week 11	05/05/2025 - 11/05/2025	Established a connection between the backend and the ML model, and integrated DeepSeek API to assess product content and store safety results in the database.	10	Creating the SkinTypeTest page: Obtaining skin test data, local state and form management.	11	Integrated SHAP for ProductSuitabilityModel and ran interpretability analyses.	11	I worked on the Product Manual report and conducted detailed research while preparing the content.	10	42,00
Week 12	12/05/2025 - 18/05/2025	Enhanced product filtering functionality by dynamically retrieving skin type and tone options from the backend and integrating them into the register form.	12	Modal and popup management: Opening and closing of SkinTypePopup and RightForMe modals, transition animations and button behaviors.	9	Updated data as needed and applied final refinements to both models.	9	I brainstormed ideas, conducted research, and designed a suitable poster for our capstone project presentation.	10	40,00
Week 13	19/05/2025 - 25/05/2025	Developed and styled email and password verification popups with countdown and resend features for both registration and password reset flows. Preparing Product Manual.	13	Preparation of Product Manual. Interface checks for GBYF competition, demo presentation rehearsal.	15	Advanced ProductSuitabilityModel training; tested API endpoints via Flasgger.	10	We participated in the “GBYF” competition. During this process, I worked on the project presentation, content preparation, and visual design.	15	53,00

Week 14	26/05/2025 - 01/06/2025	Built the /predict API for product suitability analysis using the ML model, integrated it into the React frontend for real-time user feedback, and prepared the project presentation, poster, and final deliverables.	12	Full integration of version control system: Git branching strategy (main/dev/ feature), commit rules, sharing via GitHub and responsive adjustments throughout the application, UI improvements, final touches to the design. Testing of page transitions.	14	Fixed remaining issues in the ProductSuitabilityModel and assisted with backend tasks.	10	I prepared individual datasets for each product to compare its ingredients with others and organized the matching details carefully.	12	48,00
Week 15	02/06/2025 - 08/06/2025	Implemented token-based profile retrieval for logged-in users and finalized the skin type test feature with secure backend communication.	12	Testing all user flows: form controls, popups, redirects. Detection and fixing of errors.	10	Worked on the final deliverables and on the ProductSuitability Model.	12	I checked for missing or incorrect data and made necessary adjustments by identifying what needed to be added or fixed.	10	44,00
Week 17	09/05/2025 - 15/06/2025									0,00
Week 18	16/06/2025 - 22/05/2025									0,00

Total Effort in Man-Hours			**163,00**		**154,00**		**148,00**		**150,00**	**615,00**
Total Effort in Man-Days			**20,38**		**19,25**		**18,50**		**18,75**	**76,88**

Notes:

1. This table shows the consolidated project effort, based on project effort tables prepared by each team member
2. Replace Team Member i with team member name and lastname, then fill out one column for each team member