Development of a Thermal Water-based Cosmetic Product Recommendation System based on a Facial Image Processing approach

Guilherme de Medeiros Tonello

Dissertation presented to the School of Technology and Management of Bragança to obtain the master's degree in Informatics within the scope of the double degree program with the Federal University of Technology – Paraná

Supervisors:

Prof. PhD Maria João Varanda

Prof. PhD Paulo Alves

Prof. PhD André Roberto Ortoncelli

Bragança

June 2024

Development of a Thermal Water-based Cosmetic Product Recommendation System based on a Facial Image Processing approach

Guilherme de Medeiros Tonello

Dissertation presented to the School of Technology and Management of Bragança to obtain the master's degree in Informatics within the scope of the double degree program with the Federal University of Technology – Paraná

Supervisors:

Prof. PhD Maria João Varanda

Prof. PhD Paulo Alves

Prof. PhD André Roberto Ortoncelli

Bragança

June 2024

Dedication

This work is not just a study, it's a journey coming to an end and a dream come true for me and my family. I therefore dedicate this work to all those who have been present and supported me throughout this journey.

To my mother, Sirley de Medeiros, my sister, Mylena de Medeiros Tonello and my father, Valdecir Tonello, a heartfelt thank you for all the support you have always given me.

To my friends, both those who have been with me from an early age and those I met when I arrived in Portugal, thank you for supporting me and being my companion on this path.

And finally, to my partner, advisor and friend, Eloren Meurer, you know that none of this would have been possible if you hadn't been by my side to support and encourage me throughout this time. This work is as much yours as it is mine, so a heartfelt thank you for always being there for me.

Abstract

Skincare has become a constant demand among the population, who are increasingly concerned about their health. Furthermore, environmental issues arouse the interest of the masses in natural and sustainable products. This project proposes an approach for recommending thermal-based products based on a set of information provided by the user, combined with the results of computer vision algorithms (to identify the age and occurrence of wrinkles on the user's forehead). A list of recommended products is generated based on the profile determined for the user. To predict wrinkles, for each facial image sent by the user, we apply a pre-processing step that segments and prepares the region of interest, which a CNN will process. As a CNN, we used the VGG16 architecture trained using a transfer learning and fine-tuning strategy, which improved the results obtained, reaching an accuracy of 92% in classifying wrinkles. An algorithm provided by the Deepface tool is used to predict the user's age, based on the sent picture. Which is crossed with the user's information to determine a level of aging in order to improve the quality of the the recommended products.

Keywords: Deep learning; Wrinkle Detection; Recommendation System; Convolutional Neural Network; VGG16.

Resumo

Os cuidados com a pele tornaram-se uma procura constante entre a população, que se preocupa cada vez mais com a sua saúde. Para além disso, as questões ambientais despertam o interesse das massas por produtos naturais e sustentáveis. Este projeto propõe uma abordagem para a recomendação de produtos baseado em água termal, a partir de um conjunto de informações fornecidas pelo utilizador, combinadas com os resultados de algoritmos de visão computacional (para identificar a idade e a ocorrência de rugas na testa do usuario). Para prever as rugas, para cada imagem facial enviada pelo usuario, aplica-se um passo de pré-processamento que segmenta e prepara a região de interesse, que será processada por uma Rede Neural Convolucional. Utilizou-se a arquitetura VGG16 como base para treinar o modelo através de uma estratégia baseada em transferência de aprendizado e ajuste fino, que melhorou os resultados obtidos, atingindo uma precisão de 92% na classificação das rugas. Um algoritmo fornecido pela ferramenta Deepface é utilizado para prever a idade do utilizador, com base na fotografia enviada. Este algoritmo é cruzado com as informações do usuário para determinar um nível de envelhecimento, melhorando a precisão dos produtos recomendados.

Keywords: Aprendizado Profundo; Detecçao de Rugas; Sistema de Recomendação; Rede Neural Convolucional; VGG16.

Acknowledgment

I would first like to thank my supervisors Prof. PhD Maria João Varanda, Prof. PhD Paulo Alves and Prof. PhD André Roberto Ortoncelli for their support and guidance throughout this project. I must express my total admiration and respect for you and your work, without you this research would not have been possible.

I would also like to thank the Polytechnic Institute of Bragança and the Federal Technological University of Paraná, which gave me the opportunity to carry out this work.

Thanks to all my teachers, colleagues and friends who have been with me throughout my degree, each of you has your own contribution to the final result of this work. In particular, I would like to thank my friend and colleague Arthur Sosnowski, who gave me great support in this study.

This work was supported by "Fundação La Caixa" and by "Fundação para a Ciência e a Tecnologia" under the scope of the project "Aquae Vitae - Água Termal como Fonte de Vida e Saúde" through the Promove Program ("Projetos I&D Mobilizadores") and also this work was supported by national funds through FCT/MCTES (PIDDAC): CeDRI, UIDB/05757/2020 (DOI: 10.54499/UIDB/05757/2020) and UIDP/05757/2020 (DOI: 10.54499/UIDP/05757/2020); SusTEC, LA/P/0007/2020 (DOI: 10.54499/LA/P/0007/2020).

Portions of the research in this paper use the FERET database of facial images collected under the FERET program, sponsored by the DOD Counterdrug Technology Development Program Office. [1], [2]

Contents

List of Figures

Acronyms

API Application Programming Interface. 46, 47, 60

CNN Convolution Neural Network. 2, 19, 50, 55, 59, 84, 91

DRF Django REST Framework. 60, 64, 66, 69

ERD Entity-Relationship Diagram. 35, 36

HLT Hessian Line Tracking. 12, 13

MTCNN Multi-Task Cascaded Convolutional Neural Network. 53, 85, 86

ORM Object Relational Mapper. 63

REST Representational State Transfer. 47, 60

SOAP Simple Object Access Protocol. 47

SPA Single Page Application. 46

UML Unified Modeling Language. 20, 24

UUID Universally Unique Identifier. 36

Chapter 1

Introduction

Skin health has become one of the vital parts of the population's day-to-day routine. Diseases like skin cancer account for a significant percentage of annual health concerns [3], and problems linked to health, self-esteem, confidence, and aesthetics have brought about an upsurge in the market of skincare products [4]. This increased amount of available products has made it difficult for consumers to cope with the emerging trends. With that, recommendation systems are showing an excellent approach to improve the user's experience while facing the vast amount of different available products, providing for each user only the products that best fit their profile and needs.

Besides, there is a trend toward more environmentally friendly and sustainable goods among consumers. Approximately 62.6% of consumers had shown a purchase intention for natural products [5], with this number being strongly influenced by the threat of climate change and environmental degradation. Such a trend has begun to affect the purchasing behavior of consumers, and they perceive it quite favorably. Even those who do not believe these goods have any impact are willing to pay a premium for them [6].

In this scope, the Aquaevitae [7] project was raised, which investigates thermal water's therapeutic properties and benefits and formulates functional cosmetic products that promote health and prevent diseases through thermal water-based initiatives. The current research proposes a novel neural network-based approach of natural skin care products recommendation. Its goal is to improve the user's experience while buying this product

and promote its use. The main objectives of the work can be summarized as follows:

- To develop an efficient Convolution Neural Network (CNN) model to identify wrinkles from user's facial images.
- To develop a web application capable of collecting crucial information from the user, integrating with the developed model, and generating a list of recommended thermal water-based skin care products that best fit the user's needs.
- To develop an administrator panel that allows the owner's team to manage the web application's information.

The document is divided into 5 (five) chapters. The current chapter provides an introduction and outlines the intended project's goals. The rest of the document is organized as follows:

- Chapter 2: State of the Art

  - Contains the review of the project's themes. First, a review of available cosmetic product recommendation systems will be presented, and their similarities and differences will be approached. Then, proceeding to a literature review of developed wrinkle detection algorithms;

- Chapter 3: Methodology and Background

  - Explains the approaches that will be followed in the rest of the work. It explains the web system business rules and requirements and the proposed architecture and methodology for the presented neural network model. In addition, it also presents all the concepts and background relevant to the present work;

- Chapter 4: Development and Implementation

  - Explains how the proposed web system was implemented, describing the tools and decisions made during the development process. Furthermore, contemplate the training process details and results of the proposed machine learning model;

- Chapter 5: Conclusion

  - It presents the conclusions derived from this work, highlighting areas that could be further explored in future research.

Chapter 2

State of the art

The proposed work presents a neural network-based approach to recommend thermal water skin care products. This approach is mainly composed of two branches, a web platform that will be made available to the day-to-day user allowing it to generate a personalized recommend list based on some previous information, and an algorithm able to identify wrinkles from a facial picture to improve the overall user experience. With that, a literature review was guided to collect and study the methodologies, concepts, approaches, and results that contemplate both mentioned scenarios in the current scientific landscape.

This chapter is divided into two sections that contemplate each specific part of this project. Initially, a review of the existent cosmetic product recommendation systems will be made, establishing the research goals and then presenting the collected data and conclusions. After that, the same is made to wrinkle recognition, contemplating different approaches from filters to machine learning models.

2.1 Cosmetic Products Recommendation System

Before starting the design of a new platform, is important to have knowledge of what is available nowadays, this can serve as a guide of what works and does not work, the expectations from this type of platform, and what fits and differs from each project. With

that in mind, a review of the existent cosmetic product recommendation platforms was faced, focusing on understanding the available features, methods, and results of each one, as well as trying to understand their integrations and how they work. Using keywords such as "Face Mapping", "Facial analysis" and "Cosmetic Products Recommendation" a set of relevant platforms was found. Their commercial purpose keeps these algorithm's modus operandi secret, making it difficult to understand how it was built. However, it is possible to identify that each one was trained with thousands of classified photographs, probably from the company's previous activities.

2.1.1 Dermalogica

The company Dermalogica [8], founded by a skin therapist called Jane Wurwand, has provided custom skin care solutions since 1986. Their website presents a lot about the company and the company's products, in addition to the possibility of making an analysis of your skin type and characteristics. Even if they recommend skin analysis with a specialist, preferably on-site, the platform provides an easy and free way to create a custom product list that best fits your skin.

This functionality uses a set of skin information collected from a form or from a facial picture analyzed by a machine learning approach. The skin form, first requests to the user select, from a set of 5 (five) skin types and a "Not Sure" option, all the ones that describe their skin (all the options can be selected at once, including the "Not Sure" option). After that, the user can select up to 3 (three) skin diseases from a set of 9 (nine) described diseases. In the next, it's possible to inform which kind of products are already being used by the user, from a set of 10 (ten) available product types, also being able to continue without informing any. To finish, it's necessary to choose an age group that better fits the user's condition.

While proceeding with the facial picture analysis, is first warned the user about the photo usage and asked for their permission, if not provided, the user is not able to proceed. After giving permission a screen with a small circle in the middle is shown, this is where

the user is supposed to fit their face, even being possible to submit the picture without it. After fitting it, a set of instructions is dynamically displayed guiding the user to the best photo, which is automatically captured when all the conditions have been met.

Both situations end in the products recommended list, showing 3 (three) Dermalogica products with their name, descriptions, price, picture, the product's purpose, and a link to add the product to the user's cart. The face analysis feature is integrated with the company's e-commerce, making it possible to buy the product directly.

Dermalogica's skin analysis makes use of an external face mapping tool called AI Skin Diagnosis from Revieve, founded in 2016 [9]. The tool is described as an automated computer-vision system, trained using an extensive data set of images, to uncover and measure more than 100 sub-metrics and around 20 main metrics such as redness, acne, texture, pore dilation, wrinkles, uneven skin tone, radiance, hyperpigmentation, and others. It also provides real-time feedback to the user about the quality considerations present in the photo as inadequate lighting conditions, the presence of eyeglasses, incorrect positioning of the face, and more. In addition, Revieve has a set of AI-based tools available on the Google Cloud platform, emphasizing the importance of such resources.

2.1.2 Garnier

Garnier is a company founded in 1904 that has been focused on hair and skin care [10]. The company was acquired by L'Oreal [11] in 1965 benefiting it from L'Oreal's extensive research and development resources. Their website presents a lot about the company, the products, the brands, and other information. It provides an AI-based tool called Skin Coach, which makes an in-depth evaluation of 6 (six) described signs of healthy skin from a user's facial picture and breaks down the results to him.

The tool focus appears to be the extraction of characteristics and so, do not offer any manual filling process to get a list of recommended products. The process starts with accessing the tools page and choosing to do an analysis. Then it's necessary to scan a QR Code and move forward with the smartphone. In the smartphone, the same page is shown

but instead of the QR Code, there is the option to do the analysis. The terms of use and privacy are displayed, only being possible to proceed when both were accepted, important to note that the tool is specified as not being used by people under 16 (sixteen).

After accepting the terms, some picture instructions are displayed, suggesting to the user remove glasses and makeup, put the hair behind the shoulders, keep a neutral expression, and find a bright environment with natural light. It's possible to submit an already-taken photo. Proceeding to capture an instant photo, the camera is loaded and some instructions are displayed, such as the light and face position guiding the user to the best photo, when the conditions are filled the photo is automatically taken giving the option to retake or proceed.

Once the photo has been submitted, the user is required to inform their age, then 4 (four) different skin types are displayed with a short description (normal skin, dry skin, oily skin, and combined skin), so the user's has to choose between one of them in order to proceed. To finish, the user informs a level of skin sensitivity between "None", "Low", "Medium" and "High". The results are then shown firstly giving a summary and 5 (five) analyzed signs, instead of the 6 (six) initially mentioned on the tool's page, to be chosen. When selecting a sign, the application shows a score from 0 to 10 for that specific characteristic, with some instructions, advice, and comments on the user's situation. Finally, the user can generate a skincare routine based on one of the signs that it wants to improve. The routine is divided between morning and night routine, each one contemplated with up to 5 (five) Garnier products.

The Garnier Skin Analysis feature makes use of a tool provided by L'Oreal called Skin Genius [11]. The company has been offering skin analyzer apps since 2019, using the background from years of dermatological research [12]. In total, 4,000 (four thousand) people were involved in the development of these apps in more than 50 (fifty) different disciplines. Resulting in more than 12 (twelve) developed algorithms and 4 (four) final skin analyzer apps, three of then still available for use. The researchers make use of 15,000 (fifteen thousand) skin images analyzed by dermatologists and 600,000 (six hundred thousand) photos analyzed by researchers. Initially, the algorithms were trained with

controlled images associated with a score, taken in the same light, at the same angle, and with the same background. However, after the acquisition of Modiface [13] into the L'Oreal Group, the algorithms were adapted to work with day-to-day facial pictures, being able to extract the information from photos in different environments and conditions. The company aims to keep improving the ability to predict the needs of the consumers.

2.1.3 Ignae

Ignae is a Portuguese company founded by Miguel Pombo in 2016 [14]. The company is located in the Azores and focuses on the development of cosmetic products based on the archipelago's resources. This brand offers premium anti-aging and regenerative skincare, combining natural ingredients from the region, with the latest cosmetic technology. The website displays some crucial information about the company, products, ingredients, and the Azores itself, as well as an AI-based skin analysis feature.

The tool offers to find the right products for the user based on an analysis made from a facial picture. To proceed, it's necessary to register on the platform, for that a set of information such as name, email, birth date, and phone number is requested. After that, the user has to fill in their skin profile in order to be able to accomplish the skin analysis. The skin profile consists of the user's birth date, gender, country, region, city, skin type (between neutral, oily, dry, combined, or sensitive skin), and a set of skin concerns up to 9 (nine) available. Then it's finally possible to proceed with the analysis, limited by 1 (one) free analysis per account. Which firstly shows some instructions about the photo, with an example picture, as well as some description about the tools and what it does. While proceeding to take the picture, the user can submit an already taken picture or scan a QR code to download the Ignae app in order to continue the process. As this work focuses on the web platform, the mobile application will not be evaluated.

After submitting the picture, a set of information extracted is displayed. The 9 (nine) analyzed characteristics are shown in a radar graph each one containing a score from 0 (zero) to 100 (one hundred). Besides it, the submitted picture is shown, together with

the submission date, an image quality rate, a perceived age, and an eye age. The page then highlights the 3 (three) most relevant areas to be focused by the user, which means the 3 (three) lowest scores from the 9 (nine) extracted skin information. Above that there are up to 3 (three) recommended products, that you can access on the buy page or add directly to your cart. At the end of the page, all the analyzed features and their score are visible, enabling to the user see a detailed description of the characteristics and the result. It was not possible to identify the provider used by Ignae to analyze the facial images. As it's necessary to register in the platform in order to do the facial analysis, the results are stored in the user's account and can be accessed at any time.

2.1.4 SkinQ

SkinQ is an Indian cosmetic Brand described as India's first Made Safe certified, dermatologist-formulated, multi-active skincare solutions brand for Indian Skin [15]. The brand offers a wide range of skincare solutions that address concerns such as pigmentation, acne, uneven skin tone, and aging focusing on the unique needs and challenges faced by melanin-rich Indian Skin and Skin Of Colour. The website initially presents some information about the company, products, and the people behind it, as well, as an AI-based tool called SkinQ AI Mirror, providing for free a facial analysis and a set of recommended products.

After accessing the tool's page, it's possible to see a description of it and the benefits of using a personalized skin care plan. Together with the option to proceed with the analysis or scan a QR Code to proceed with the picture submission in the smartphone. It's necessary to accept the use terms before moving forward. The user's camera is then shown with instructions being dynamically displayed on the screen to guide the user, a small button to submit an already taken photo is available at the bottom of the screen. After all the needed conditions are accomplished, the photo is automatically taken, so the user can submit it or take it again. Then it's necessary to provide a phone number, name, and email address in order to proceed, if informing a phone number that was already in use, the other two information will be automatically filled in.

The submitted photo is then displayed with the extracted information. The data is divided into two sections, the first one is "Life-Style", containing the gender, age, location, and skin type recognized in the photo (the skin type attribute is not clear on this page but looking in the provider's website, it's possible to understand that represents the identified skin tone in a number from one to six, being lightest the lowest). The second one is "Skin Care", which contains 12 (twelve) skin characteristics, with a level from 0 (zero) to 5 (five), and 3 (three) additional data, e.g., facial landmarks. A button is displayed to access the recommended products list, containing the product's image, price, description, ingredients and use instructions, with the possibility to directly add the product to the user's cart and check the purchase.

The SkinQ AI Mirror makes use of an external tool called DeepTag [16], which was created by BrighTex Bio-Photonics, an American startup founded in 2005, specializing in machine vision applications for beauty, personal care, and security surveillance. The tool trained with more than 10,000 (ten thousand) facial pictures, can extract a complete set of information from the user's photo, such as their skin tone, age, hair type, and much more.

2.1.5 Lululab

Lululab is a company founded in 2017, born from the Lumini project of Samsung Electronics C-Lab started in 2016 [17]. It's described as a skin data-based beauty and healthcare platform, promoting healthier daily life. Lumini was initially a complete product, with hardware and software to capture and analyze skin photos with quality.

With a touch screen, the gear was responsible to provide the ideal environment and take the user's best picture. That photo was then sent to a cloud to be processed by an algorithm able to extract the skin information, such as pores, wrinkles, and redness, and displayed to the user together with a set of recommended products. Nowadays it's possible to make use of the Lumini process without the hardware, being able to get similar results from the smartphone, for example.

2.2 Wrinkles Detection

The advancement of computer vision and deep learning technologies has revolutionized the field of image analysis, leading to significant progress in the automatic recognition and assessment of facial wrinkles. This section presents a comprehensive review of the current state of the art in wrinkle recognition in images. In order to understand the approaches and methods commonly used in this type of task, research was faced searching for materials for wrinkle detection and recognition techniques. Relevant articles will be contemplated in this work, approaching the objectives, methods, and results of each one, as well as the research environments.

2.2.1 Hessian Line Tracking

Members of the Manchester Metropolitan University and Loughborough University aims to address the challenges of manual wrinkle annotation, which is subjective and time-consuming, by proposing an automatic wrinkle detection method [18]. The methodology involves the use of Hessian Line Tracking (HLT) to identify potential wrinkle pixels in the ridge area of the Hessian matrix. The study utilized a dataset of 100 cropped forehead images from the Bosphorus dataset [19], annotated by three coders, including a beauty therapist, to assess the accuracy and reliability of the proposed method against benchmark algorithms.

In terms of methodology, the article highlights the importance of accurate wrinkle detection for age estimation and skin aging studies. The proposed HLT method outperformed existing methods such as Cula Method, Frangi Filter, and Hybrid Hessian Filter [20], achieving an accuracy of 84%. The study also evaluated reliability of manual annotation, demonstrating accuracies of 94% and above. The results showed that the HLT method significantly improved wrinkle localization accuracy compared to other methods.

The results and discussion section of the article emphasized the importance of accurately locating wrinkles relative to the ground truth, showcasing the effectiveness of the proposed HLT method in increasing connectivity of wrinkle detection, including its ability

to explore wrinkle connectivity, improve on the ridge and valley pattern. The article concludes by highlighting the potential of the HLT method in automating wrinkle detection from 2D images, providing a valuable tool for age estimation and skin aging research. Further validation and detailed quantification of wrinkle detection are suggested for future research to enhance the method's accuracy and applicability, including investigating wrinkle depth, length, width, and exploring curve fitting algorithms for age estimation and micro movement detection.

2.2.2 Semi Automatic Labeling Technique

A semi-automatic labeling methodology was developed by the Lululab Inc. in [21], aiming to improve facial wrinkle detection through this novel strategy and a deep learning model. The primary goal is to address the limitations of traditional image processing methods by utilizing deep learning techniques that require accurately labeled datasets, which are challenging to generate due to the variability in wrinkle characteristics. This study proposes a method that combines texture maps and roughly labeled wrinkle masks to generate ground truth data, which is then used to train a deep learning model for enhanced wrinkle detection.

The methodology involves several key steps: first, a binary mask is created to roughly indicate wrinkle areas. Then, a texture map is extracted from the original image, and non-wrinkle textures are removed by multiplying the map with the binary mask. This processed map is further refined through adaptive thresholding to produce the final ground truth for training the model. A machine learning model for semantic segmentation of photographs, based on the UNet [22] network, is then trained from them. The model uses an image of the face cropped from the forehead to the bottom of the eyes as the input, concatenated with the previously generated texture map as the segmentation mask. In total 300 (three hundred) images were used, 250 for training and 50 for validation, taken from the Lululab's Lumini KIOSK [17].

The results of this approach are promising. The proposed method significantly outperforms traditional Hessian and Gabor filter-based techniques [18], [20], [23], as evidenced by superior performance metrics such as the Jaccard Similarity Index (JSI), reaching a maximum value of 0.47 for the forehead and 0.44 for the eye area. The study shows that the deep learning model trained with the semi-automatically labeled data detects more wrinkles and generates fewer false positives. However, the authors acknowledge some limitations, including challenges in detecting fine wrinkles and instances where hair or fluff is mistakenly identified as wrinkles. They suggest future work should focus on improving fine wrinkle detection and reducing false positives.

In conclusion, this study presents a compelling strategy for enhancing facial wrinkle detection using deep learning. By addressing the difficulties in generating accurate ground truth data, the proposed method improves upon traditional image processing techniques and demonstrates the potential for more reliable and comprehensive wrinkle detection. Future research will aim to refine these methods further, enhancing their applicability and effectiveness in real-world scenarios

2.2.3 Novel wrinkle detection evaluation metrics

The work proposed in [24], developed by members of the Shanghai Jiao Tong University and Chinese Academy of Sciences, presents a new approach for recognizing wrinkles and evaluating wrinkle detection algorithms. The primary goal is to enhance the accuracy and efficiency of wrinkle detection by proposing a method that considers the ridge-valley-ridge pattern of wrinkles in images. By collecting high-resolution face images from diverse sources and annotating entire face regions, the study seeks to provide a comprehensive dataset for training and evaluating the wrinkle detection algorithm. The methodology involves inviting dermatologists to grade wrinkles, followed by independent annotation of skeleton lines by three coders based on predefined criteria, ensuring a meticulous and standardized approach to data annotation.

The article introduces an encoder-decoder network structure for wrinkle detection,

trained with 1024 (one thousand twenty four) full-face photographs of volunteers around China of varied quality and environments, and proposes a distance-based loss function centered on skeleton lines to improve the detection of significant wrinkles while minimizing the detection of insignificant ones. The evaluation metrics include curve similarity, location similarity, and gradient similarity, providing a comprehensive assessment of the proposed wrinkle detection method's performance. The study compares the proposed approach with existing methods, showcasing its ability to accurately detect prominent wrinkles, such as those on the forehead, while maintaining precision in identifying full-face wrinkles. The methodology's effectiveness is further demonstrated through the evaluation of various performance metrics, highlighting the method's superiority in wrinkle detection.

The results of the study indicate that the proposed wrinkle detection method based on skeleton lines and distance-based loss function significantly enhances the accuracy and reliability of wrinkle detection in facial images. By focusing on the ridge-valley-ridge pattern of wrinkles and introducing a novel assessment metric, the method proves to be more effective in identifying relevant wrinkles and minimizing false detection. The comparison with other methods illustrates the superiority of the proposed approach in detecting prominent facial wrinkles, showcasing its potential for practical applications in dermatology and aesthetic treatments. Overall, the study's results validate the efficacy of the proposed methodology in improving wrinkle detection accuracy and provide a promising avenue for further advancements in this field.

2.2.4 Convolutional Neural Network approach

The article [25], the researcher from Dublin City University and Princess Nourah bint Abdulrahman University, presents a comprehensive study aimed at revolutionizing wrinkle detection through advanced computer science applications. The primary goal of the research is to introduce an innovative methodological framework that leverages cutting-edge technology to accurately identify and classify wrinkles on the facial skin, particularly

focusing on the forehead region. By proposing a multi-faceted approach that integrates image processing techniques and sophisticated classification algorithms, the researchers strive to enhance the precision of wrinkle detection and provide valuable insights for cosmetic healthcare practitioners.

Methodologically, the study used the FERET database [1], [2] to train and validate the algorithm with a InceptionV3 model [26].Only photographs with a clear view of the volunteer's forehead were chosen. A "Multi-task Cascaded Convolutional Networks" (MTCNN) model [27] was used to crop only the forehead of each image, reducing disturbances in the data. From this, the KMeans clustering algorithm was used, which related the photos into groups, assisting in the classification process, resulting in a dataset of 618 images (309 with expression lines and 309 without).

The results of the study demonstrate a significant achievement, with the computer-based approach yielding an accuracy rate of 85.3% in wrinkle detection. Through meticulous wrinkle detection and classification using current models and algorithms, the researchers showcase the potential of technology in enhancing cosmetic procedures and healthcare practices. The study not only underscores the effectiveness of the proposed methodology but also emphasizes the importance of further research to address limitations and explore broader applications in the realm of facial cosmetic healthcare.

In addition, the article provides a compelling insight into the intersection of computer science and medical aesthetics, highlighting the transformative potential of advanced technologies in improving wrinkle detection and recommending facial fillers. The meticulous methodology and promising results underscore the significance of leveraging computational tools to enhance cosmetic treatments and advance healthcare practices. The study sets a solid foundation for future research endeavors in the domain of computer-based approaches to skincare and cosmetic enhancements, paving the way for innovative advancements in the field.

2.3 Conclusion

In this chapter, it has been possible to get an idea of the state of the art in cosmetic product recommendation systems and wrinkle detection techniques. It became clear that some features are mandatory in this type of project, such as the user's permission to use photos, and that each approach has its own particularities but ends in a similar way. Most of the platforms don't offer an easy way to retrieve the list of products generated and only one offers the possibility of manually filling in information about the skin. The tools have mainly been used to promote the sale of their own products and make use of an external provider, not using their own algorithm.

In addition, older approaches for wrinkle recognition is mainly based in mathematical filter while the most recent one makes use of neural network and deep learning approaches. The second case presents better results but needs a large set of available and classified data to be possible, which was pointed as a difficult in all works. The quality and quantity of the data was mostly related with the final result, emphasizing that established companies and products have a greater chance of developing their own successful algorithms.

Chapter 3

Methodology and Background

In this chapter will be developed the requirements and methods needed for the development of the Recommendation System and the neural network responsible for the wrinkles detection. It will be elaborate the functional and theoretical requirements in each context, use cases and the expected behaviour in the web system. As well the concepts that encompasses the development of a CNN.

3.1 Proposed solution

This works try to improve the experience of the final user while buying skin care thermal water based products. The way which it does it is providing a list of recommended products given a set of user's information. More specifically, the user fills a form with their information and send it to a server, that process it an elaborate a grade of importance for each product. This grade is later used to generate the user's recommended list. Also, the system gives the user an option of prefill the form from a facial image. That image, will be send it to the server before the form data and will be processed by machine learning models to collect additional data, that prefill part of the form's information and improve the products list generation. In addition, this works gives an administrator panel that make able for the system's responsible keeps improving the furnished set of products, and evaluate partnerships proposal from another thermal water based companies that wants

to distribute theirs products in the platform. In this way, keeping the system life cycle longer.

3.2 Use cases

Defining the system use cases is an important step in software engineering. By definition, "A use case is a description of the possible sequences of interactions between the system under discussion and its external actors, related to a particular goal." [28]. In this way, is a good approach to clarify the system needs being easily understandable by all stakeholders, improving the decision make process all along the system life cycle.

The use cases can be represented in many ways as flow charts, sequence charts, pseudo-code, or simple text. In this work, the use cases is being represented as an Use Case Unified Modeling Language (UML) [29] diagram, to make it more visual. This type of diagram makes use of the UML pattern together with the Use Cases main definition as explained bellow.

- **Actors** represents the final users. It can be any source that interacts with the system. In the diagram, is represented by the stick figure outside the box;
- **Scenarios** represents a possible sequence of interactions. In the diagram, can be visualized as a chained goal group;
- **Goals** represents a goal that wants to be achieve by an actor, also knows as the use cases. In the diagram, is represented by the flat circle;
- **Relationships** is the relations between a goal and an actor or between goals. It can be in 4 (four) different types: i) An Association, which represents a communication or interaction between an actor and a goal and it is depicted by a simple line between an actor and a use case. ii) An Include Relationship which indicates that a use case includes the functionality of another use case and it is denoted by a dashed arrow pointing from the including use case to the included use case with the keyword

"include"; iii) An Extend Relationship which represents a use case that can be extended by another use case under specific conditions and it is represented by a dashed arrow with the keyword "extend"; iv) A Generalization Relationship which indicates that one use case is a specialized version of another and it is represented by an blank arrow pointing from the specialized use case to the general use case.

3.2.1 Recommendation System

For this work an Use Case Diagram was created to represent all the high level functionalities of the system in order to improve the development process and align the expectations with the Aquaevitae team. The full diagram can be visualized in the figure 3.1. Each component and characteristics of the diagram will be elaborated in sequence.

Actors

The system has two actors, the "Final User" and the "Administrator". The Final User Actor represents the system's persona and it's the target of the main features, as generating the recommended products list. The other one is the Administrator Actor, which represents the a member of the Aquaevitae team, that will be responsible for the management of the platform and validation of the partnerships requests.

Goals and Relationships

Each represented goal and relationship will be explained in a list separately for each actor. The Final User Actor features the follows use cases:

- **See list of available products:** the user can see all the available products in the web platform;
- **See products details:** while seeing a list of products, the user can see the respective details of each product, as its name, size, type, category, a short description and a where-to-buy URL;

- **Access product's shop url:** as an extension of "See products details", the user can access the product URL showed in the details to be redirected to the informed sale's website;

- **Fill facial skin characteristics form:** filling the facial skin form is the first step in the scenario of receiving the recommended product's list. The user can inform their skin type and a set of skin diseases that will be used to generate the list;

- **Submit facial picture to prefill form information:** as an extension of "Fill facial skin characteristics form", it represents the possibility of the user, if they want to, to prefill the form information with the machine learning algorithm. An user's facial picture is necessary to perform this goal;

- **Visualize recommended products list order by importance:** after informing their facial characteristics, the user can see the list of recommended products. This goal is an generalization of "See list of available products", which means, it has the same characteristics as it but just shows the recommended products with the addition of each importance grade;

- **Receive recommended products list on email:** as an extension of "Visualize recommended products list order by importance", the user has the option of receive the recommended list by email;

- **Request Partnership:** the user can send a partnership request filling a form with their information and their company information, as well a short cover letter.

The Administrator use cases are being depict as bellow:

- **Login:** different from the Final User, the Administrator can and have to be logged in the application to perform any other action;

- **Register/Delete users:** as an Administrator, its responsible for manage the administrator users, as removing old ones and adding new ones;

- **List partnerships requests:** as one of the main activities of the Administrator, it can see all the sent partnerships requests, as well their status;
- **Approve/Deny requests:** each request can be evaluated by an Administrator, being able to approve or deny each one. Each approved requests results in the addition of the new company/products;
- **Manage Companies:** the Administrator is also responsible to manage the companies registered in the system, as list, add, remove and edit companies information. Each company has their own set of products;
- **Manage products:** as an extension of "Manage Companies" but also an independent action, it's possible to the Administrator manage all system's products and Companies products. Including list, add, remove and edit the products.

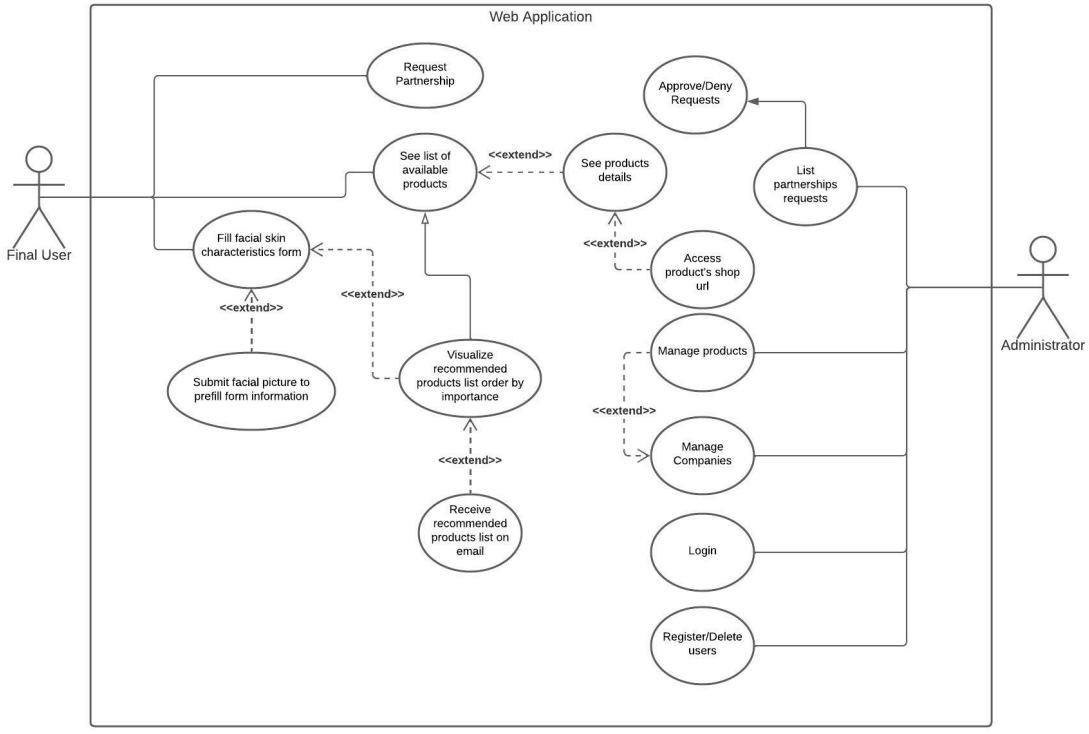


Figure 3.1: Web System Use Cases Diagram.

3.3 Activity Diagram

An Activity Diagram is one of the diagrams contemplated by the UML. It describes the expected behaviour and activities of a system's flow, facilitating the communication between all the stakeholders and making decisions about architecture, methods and functionalities. With this type of diagram is possible to have a clearer vision of the logic, decisions and operations inside an specific use case.

This type of diagram allows to visualize all process steps. The main components are described as follows:

- **Actions:** it's a task executed by an operator, which can be both an user or a system. They are represent in the diagram as a rectangle with rounded edges;
- **Decision Node:** it's a conditional branch in the flow represented by a diamond shape. Each condition have at least two possible output paths, each path have their own condition described right above it;
- **Control Flow:** it's represented by the connection arrow between components;
- **Initial Node:** symbolizes the flow's begin. It is represented by a black circle;
- **End Node:** represents the activity's end and is symbolized by an outlined black circle.

Moreover, some additional components are available to describe more complex steps in the activity. In total there is more then 10 (ten) additional components that can embody diversified actions, e.g., parallel steps, conditional loops and activity states. For this work, only the used ones will be explained and are described as next:

- **Pool:** a pool is the representation of the activity. It's symbolized by a wide rectangle around the other components and it has the activity's title on top;
- **Swim-lanes:** it's symbolized by smaller rectangles inside the Pool Rectangle. Each Swim-lanes represent an actor in the activity, e.g., the user, a web system, an external API, etc;

- **Note:** represented by a square with a folded edge, It allows the creators or collaborators of the diagram to communicate additional messages that do not fit within the diagram itself;
- **Conditional loop:** allows to model a repetitive sequence of instructions. It is represented by an rectangle with an instructions flow inside of it and the loop's condition described in the top left;
- **Fork:** It divides a single activity flow into simultaneous activities. It is symbolized with multiple lines with arrows from a horizontal black bar;
- **Synchronization bar:** combines simultaneous activities and reintroduces them into a flow where only one activity takes place at a time. It is symbolized with multiple lines with arrows to a horizontal black bar;
- **Send Signal:** indicates that a signal is being sent to a receiving operation, as an API call. It's symbolized by a rectangle with a pointed side;
- **Receive Signal:** Demonstrates the acceptance of a signal. It's the other end of "Send Signal" and is represented by a rectangle with inverse arrow shaped side;
- **End Flow Node:** represents the end of a specific process flow and should not represent the end of all flows in an activity. It's represented by a circle with a inner cross.

3.3.1 Proposed Web System Activities

For this work, a set of activity diagrams was built to represent the main flows of the system, symbolizing multiple scenarios that are important for understanding each stage involved in drawing up the list of recommended products and extract the data from the facial picture. Along with a diagram to represent the partnership request and approval flux. With these diagrams, is possible to have a clearer vision of the business decisions, architecture, communication and operations that contemplates each scenario.

Buy a recommended product

The main objective of the web system developed in this project is the promotion of cosmetic products based on thermal water, in this way an activity diagram was built to represent the scenario of a user buying a recommended product through the platform. This scenario begins with filling in the skin characteristics form and goes through submitting a facial photograph, receiving a list of recommended products and accessing the product sales platform. The full diagram can be seen in the figure 3.2

To start, the diagram contains a Pool to represent the scenario of "Buy a recommended product", this Pool has 3 (three) different Swim-lanes. The first one is the "User" that represents the final user and who is actively operating with the web system in this scenario. The second one, is the "Web Client", which is the web application's client running in the user's browser. The last one represents the application's back-end and the "API", responsible to operate in the server side. This distinction is valuable to understand the different responsibilities from each actor and how they interact, as their logic and expected behaviours.

The scenario starts with the user accessing the web application's home page and choose to fill in the form to generate the list of products. The client then gives the possibility of submit a facial image to pre-fill in the form. If the user proceed to the picture submission, a set of instructions will be shown (e.g., remove glasses and hats, no makeups and take the picture facing the light). Along with this, the user can allow or not allow the sent image be stored in the project server. The image storage, with the user's authorization, provides the application the capability of collect data to retrain or develop new neural networks in the future.

The image is then captured and sent to a server, while, in parallel, the user proceeds to enter their skin type, as well as whether they have chosen to fill in the form manually. The server, store the image and trigger a task to a queue that will process it in parallel (the image processing flow can be better understand in section 3.3.1), and return an id to the client that refers to the correspond image analysis. When the user reaches the stage

of inform their skin diseases, the client will check if the user had submitted an image, if so, it will ask the results to the server with the provided image analysis id. If the server has no results yet, it will return an empty list. The client will keep asking until get a valid result list or reach 40 (forty) tries. The result is a list of identified skin diseases from the submitted picture with their respective levels.

With the analysis results in hand, the flow proceed to the skin diseases step, where the user can manually fill all their facial skin diseases, or edit any pre-filled disease and their levels. In this moment of the project, the only recognizable disease is the wrinkle level, provided from the integration with the developed neural network model.

When satisfied with the reported diseases, the user then enters some personal information, as their name, age and, if they want to receive the recommended products list by email, their email. Finally, the form data is sent to the server and the client redirect the user to their recommended products list page. While the server, calculate the importance grade of each product (the importance grade calculation process can be seen in section 3.3.1), builds the recommended list and sends it back to the client.

With the listed products, the user can check any product's details, to see their specific categories, ingredients, characteristics, recommended use and some other information as well a where-to-buy button. That will redirect the user directly to the product's owner website so it can be bought.

This flow was architect aiming the functionality and usability of the system, to improve the user's experience while filling the form and purchasing the products. The parallel processing steps provide speed in the scenario while the multiples validations adapt the experience for each individual user.

Process facial image

An important step in creating the list of recommended products is the automatic extraction of features from the facial image. The diagram designed in this part is a piece of the scenario described in section 3.3.1 and includes the actions required in the facial analysis process to gives a clearer vision of the decision and interactions to integrate the

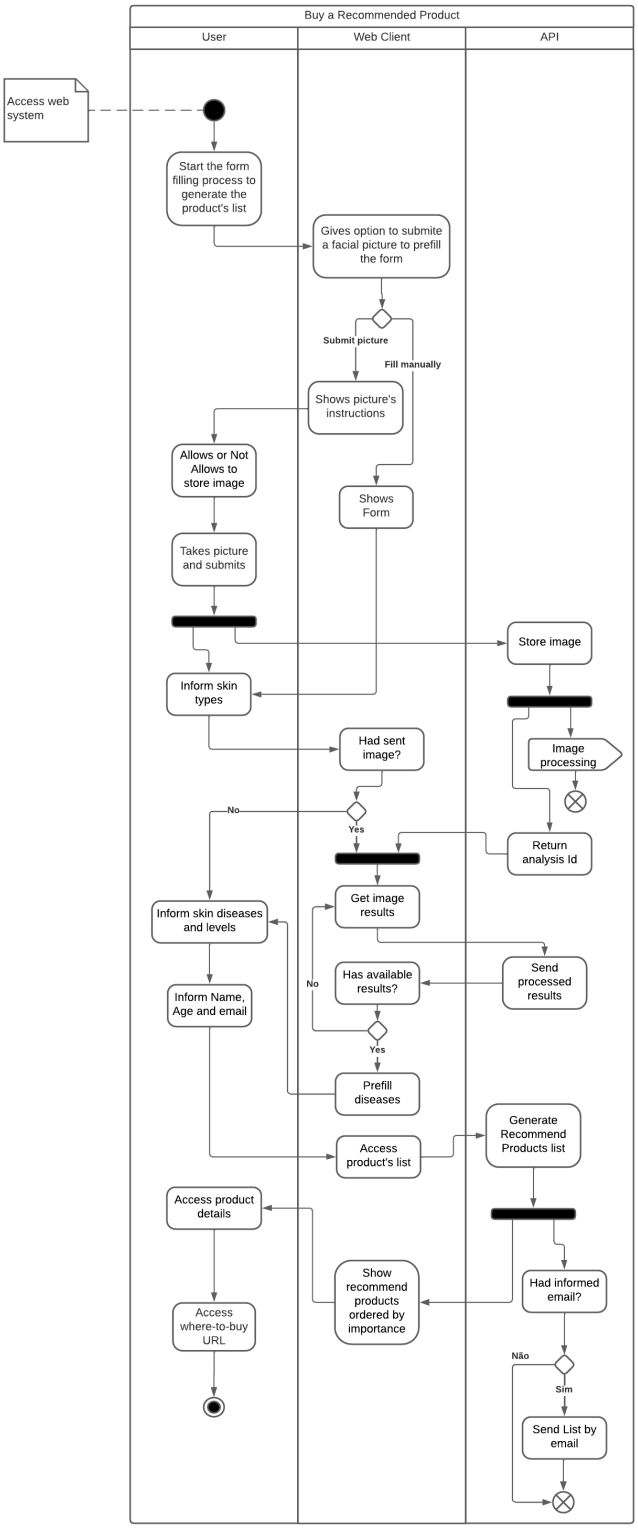


Figure 3.2: Activity Diagram of Buy a recommended product scenario.

web system with the machine learning models.

This diagram has a Pool to represent the "Process Image" scenario. This pool has 3 (three) swim-lines, "Web Client" and "API", the same as in the "Buy a recommended product" diagram, in addition with the "Celery Server", that represents the task queue worker used in this work (a better definition of the task queue can be found in section 3.5.4). A full vision of the diagram can be seen in the figure 3.3.

The scenario begins with the received request from the client with the user's facial picture and the user's authorization to store the image. The image is then stored, so it can be used by the Celery Server later, and a signal to the task queue is triggered in parallel with returning the identification of the analysis process to the client.

The Celery Server receives the task from the queue and starts the characteristics extraction process. In this work, two machine learning models is being used with the user's image, the first one is responsible to estimate the user's age and the second one to identify the forehead user's wrinkles. Both models are applied in parallel to the image and at the end, the application checks for errors and the analysis register stores the information. Then, it verifies the results and does the same, finishing by setting the analysis process as done. It is important to mention that only the wrinkle detection model generate a returnable result, the estimated age is not visible to the user and will only be used later to improve the recommended products list generation, as shown in section 3.3.1. Lastly, the Celery Server checks whether the user has allowed the image to be stored, if not, it deletes the image from the server and then finishes its work.

The flow proceeds when the client asks for the analysis results to the API, which evaluates if the process is already done, if it's not, it returns an empty result list. If it's done, it follows to verify if there is any result and if it is return the result list, if it's not, return an error message. This is because, if the analysis process is done but has no result, means that something went wrong. The specific error is not returned in the message because it is not relevant to the end user, but it is still stored in the analysis register. If the clients receive a not valid result list, it will end the flow if it receives an error message. If not, it will try again until receives a valid list or reach 40 (forty) tries. When a valid

list is received, it will pre-fill the diseases in the form with the result data.

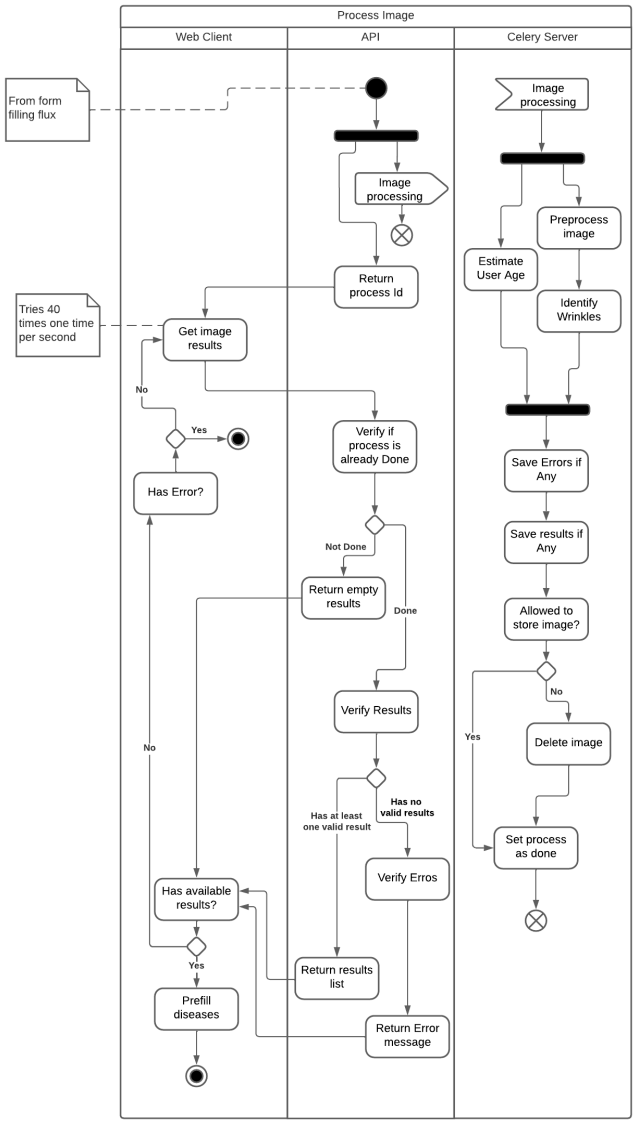


Figure 3.3: Activity Diagram of Process User's Facial Image scenario.

Calculate products importance grade

In order to elaborate a recommended products list, the application establish a importance grade for each product, based on the information provided by the user in the form. The diagram in the figure 3.4, represents in a visual way the rules behind the grade

composition. The grade cross both the user's and product's information to consider its compatibility with the user's needs. It also contemplates how the user's submitted picture improve the quality of the recommended list.

First, is important to cite that each product has a set of treatment provided and a set of skin types that it is suitable for, as defined in section 3.4. Also, each disease informed by the user in the form have a set of correspondent treatments. So, the grade is comparing the treatments needed by the informed diseases with the provided treatment by the product, as well the skin types compatibility.

The designed diagram has a Pool that represents the "Calculate product importance grade" scenario, this pool has only one swim-lane symbolizing the API, that is the only responsible to calculate the products grades. The process starts getting the product's and the user form 's info. Then, for each disease listed by the user, it gets it's correspondent set of treatments, that is intersect with the set of product's provided treatments. The length of the intersection is multiplied by the disease level and after this value is sum to the current product score.

After computed all the informed diseases, the system proceed to calculate the user's aging level, for that it first verifies if the product has anti-aging properties, if it does, it tries to get the estimated age from the user's facial picture. If there is no submitted image or the product has no anti-aging properties, the importance grade remains the same. Otherwise, it will subtract the user's informed age with the estimated age and increase the importance grade based on that. In this way, as higher the age difference, the higher importance of products with anti-aging properties.

Finally, the product's suitable skin types is intersect with the set of user's informed skin types. From that, the intersection length is elevated by the 2nd (second) power and then sum with the product's importance grade. In this way, a product that provides a good number of the treatments needed by the user gets a high score, as does a product that provides treatments for a high-level informed disease. Also, the product is valued for its compatibility with user's skin types and their anti aging properties depending on the user's aging level detected from the submitted facial picture.

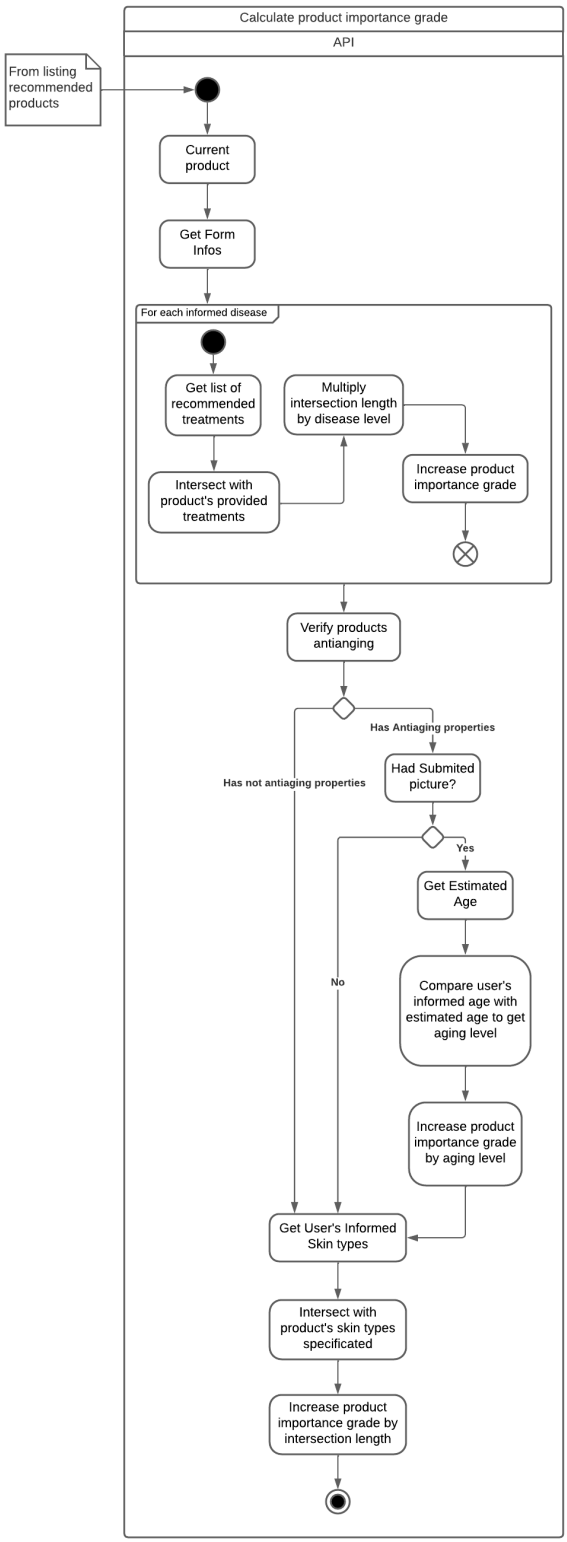


Figure 3.4: Activity Diagram of Calculate Product's Importance Grade scenario.

Partnership Request and Validation

In order to improve the life cycle of the platform, this project provides an alternative so outside companies, with natural thermal water based cosmetic products, can submit a partnership request. In this way, requesting that their products become part of the system's portfolio and can be recommended to the users. After receiving a request an administrator will be responsible to get in touch with the requesting company and evaluate if it is or it is not able to integrate the project, after that the admin user will be also responsible to add and manage the company's products inside the system. This approach goals to keep the application updated and with the most recent products available in the market.

This diagram represents the whole partnership flow from requesting the partnership by the user to the request's evaluation by the administrator. It contains a Pool representing the "Request/Approve Partnership" scenario, with 4 (four) swim-pools in total, the "User" and the "API", as already present in other scenarios, in addition with the "Administrator", representing the system's administrator responsible by the partnership evaluations and system's management, and the "Administrator Panel", being a different web platform accessible only by the administrators. A view of the full diagram can be seen in figure 3.5.

After accessing the partnership's form, the user can fill all their company's information as well as some own information and an cover letter to introduce the request to the administrator. With that, the API will check by the presence of open requests for that company, if there is any, it will automatically deny the request and notify the user of it. If not, it will search any company's request in a 3 (three) months range and will also automatically deny the request and notify, if founded. In this way, avoiding request spams and repeated request.

After created, the administrator will be notified and the request will be visible in the administrator panel. The administrator can then see the requests data, being able to contact the request's owner in order to get more information to help in the evaluation,

as well new company's data and the products information to be latter added. When satisfied, the administrator now has to add a comment justifying their judgment, e.g., providing the specific denial or approval reason, and then change the request status. In this way, will be easy to understand the evaluation if necessary in the future.

If approved, the system will notify the request's owner and create the correspond company's register. The administrator then is responsible to get the products information and add it to the system. In this way, being able to evaluate everything one more time and to insert the products with the correct data. In addition, if it is denied, the owner will be notified, and it is important to note that the administrator's comment is not visible to the user and is only used for internal purposes.

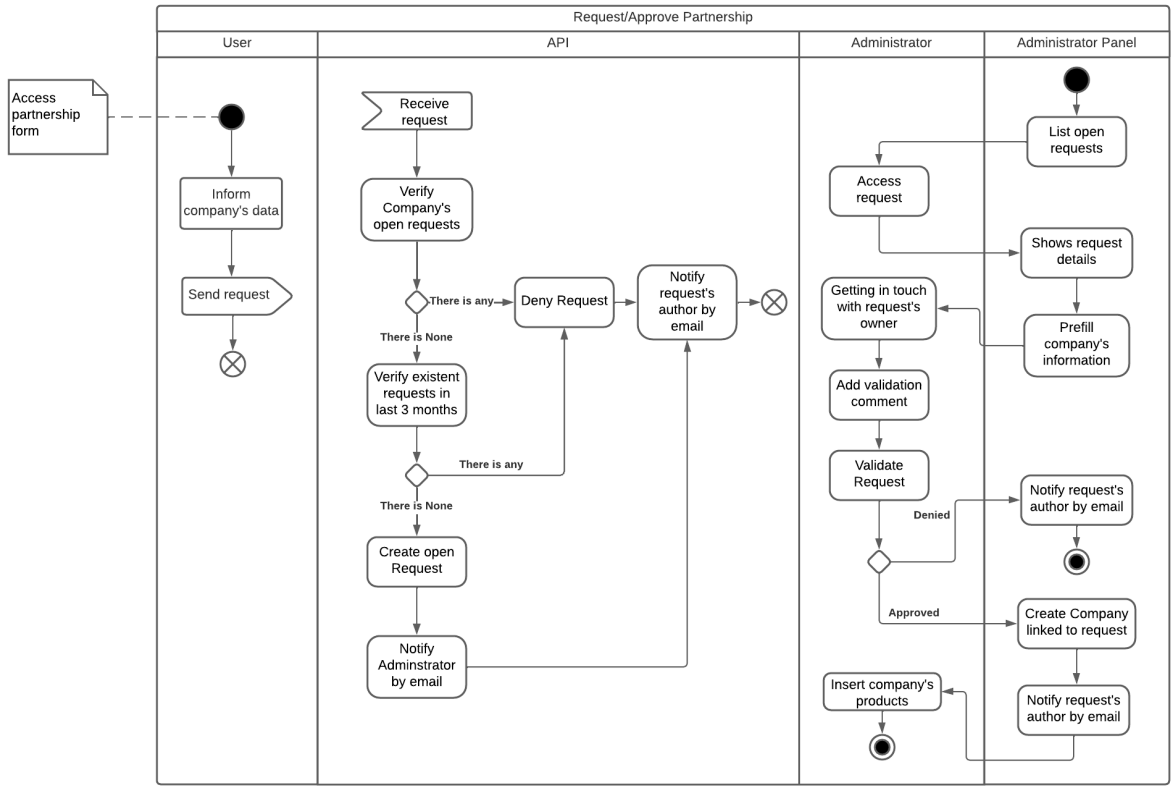


Figure 3.5: Activity Diagram of Request/Approve Partnership scenario.

3.4 Entity-Relationship Diagram

An Entity-Relationship Diagram (ERD) [30] is a graphical representation of the relationships and data structures among databases. It comes in handy during database design to map the many entities, objects or concepts that are to be stored, and the inter-relationships of those entities with one another. A rectangle will usually represent an entity, and the name of the entity will be inside it. The relationships among different entities are represented by connecting lines. Other characteristics of entities, which further describe the entity attributes are represented by lines inside the entity rectangle. ERD make the systematic sorting of data possible, through which the design of the database can be thoroughly thought out and efficient well before any actual data storage process is initiated.

There are so many benefits that come with the use of ERD. For example, they allow for the structure of the database to be presented clearly and concisely, so that designers and other stakeholders can understand the components of the database and their interrelation. This visualization can help to identify probable issues or redundancy possibilities in the design stage, thus saving time and resources. Besides, ERD allow team members, including developers, database administrators, and business analysts, to communicate with each other, offering a common, unified model for specifying the structure of the database. During the implementation phase, ERD can also be considered as the blueprint for the actual schema design; in this stage, consistency and accuracy are maintained. An ER diagram consists mainly of three important components: entities, relationships, and attributes. All are absolutely necessary for the definition of the database structure and to ensure data integrity.

3.4.1 Proposed System ERD

In this chapter, it will be present the ERD developed for the proposed work. The ERD provides a comprehensive visual representation of the database structure underpinning the cosmetic recommendation system. It elaborates how entities as users, products, form

data and partnerships are displayed and related to each other. Ensuring that all the data relevant to it is properly captured in an accurate manner and is thus properly arranged for efficient use in the web application. Each entity, as well it's relationships and properties, will be explained individually as next. The full ERD can be seem in figure 3.6.

Commons properties

For architectural purposes, most tables share some common properties that focus on the table's registers audition. These properties are:

- **id:** the unique identifier of each register, also known as "Primary Key". It's an Universally Unique Identifier (UUID) [31] generate automatically when the register is created;
- **create_at:** a date field to register the creation date of the register;
- **updated_at:** a date field updated every time some change in the register is computed, it keeps the information of the last change's date;
- **is_deleted:** the system will contemplate a soft delete functionality, which means the data will remain in the database but be flagged as inactive, this property represents the inactive flag of the row. This method permits recovery of data if wrongly deleted and enables the maintenance of a historical log of operations.

Administrator User

A table named "*auth_user*" represents the administrator user's entity inside the system. This specific table is automatically generate by the web framework used in the development process (explained in section 4.1.1), so only the relevant properties will be explained. This table, is mainly used to registration and authentication of the administrator users, and it's properties fulfill this exact purposes.

- **username:** the administrator unique username;

- **password:** the administrator password, it's stored as an encrypted value;
- **is_superuser:** a flag that determines the level of authority of the administrator, when active, the user is allowed to manage the system's users and delete softly deleted registers;
- **email:** the administrator's email, where it will receive the partnership request's notifications.

Products

The products in the system are stored in the "*product*" table. It contains most of the product's information that is used to generate the product score and the product's specifications. This table and it's properties, is a representation of the product's sheet provided by the Aquaevitae's team, that can be seen in the Appendix H.

- **name:** the product name;
- **category:** the product category, it's a single-length char value interpretable by the system. It can be one of the 8 (eight) possibilities described in the product's sheet. The relation between it's value and the category is visible in Appendix A;
- **size:** it's the numerical size of the product;
- **size_type:** it's the type of the size, which can be in grams or milliliters;
- **characteristics:** a text explaining some product details;
- **recommended_use:** a short text with some instructions of how use the product;
- **contraindications:** a short text with some contraindications of the product;
- **price:** the product's price, it can be null to avoid need of manually update the product's price in case of owner changes;
- **url:** the buy URL of the product, it redirects to the product's selling page;

- **type:** it's a small integer value interpretable by the system to represent the product type. It can be any integer in the range of 0 (zero) to 15 (fifteen). The relation between it's value and the type is visible in Appendix B;
- **image:** the path to the product's image stored in the server;
- **company_id:** the product's company primary key, establishing a one-to-many relationships between "*company*" and "*product*" table.

Product Child Tables

The "*product*" table has a set of child tables to cover all its multi-valued properties. Each table has the audition properties, as "*created_at*" and "*update_at*", as well the identification property "*id*", but in this case being a regular auto-generated integer value to make the relation lighter. In addition with a "*product_id*" property referring to the parent table. They differ in the property representing the actual value. Each table has it's own name and restrictions to this property as explained next.

- **products_ingredients:** the table which contains the list of product's ingredients, each row has a string property "*name*" representing the ingredient's name;
- **products_skinneeds:** the table which contains the list of product's provided treatments, each row has a small integer property called "*skin_need*" in the range of 0 (zero) to 17 (seventeen) representing the treatments described in the product's sheet. The relation between it's value and the treatment is visible in Appendix C;
- **products_skinsolarneeds:** the table which contains the list of product's solar provided treatments, each row has a small integer property called "*skin_solar_need*" in the range of 0 (zero) to 5 (five) representing the treatments described in the product's sheet. The relation between it's value and the solar treatment is visible in Appendix D. This table exists because solar care products do not share treatments with regular products as also described in the product's sheet;

- **products_skintype:** the table which contains the list of product's suitable skin types, each row has a small integer property called "*skin_type*" in the range of 0 (zero) to 8 (eighth) representing the skin types described in the product's sheet. The relation between it's value and the skin type is visible in Appendix E;

Company

To represent the companies that owns the products in the system, the "*company*" table was created. This data, is not visible to the final user and is being used for organization and management purposes. It's registers are mainly created from a partnership request and it has a set of attributes that represents company's information and establish a company's representative described as next:

- **name:** the company's name, it's an unique attribute to avoid multiple register of the same company. Even the name is not the best approach for it, it was the chose solution since the system doesn't store any crucial information, e.g, the NIPC or CNPJ, to avoid dealing with data protection Laws;
- **agent_fullname:** the name of the company's representative;
- **agent_role:** the role of the company's representative, e.g, secretary, salesman or CEO;
- **agent_email:** the email of the company's representative, it's an unique attribute to avoid the same representative to multiple companies;
- **phone:** the company's contact phone, in example, it can be a commercial number used by the company or the representative's phone as well;
- **country:** the country of the company's headquarters;
- **assigned_partnership_id:** It's filled when a company's register is created directly from a partnership request, to make it clear that we are dealing with a partnership and so the original information can be checked any time. It refers to the

correspondent partnership request row in the "*partnership_request*" table. By tool's limitations, in the diagram appears as one-to-many relationship, but as this field has an unique constraint, it establish an one-to-one relationship between both tables.

Partnership Request

The "*partnership_request*" table was create to register the partnership requests received from the application. In this way, a set of usefully information are stored so it can be checked and evaluate for an administrator later. This information are crucial to the administrator get in touch with the request owner, proceed with the company's registration and manage the partnership requests. The register only pre-fills a company's information, rather than being fully converted into a company's row, in order to contemplate information changes and keep the original informed data intact. The table's properties are describe as bellow:

- **company_name:** the informed company's name in the partnership form, if not changed it will become the company's name in "*company*" table;
- **agent_fullname:** the informed name of the company's representative in the partnership form, if not changed, it will become the agent_fullname in "*company*" table;
- **agent_role:** the informed role of the company's representative in the partnership form, if not changed, it will become the agent_role in "*company*" table;
- **agent_email:** the informed email of the company's representative in the partnership form, if not changed, it will become the agent_email in "*company*" table;
- **agent_message:** a mandatory cover letter informed by who filled the partnership request form.
- **phone:** the informed phone number of the company in the partnership form, if not changed, it will become the phone in "*company*" table;

- **country:** the informed country of the company in the partnership form, if not changed, it will become the country in "*company*" table;
- **status:** the request status, it's a single-length human readable char value that can have 4 (four) different values being: "W" as waiting to represent pending requests that have not yet been evaluated by an administrator; "D" as denied to represent requests refused by an administrator; "A" to approved, symbolizing requests approved by an administrator and "S" as Denied by Server, to represent the requests that was automatically denied for some reason, e.g., already have an opened request to the informed company;
- **approved_date:** is the date on which the status of the request was set to "S" and was therefore approved. In any other cases, the value is null;
- **comments:** a comment that the administrator has to make before change the status of the request. In this way, ensuring that any decision can be future checked. Moreover, it will also be filled as the reason and description when the register is automatically denied.

Recommendations Form

To deal with the skin forms provided by the users, the "*recommendations_form*" was created. This table has mainly the user's personal information and it's used as the father of the other tables containing the skin's characteristics that will be used to generate the recommend products list. Its attributes are described as follows:

- **user_email:** the user's informed email, if not empty, the user will receive the recommended product's list by email;
- **user_name:** the user's informed name, used to create a more close and customized experience;
- **informed_age:** the user's real informed age, can be later crossed with the machine learning model predicted age to recommend more accurately anti-aging products;

- **authorized:** a boolean field to keep register that the user authorized to have his information stored in order to receive it's recommended products list. It's false by default but no register will be created with a false value in this field;

- **facial_analysis_id:** it's the relationship of the user's form with it's facial analysis. It can be empty, but if it's filled it will provide the analysis results to generate a more accurate recommend products list;

Recommendation Form Child Tables

To complement the "*recommendations_form*" table information, a set of child tables was establish. The 2 (two) tables "*recommendations_formskindiseases*" and "*recommendations_formskintypes*" represents the set of skin diseases and the set of skin types of the user, respectively. Each table contains the log attributes as "created_at" and "update_at", the identification attribute as "id", being a regular auto-incremented identifier and the "form_id" attribute that relates each row to the "*recommendations_form*" father register. The specific attributes of each table is described as next:

- **skin_type:** is the user's skin type in the "*recommendations_formskintypes*" table, it can be up to 3 (three) unique registers to the same related form. It is a small integer property in the range of 0 (zero) to 8 (eighth) representing the skin types described by the user. The relation between it's value and the skin type is visible in Appendix F;

- **skin_disease:** is the user's skin diseases in the "*recommendations_formskindiseases*" table, it can be up to 5 (three) unique registers to the same related form. It is a small integer property in the range of 0 (zero) to 11 (eleven) representing the skin diseases described by the user. The relation between it's value and the skin type is visible in Appendix G;

- **level:** the disease level informed by the user stored in the "*recommendations_formskindiseases*" table. It's a small integer value in a range of 0 (zero) to 3 (three), being 0 (zero) as

not having the disease, 1 (one) as a low disease level, 2 (two) as a medium level and 3 (three) as the highest disease level. The zero value is only used in cases where the disease prediction from the machine learning indicates as not having the disease, otherwise, if the user do not inform a disease, the register will be not created.

Facial Analysis and Predictions

In order to orchestrate and store the information about the user's facial analysis, the table "*facial_analysis*" was created, keeping important information necessary to manage the parallel analysing process. In addition, the table "*analysis_predictions*" store the results of each machine learning model used to automatically extract any disease information from the submitted facial picture and contemplate the information that will be later used to prefill the user's form. It was designed to contemplate an easy addition of additional machine learning models capable of recognized diseases other than forehead wrinkles, initially developed in this work. Each table has a unique set of attributes described as next:

- **image:** this attribute contains the path to the user's facial image stored in the system, it's used to relate the image with the analysis process and form;
- **authorized_to_store:** is the permission given or not by the user to keep their photo in the system after the analysis process has been finished;
- **is_done:** a boolean to determine if the analysis process already over;
- **estimated_age:** the age estimation made from a machine learning model that will be later used to generate a more accurate anti-aging products recommendation. This value is not being stored as the other results because do not correspond to any diseases and is not visible to the final user in any circumstances;
- **error:** the error messages given by the analysis process, if any. It is used for logging purposes and it's not visible to the final user;

- **prediction_type:** In the "*analysis_predictions*" table, this attribute inform the disease analysed in the result record. It has the same value as the skin diseases in the "*recommendations_formskindiseases*" table visible in the Appendix G;

- **value:** the real result of the machine learning model for the correspondent disease, e.g., for the wrinkles recognition model it will be a value between 0 (zero) and 1 (one) representing the probability of the image to contains wrinkles resulted from the model's prediction;

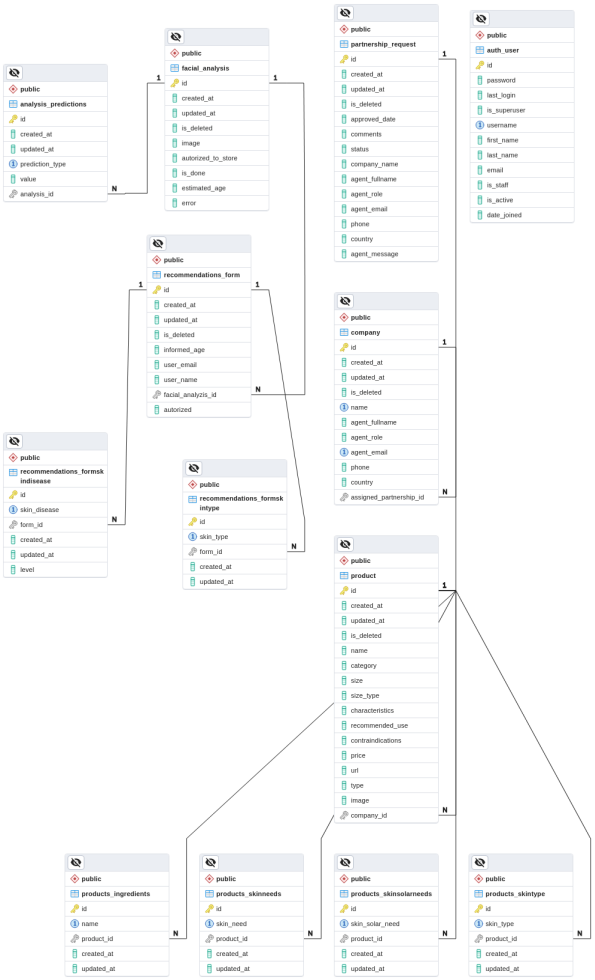


Figure 3.6: Entity Relationship Diagram.

3.5 Software Architecture

The software architecture of a program is a top-level structure model. It includes the arrangement and behavior of the software's in-built components and the framework and principles governing their design [32]. It is something in the form of a blueprint that shows how different modules and components can inter-communicate with each other for coherence and scale.

The indubitable advantages of having a precise software architecture include enhanced maintainability, scalability, and performance of the system that are going to be coherent when several teams work on development. In general, software architecture can be described as a mixture of top-down and bottom-up development processes involving discovery of core requirements of a system, identification of key elements and their interactions in a design.

Mainly, the developed system follows a common form of distributed system architecture called "client-server" model [33]. This model is characterised by a server (commonly known as Back-end) that provides services to other processes, known as clients (commonly known as Front-end). The server Typically has no information about the identities or number of clients that will access it during runtime. In other hand, clients are aware of the server's identity and access its services through remote procedure calls.

For this work, a set of components was establish to contemplate all the system's non-functional requirements [34]. This approach focus on provide a scalable, fast and economical infrastructure. A diagram was created to visual represent the decisions and the communications of the system and it can be seen in the figure 3.7. Each component concept will be explained as next.

3.5.1 Web Client and Front-end Application

In order to improve the system's usability with low effort, a dynamic web client was developed, able to being accessed either from a smartphone, a tablet or a computer directly from a web browser. Represented by the "Web Client" in the diagram, this

component is the main application's client accessed by the final user. In other hand, the "Frontend Application" symbolizes the server where the client static data is store and made available. It's also responsible to manage client-side logic and routing, serving as an intermediary that processes requests from web clients, retrieves necessary resources, and ensures that the correct content is displayed efficiently and securely.

For this work, it was used a Single Page Application (SPA) approach to the front-end development, which is a type of web application that load a single HTML page and dynamically update the content as the user interacts with the application, without requiring a full page refresh [35]. This approach provides a more fluid and responsive user experience by fetching and rendering content asynchronously, often utilizing JavaScript frameworks like React, Angular, or VueJS. This design is well-suited to modern web development needs, where seamless user interaction and faster load times are crucial for user satisfaction.

SPAs have become prevalent in nowadays projects due to their ability to deliver rich, desktop-like experiences directly within the browser, improving performance and reducing server load by minimizing data transfer. This makes them particularly beneficial for complex, interactive applications such as social networks, project management tools, and online collaboration platforms.

3.5.2 REST API

To facilitate seamless communication between the client and server components of our system, it was implemented a REST API. An API is a set of protocols, tools, and definitions that allows different software applications to communicate with each other [36]. It defines the methods and data formats that applications can use to request and exchange information. APIs enable the integration of diverse systems and platforms, facilitating the development of complex applications by allowing developers to build on top of existing services.

With that said, there are various types of APIs available today, including Representational State Transfer (REST) [37], Simple Object Access Protocol (SOAP) [38], GraphQL [39], and gRPC [40]. The choice of a REST API for the current work is motivated by several factors. REST APIs are highly popular due to their simplicity, scalability, and use of standard HTTP methods, which are well-understood and widely supported. They facilitate easy integration with web applications and services, making them ideal for building interoperable systems. It also support stateless communication, improving the scalability and reliability of the application. Additionally, their resource-oriented design aligns well with modern web architectures, allowing for clear and intuitive API endpoints.

This makes it easier for developers to understand and work with the API, reducing development time and improving maintainability. Given these advantages, REST APIs was the followed approach for the current project, aligning with the needs for a scalable, efficient, and easy-to-integrate solution, supporting the distributed nature of the client-server architecture. It serves as the backbone for data exchange, ensuring that the frontend can effectively interact with the backend services to meet the system's non-functional requirements.

3.5.3 Relational Database

A relational database is a type of database that structures data into interconnected tables. Each table, or relation, contains rows representing individual records and columns representing the attributes of those records. The relationships between tables are established using primary keys, which uniquely identify each record within a table, and foreign keys, which reference the primary keys in other tables [41]. This structured approach facilitates efficient data retrieval and management, data integrity, and normalization to minimize redundancy and ensure consistency.

For this work, a relational database was used to store all the relevant data that contemplates the system requirements. The provided reliability and efficiency improve the user trust in the recommendation system, maintaining consistent and accurate data. This

organization facilitates the creation of complex queries needed to generate personalized product recommendations based on multiple criteria.

3.5.4 Message Broker and Task Queue Worker

To manage and distribute asynchronous tasks or jobs across multiple workers or processes, a task queue mechanism is used in software systems. It helps decouple task execution from task initiation, allowing for improved scalability, fault tolerance, and performance. In a task queue system, tasks are placed into a queue by producers, the REST API, while workers or consumers retrieve tasks from the queue and execute them independently [42], [43].

This enables the efficient processing of time-consuming or resource-intensive tasks without blocking the main application thread. Task queues are commonly used in web applications for handling background processing tasks such as sending emails, generating reports, processing data uploads, or, in this case, process the user's submitted picture with the machine learning models.

In addition, a task queue broker is a component that facilitates the communication between producers and consumers in a task queue system. It acts as a centralized message broker, managing the queuing, routing, and delivery of tasks between different components of the system. Popular task queue brokers include RabbitMQ, Apache Kafka, and Redis. These brokers provide robust messaging features such as message queuing, delivery guarantees, message acknowledgment, and message persistence, ensuring reliable and efficient task processing.

3.6 Neural Network and Deep learning

Being the most common type of computational model used today and the basis of most modern machine learning applications, neural networks are inspired by the structure and function of the human brain and consist of a nominal collection of interconnected nodes or neurons that process data in layers [44]. These models are designed to recognize learned

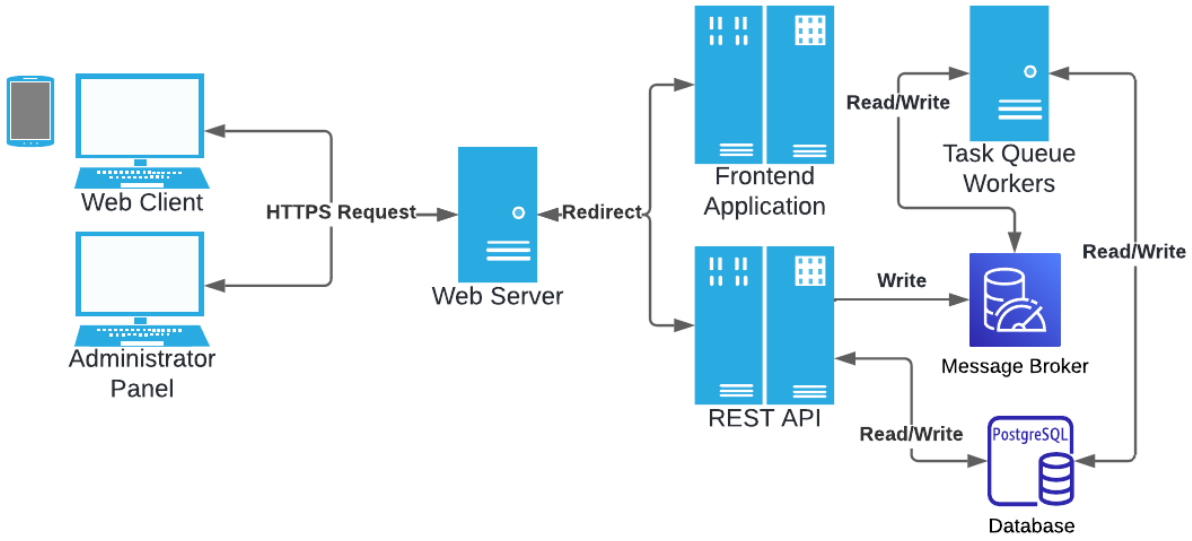


Figure 3.7: System Architecture Diagram.

patterns in a data set and become capable of making accurate decisions and predictions to unknown information. Each neuron in the network receives an input, processes it using weights and biases, and then sends it via an activation function to the next layer.

Deep learning models raises as a subset of neural networks, being a neural network machine learning model with a big amount of layers [45]. It's makes use of large amounts of data and computational power to automatically learn complex representations and features from raw input. Deep learning models like Convolutional Neural Networks and Recurrent Neural Networks have shown state-of-the-art performance in tasks ranging from object detection to machine translation. This type of model performs very well when the task is about high-dimensional data, given it's capability to capture hierarchical patterns and representations.

In addition, machine learning models can be majority classified as supervised or unsupervised, that differ primarily in the type of data used and the learning objectives. Supervised learning concerns about training a model on labeled data, where each input has a corresponding output predefined, allowing the model to learn the shared patterns between them. This method is often used for tasks such as classification and regression.

In other hand, unsupervised learning works with unlabeled data, which means that is no description or explicit category, and the model aims to identify patterns, structures, or groups in the data without guidance. The most typical applications of unsupervised learning are techniques such as clustering and dimensionality reduction, being most often explored and used to discover hidden relationships in data.

3.6.1 Convolutional Neural Network

Computer vision is a field of artificial intelligence that enables machines to interpret and understand visual information from the world, such as images and videos. It involves techniques for acquiring, processing, analyzing, and extracting high-dimensional data to make decisions or provide insights. Applications of computer vision include facial recognition, object detection, image segmentation, and autonomous driving, where the goal is to replicate human vision capabilities in a computational context.

A Convolutional Neural Network (CNN) are a specialized type of neural network designed specifically for processing images. As a subset of deep learning, CNN consist of multiple layers that automatically learn to extract hierarchical features from input data through convolution operations, pooling, and fully connected layers. Typically employed in supervised learning, CNN are trained on labeled datasets where each image is associated with a specific label, allowing the network to learn to recognize patterns and make accurate predictions. This architecture's ability to capture spatial hierarchies and reduce dimensional complexity has made CNN fundamental in advancing the field of computer vision [46].

For the present work, a CNN model was developed to classify facial images as containing and not containing forehead wrinkles. The model was trained with a set of collected and manually classified data. Using a CNN approach for wrinkle detection leverages its ability to automatically learn and extract complex features from images, providing high accuracy and robustness compared to traditional methods. This approach efficiently handles variations in lighting, scale, and orientation, making it well-suited for the current

work goals of analyzing self-taken pictures.

3.7 Fine Tuning and Transfer Learning

Transfer learning involves taking resources learned from a problem and leveraging them into a new, similar situation. Transfer learning is usually done for tasks where a dataset has little data to train a full-scale model from scratch. Commonly, to execute the transfer learning process, layers of a previously trained model are taken, and these layers are frozen to avoid the destruction of any information they contain during the next training rounds. New trainable layers must be added after the frozen layers (which, in the training process, are learned to transform old features into predictions in a new dataset). The last optional step, fine-tuning, consists of unfreezing the entire model you obtained above (or part of it) and retraining it on the new data with a meager learning rate. These strategies can improve significantly by incrementally adapting pre-trained features to new data [47].

Transfer Learning and Fine Tuning strategies have proven relevant for solving several deep learning problems in recent years. Using pre-trained network weights improves the training process, reducing time and obtaining better results, even with low data availability. Both approaches are efficient for various activities related to image classification [25], [48]. This type of strategy reuses the network's previous learning, which is already capable of accurately identifying less complex shapes such as lines and points. It also guides your understanding of more complex forms for the desired activity, such as complete formats.

Even though they are complementary, their application differs, primarily due to how the original network is adapted. Transfer Learning is focused on interpreting the model's results, directing them to the new problem. Fine-tuning works on reworking the weights of part of the layers of the original network, retraining them to interpret the data in a targeted way until reaching the expected final result, requiring greater computational resources. Due to the low amount of data available, the use of these strategies proved fundamental for this work.

3.8 Wrinkles Detection Approach

This section reports the activities developed to create and validate the techniques based on a CNN for recognizing expression marks through facial photographs. The experimental dataset is described in subsection 3.8.1. The process of cropping the region of interest in the images is in subsection 3.8.2. subsection 3.8.3 describes the gray balance applied. Details about the CNN architecture and training parameters are in subsections 3.8.4 and 4.3.4, respectively. A discussion about the training process as in subsection 4.3.5. Finally, the wrinkles detection results are in subsection 4.3.6.

3.8.1 Experimental Dataset

We did not find public databases labeled with the occurrence of wrinkles in forehead region, so we had to combine images taken from a facial image database with photos taken from internet searches. We worked with a small set of images because we could not use all the photos in the dataset due to problems related to the face's position/orientation, the occlusion of the forehead region, and difficulties associated with the manual labeling process (that demands specialists).

Images from the Color FERET database [1], [2] was used to train and validate the CNN used in our approach. This base aims to encourage the development of technology in facial recognition. It consists of 11338 unclassified photographs of the face, measuring 512x768, collected between 1993 and 1996 from 1,208 candidates in different positions.

Among these 11,338 face images, only those showing the candidate's face from the front were selected, totaling 1,700 unclassified images. These were classified manually, discarding those in which it was impossible to see the entire forehead clearly – totaling 150 images (75 with facial wrinkles; and 75 without).

Furthermore, to test the model, a bank of 50 images was composed, randomly selected from the internet, of facial photographs with a wide variation in position, light, and environment, which contained a clear view of the individual's forehead. The new images vary from each other and are related to the training dataset, ensuring that the model is

tested with information never seen before.

3.8.2 Cropping the Region of Interest

Each image in the experimental database (one example in Figure 3.8a) was cropped, selecting only the volunteer's forehead in an elliptical shape. To detect the face, we used the Multi-Task Cascaded Convolutional Neural Network (MTCNN) facial identification model [27], which establishes a box around the faces identified in the photograph (Figure 3.8b). Next, we identify facial key points with the MediaPipe tool [49] (Figure 3.8c).

The key points are used to perfect the cut of the forehead, using the eyes and the top of the image as a base to preserve as many expression lines as possible. The cropped image is converted to grayscale (Figure 3.8d), and its edges are discarded to eliminate any disturbances, such as hair, hats, and shadows. The selected area of interest has an elliptical shape in an image with dimensions 60x200, containing only information relevant to the final objective of the algorithm (Figure 3.8e).

3.8.3 Gray Balance

The diversity of skin tones is a recurring problem in research related to facial recognition. We use gray balance to convert the cropped part into a gray scale image, in order to prevent that to happen. The equation 3.1 calculates the average global brightness, covering all pixels of all selected images. Where I denotes the set of images, P represents the set of pixels in each image and b indicates the pixel brightness. By taking this approach, in contrast to simply selecting a fixed tone, it is possible to mitigate the distortion caused by balancing the tones around the dataset.

$$G = \frac{\sum_i^I (\frac{1}{P_i} \sum_p^{P_i} b_p)}{I} \tag{3.1}$$

Each image pixel has its brightness adjusted from the calculated average, represented by the equation 3.2. The change in brightness considers the relationship between the pixel value and the total image average. This process normalizes the photograph as a

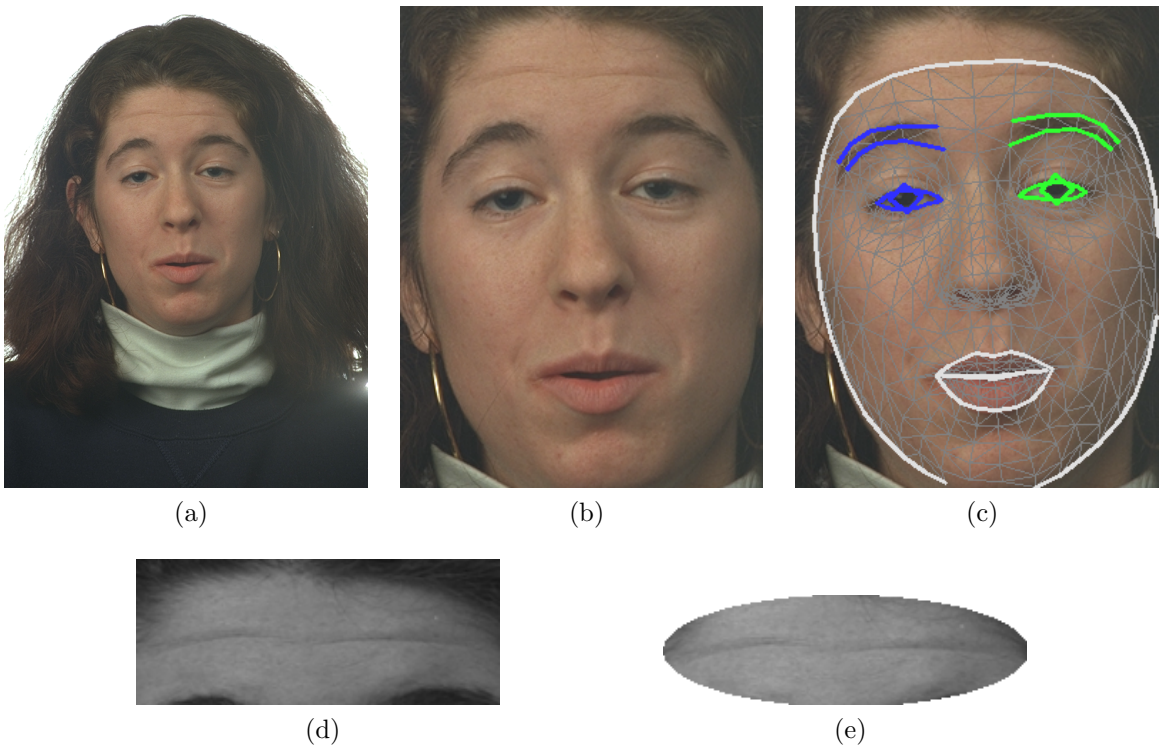


Figure 3.8: Steps for cropping the region of interest in a image: a) original image; (b) cropped face; (c) facial landmarks; (d) forehead cut out together with the gray balance applied; and e) selected region of interest.

whole while preserving the contrast between the lines and its characteristics to darken light tones and lighten dark tones without losing relevant levels of information.

$$\forall(p) \in (P_i), B(p) = \frac{G.b_p}{\frac{1}{P_i}\sum_{p}^{P_i} b_p} \tag{3.2}$$

3.8.4 CNN architecture

The neural network used to classify Wrinkles in this paper is based on the VGG16 [50] model. We work with Transfer Learning and Fine-Tuning, considering the weights of VGG16 previously trained with the ImageNet database. The CNN was retrained to enable it to recognize wrinkles. The initial layers of the model remained with frozen weights, while the others were being retrained to better adapt to the new problem. A

representation of the complete architecture can be seen in Figure 3.9.

The model is initially loaded without the "top," which concerns the network portion that makes the classification decision after the convolutional layers. The new "top" developed is composed by: a layer that rasterizes the matrix coming from the last convolutional layer; a fully connected layer with ReLU activation function; a DropOut layer, which randomly discards 20% of the incoming neurons to avoid overfitting the model; and a last fully connected layer with Sigmoid activation function, which brings the result of the image classification.

Additionally, some initial layers have been added. These, known as preprocessing layers by the TensorFlow [51] tool, aim to modify the input data before it reaches the real layers of the network. Different sets of preprocessing layers were tested, and the best results obtained were using a normalization layer, which brings all pixel values to a range between 0 and 1, and a rotation layer, which rotates the image horizontally and/or vertically, increasing the quantity and diversity of data used. It is worth mentioning that this type of layer is only present during model training and does not impact the final result.

3.9 Conclusion

This chapter presents the proposed methodology for the development of the web system and the wrinkle recognition algorithm. The web system rules were specified in different diagrams, clarifying the project scope and goals. Which focuses on promoting the use of thermal water-based cosmetic products with an easy and intuitive approach, sending the recommended list by email, allowing the user to manually fill in their skin information, and accepting partnership requests from other providers. As well as making use of a machine learning approach to generate the recommended products list more accurately.

The wrinkle detection algorithm is based on a CNN approach, using techniques such as transfer learning and pre-processing to improve the classification quality. A dataset was acquired by mixing images from the FERET Database [1], [2], for training and validation,

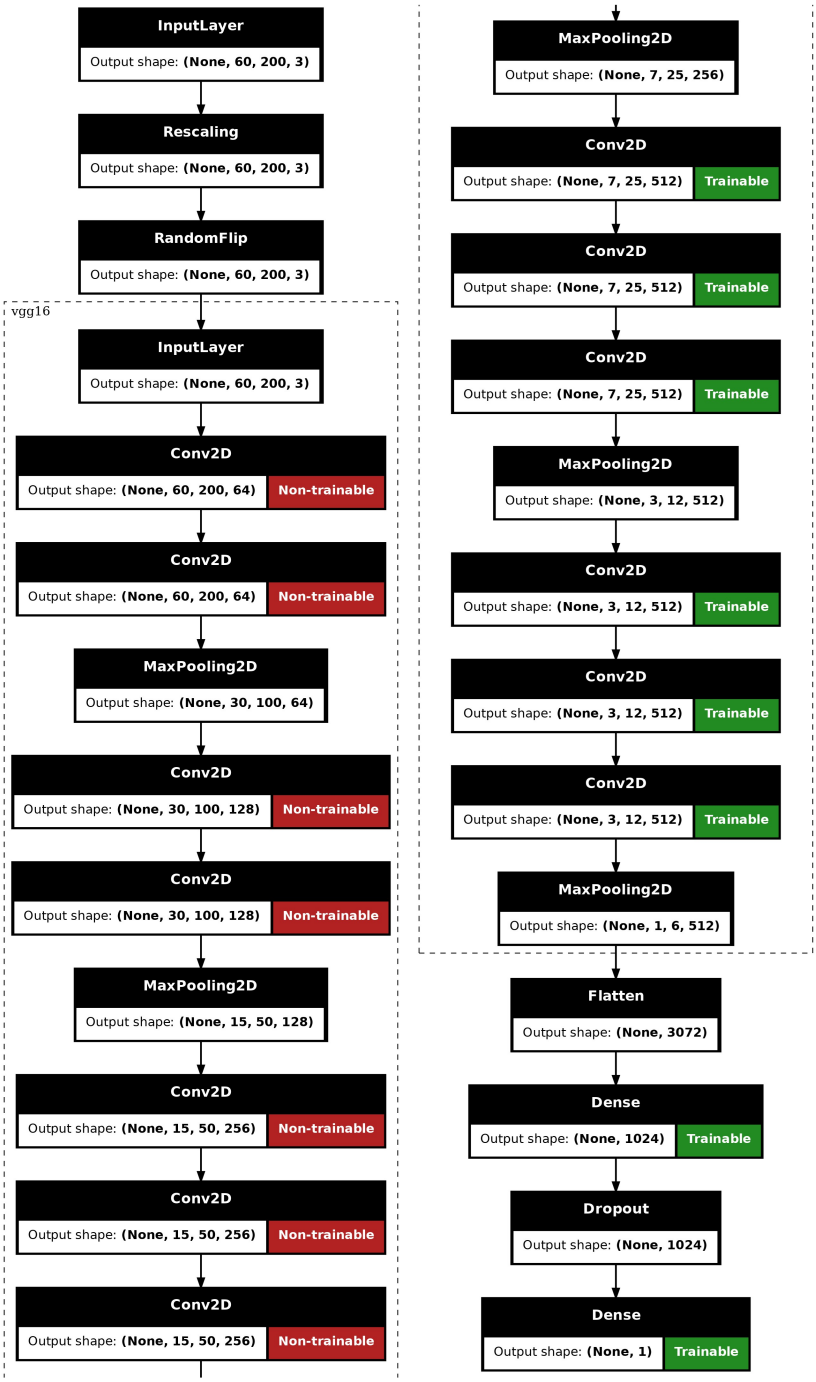


Figure 3.9: Architecture of the CNN model explored to Identifying Wrinkles

with manually collected images, for testing. To compensate for the small number of available images, the developed model was trained with the specific region of interest of each image and with a partially balanced dataset.

Chapter 4

Development and Implementation

This chapter will contemplate the development and implementation steps that encompass the proposed web application and neural network. Aiming the reliability and functionality of the project the set of used tools will be explained, together with their implementation to execute the methods presented in the Chapter 3. The architecture and methods of training and validation of the developed CNN model will be also approached.

4.1 Back-end Implementation

This section approaches the implementation strategies and tools used in the back-end of the web application. In the diagram 3.9, it's possible to visualize that a set of components are present in the server side, each one has their uniquely and need a specific method to be implemented, described as next.

4.1.1 Main Tools

To move forward with the implementation of a system, it is important to evaluate and choose the available tools that best meet the needs of the project. Python [52] is being used as the programming language, due to the possibility of developing the neural network and API similarly, as well as the researchers' previous knowledge of the tool. To keep the code

quality, the Black [53] tool is being used, which automatically formats the code following the code patterns specified in the PEP8 (a Python document with the recommended practices for the language).

Django is an open-source Python web framework that encourages rapid development and clean, pragmatic design [54]. It follows a "battery included" philosophy, which brings with it a set of useful optional tools that solve common web development problems, to improve the development process. Considering the project's nature and needs, together with the researchers' previous knowledge of the tool, Django was chosen for the API implementation, bringing a lot of fast and ready-to-go solutions. In addition, Django was not naturally built to develop REST APIs and so, the open source Django REST Framework (DRF) tool [55] was used to complement it, bringing a flexible toolkit for building Web APIs.

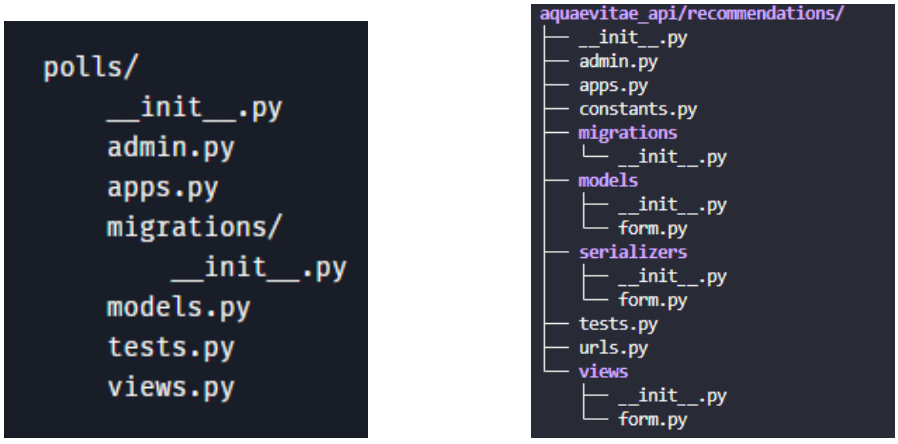
For the relational database in the project, the open-source object-relational database PostgreSQL [56] was chosen, given its high compatibility with other tools such as Django and the capability of Fuzzy Matching (that identifies similar, but not identical elements) strings in the database, not available in other options. A Distributed Task Queue open-source tool named Celery [57] was used, given the need to execute heavy code asynchronously and its high compatibility with Django, which provides a ready-to-use task queue with a focus on real-time processing. In addition, Redis [58], an in-memory key-value database, is the message broker, due to its compatibility with Celery. To finish, NGINX [59] was used as the web server, since it was already being used in the available deployment server.

4.1.2 Projects and Apps

The first step while developing a web application with Django is to initiate the "Project" which generates a directory with a collection of settings for the Django instance, including database configuration, Django-specific options, and application-specific settings. In this case, the project is called "aquaevitae_api", representing the system's REST API to be

developed. Inside the generated folder, a default "App", with the same name as the project, will automatically be created, which contains all the settings and configuration necessary in the development process.

Django Apps is a Python package that follows a structure convention auto-generated by the framework, contemplating a web application's main needs. Each App represents a functional part of the project while the project represents a collection of configurations and apps for a particular website. Important to note that both generated structures can be modified according to the project, e.g., the picture 4.1a shows an App default generated structure by Django, while the picture 4.1b is the final structure of the "Recommendation" App in the developed system.



(a) (b)

Figure 4.1: Comparison between the Django auto-generated App structure (a) and the final structure of the "Recommendation" App (b) in the work.

In this work, a lot of changes were made to the original Apps and Projects structures in order to contemplate the project's functionality and quality needs. As well, a set of Apps was created to modularize the different developed features. The final version of each one is described as next.

- **aquaevitae_api:** it is the project and the default App created. It mainly contains all the project's configurations and settings, including Celery setup, URLs, and others. It also has a set of base structures that can be used in the whole project, as

the Base Model explained in section 4.1.3;

- **analysis:** it is the App responsible for the image analysis process. It contains the Celery Tasks, models to orchestrate the analysis, the image storage directory, and the API Routes to start and analysis and get the analysis results;

- **companies:** an App to represent the company's behaviours. It contains the models representing a company in the database and the company's page configuration in the admin panel. It has no API routes since none company's information is available to the final user;

- **partnerships:** the App representing the partnership request flow, which contains the routes to generate a partnership, as well as its models and its panel on the admin page. It also contains the email-sending process;

- **products:** the App to represent the flow of products, with the routes to receive the general list of products and the recommended list, generate the product's score, as well as the product page of the administration panel;

- **recommendations:** it's the App that contemplates the form-related features, with the routes to receive the user's information and send the form results by email.

Figure 4.2: Project and Apps folder structure.

4.1.3 Models

A development good practice nowadays is to make use of an Object Relational Mapper (ORM), a programming technique that allows developers to interact with a database using object-oriented code instead of SQL, simplifying database operations and improving code maintainability by mapping database tables to classes and their instances. This way, Django provides the Django ORM, which it's not necessary to be used but, given its high compatibility with the other chosen tools, it was a good approach.

The initial database setup and configuration are automatically done by Django while generating the Project, the only really needed first step is to configure the database credentials and type, in this case, PostgreSQL, in the provided settings file. Mostly, the Django ORM is based on the "Models" which is basically a Python Class representation of a database entity. The models have a set of default attributes as the table name and the auto-generated row primary key, which can all be modified in the code.

To create an ORM model it's necessary to define a Python class inheriting, a Model class provided by the tool. When a model is created, a correspondent database table is also created, with the columns being the class attributes, which have to be declared as one of the field types provided by the Django ORM. Each database row is treated by the application as an instance of the declared class, in any type of operation.

To avoid duplicated work and keep a pattern between the tables, a "BaseModel" was created and then all the following models were created inheriting from it. For that, Django ORM allows the creation of an Abstract model which is a Model Class representation that does not correspond to any database table. The BaseModel contains all the default table attributes, such as the UUID primary key and the log purpose attributes, and the soft-delete behavior, as well. Its declaration is visible in the figure 4.3. Each table described in the section 3.4.1 has it's equivalent Model with all the specified attributes.

```python
class BaseModel(models.Model):
    id = models.UUIDField(primary_key=True, default=uuid.uuid4, editable=False)
    created_at = models.DateTimeField(auto_now_add=True)
    updated_at = models.DateTimeField(auto_now=True)
    is_deleted = models.BooleanField(default=False, blank=True, null=False)

    class Meta:
        abstract = True

    def delete(self, *args, **kwargs) -> Tuple[int, Dict[str, int]]:
        if self.is_deleted:
            return super().delete(*args, **kwargs)
        else:
            self.is_deleted = True
            return self.save(
                update_fields=[
                    "is_deleted",
                ]
            )

    def __str__(self):
        if hasattr(self, "name"):
            return "%s" % (self.name)
        return "%s id: %s" % (self._meta.verbose_name.title(), self.pk)
```

Figure 4.3: BaseModel Python declaration.

4.1.4 Serializers

One important feature provided by the Django REST Framework is the Serializers, described as a way to convert complex data, like Model instances, to native Python data types, that can then be easily rendered into the required content type, such as JSON in this project. Serializers also provide deserialization, allowing parsed data to be converted back into complex types, after first validating the incoming data.

A Serializer is created by inhering the Serializer Class provided by DRF, or, in this case, inhering from the Model Serializer Class, which creates a serializer for a specified model, automatically converting the model's attributes to the corresponding data types. In practical terms, the Serializer is a bridge between the API received data and the database register, and so, can be customized to receive specific types of data and process it the way it's needed.

```python
class Form(BaseModel):
    facial_analyzis = models.OneToOneField(FacialAnalysis, blank=True, null=True, on_delete=models.DO_NOTHING)
    user_email = models.EmailField(blank=True, null=True)
    user_name = models.CharField(blank=False, null=False, max_length=50)
    informed_age = models.PositiveSmallIntegerField(blank=False, null=False, default=0)
    autorized = models.BooleanField(blank=False, null=False, default=False)

    @property
    def aging_level(self):
        if not self.facial_analyzis:
            return 0

        aging = self.facial_analyzis.estimated_age - self.informed_age

        if aging < 0:
            return 0

        return (aging / settings.MAX_AGE_DIFFERENCE) + 1

class FormSkinType(OneToManyBaseModel):
    form = models.ForeignKey(Form, related_name="skin_types", on_delete=models.CASCADE)
    skin_type = models.SmallIntegerField(
        blank=False,
        choices=FormSkinTypeChoices.choices,
        default=FormSkinTypeChoices.NOT_SURE,
    )

    class Meta:
        unique_together = (
            "form",
            "skin_type",
        )
```

Figure 4.4: Form and Form Skin Type Model declaration.

In this project, different serializers were created to contemplate the multiple scenarios required by the API routes. The serializers are being used to specify the way data has to be sent to the API, as well as how it's being returned to the client. One-to-many relationships, as in the Product database with the product's skin types, are commonly received as a list of the correspondent type but have to be converted to a Django Model before it can be related to the final register, all of this is being done as a serializer. Likewise, the way the related values are being returned to the client has to be specified as a serializer, figure 4.6. Any kind of data validation, is also a serializer responsibility, together with other operations that have no need to be in the Model level.

The Partnership Serializer is responsible for validating the partnership's condition

before proceeding with any system operation. It receives the partnership form data and is used to find any other correspondent register. If any condition is satisfied, the partnership register will be saved as "DENIED BY SERVER" with the detailed reason specified in the comment field, as shown the Figure 4.5.

4.1.5 ViewSets

A Django View is a function or class that handles web requests and returns web responses, enabling the creation of dynamic web pages by encapsulating the logic required to process user input and generate appropriate outputs, thereby simplifying the development and maintenance of web applications. With that, a DRF ViewSet is an abstraction that simplifies the creation of REST APIs by combining multiple Django Views into a single class, allowing for cleaner, more efficient code and streamlined routing for common actions like list, create, retrieve, update, and delete.

DRF provides a way to generate default REST API behaviors for a specific Django model with the ModelViewSet class. To create a ViewSet is necessary to declare a class inheriting the ModelViewSet class. Some attributes have to be specified, such as the "queryset" which receives a model query, e.g., to list all the model's database register and a default serializer.

DRF do not provide a native way to have different serializers for different routes in the same ViewSet, in this way, the route's expected data is the same for any API operation, which is a problem in situations where, e.g., a creation request receive a specific structure of data, but the listing request has a totally different one. To bypass that, a method was developed where each action has a correspondent serializer attribute with the action specified as a prefix in the attribute name, and a custom base ViewSet was implemented able to relate the actions with the serializers, as shown in Figure 4.7.

Within the viewset is possible to execute custom code beside the default operations. Figure 4.8 shows that together with the facial analysis database register creation is being triggered the Celery tasks to proceed with the analysis process. Moreover, using the

```python
class DetailProductSerializer(serializers.ModelSerializer):
    ingredients = serializers.StringRelatedField(many=True, allow_empty=False)
    skin_types = DetailSkinTypeSerializer(many=True, allow_empty=False)
    skin_needs = DetailSkinNeedsSerializer(many=True, allow_empty=False)
    skin_solar_needs = DetailSkinSolarNeedsSerializer(many=True, allow_empty=True)
    image = serializers.ImageField(required=False)
    score = serializers.SerializerMethodField("get_score", required=False, read_only=True)

    class Meta:
        model = Product
        exclude = ["company"]
        editable = False

    def to_representation(self, instance):
        data = super().to_representation(instance)
        category = data.pop("category")
        data["category"] = DetailCategorySerializer().to_representation(
            {"code": category, "verbose_name": instance.get_category_display()}
        )
        type = data.pop("type")
        data["type"] = DetailTypeSerializer().to_representation(
            {"code": type, "verbose_name": instance.get_type_display()}
        )
        return data

    def get_score(self, instance):
        request = self.context.get("request")
        if request.GET and request.query_params.get("form_id"):
            try:
                form = Form.objects.get(id=request.GET.get("form_id"), is_deleted=False)
            except Form.DoesNotExist as e:
                raise serializers.ValidationError("Form not found")
            except ValidationError as e:
                raise serializers.ValidationError(e)

            return instance.get_score(form)

class DetailSkinTypeSerializer(serializers.ModelSerializer):
    code = serializers.IntegerField(source="skin_type")
    verbose_name = serializers.CharField(source="get_skin_type_display")

    class Meta:
        model = SkinType
        fields = ["code", "verbose_name"]
        editable = False
```

Figure 4.5: Partnership Request Serializer.

```python
class CreatePartnershipSerializer(CountryFieldMixin, serializers.ModelSerializer):
    phone = PhoneNumberField(allow_null=True, required=False, allow_blank=True)

    def validate(self, attrs):
        q_partnerships = PartnershipRequest.objects.filter(
            Q(agent_email=attrs["agent_email"]) | Q(company_name=attrs["company_name"]),
            country=attrs["country"],
            is_deleted=False,
        ).order_by("-created_at")

        waiting = q_partnerships.filter(status=RequestStatusChoices.WAITING).first()
        closed = q_partnerships.filter(
            ~Q(status=RequestStatusChoices.WAITING),
            updated_at__gte=timezone.now()
            - timedelta(days=settings.REQUEST_TIME_LIMIT),
        ).first()

        if waiting:
            attrs["status"] = RequestStatusChoices.DENIED_BY_SERVER
            attrs["comments"] = _(
                "Already have an OPEN request for this company created by %s of ID %s\n"
            ) % (waiting.agent_email, waiting.id)
        elif closed:
            attrs["status"] = RequestStatusChoices.DENIED_BY_SERVER
            attrs["comments"] = _(
                "There is a recent CLOSED request for this company created by %s of ID %s\n"
            ) % (closed.agent_email, closed.id)

        return super().validate(attrs)

    class Meta:
        model = PartnershipRequest
        exclude = ["status", "approved_date", "comments", "is_deleted"]
```

Figure 4.6: Product and Product Type details serializer.

DRF Mixins models, it has the possibility to just create the needed views actions, in the example, only the create and retrieve routes are available.

```python
class BaseViewSet(viewsets.GenericViewSet):
    def get_serializer_class(self):
        serializer = getattr(self, self.action + "_serializer_class", None)
        if not serializer:
            serializer = getattr(self, "serializer_class")
        return serializer

    def get_queryset(self):
        return super().get_queryset().filter(is_deleted=False)
```

Figure 4.7: Custom Base ViewSet declaration.

```python
class FacialAnalysisViewSet(mixins.RetrieveModelMixin, mixins.CreateModelMixin, BaseViewSet):
    queryset = FacialAnalysis.objects.all().order_by("-created_at").prefetch_related("predictions")
    serializer_class = ResponseFacialAnalysisSerializer
    create_serializer_class = CreateFacialAnalysisSerializer
    retrieve_serializer_class = DetailsFacialAnalysisSerializer
    parser_classes = [MultiPartParser,]
    permission_classes = [
        permissions.AllowAny,
    ]

    def perform_create(self, serializer):
        instance = serializer.save()
        tasks = (process_wrinkles_facial_analysis.s(analysis_id=instance.id), process_age_prediction.s(analysis_id=instance.id))
        callback = set_as_done.s(analysis_id=instance.id)
        chord(tasks)(callback)

    def create(self, request, *args, **kwargs):
        serializer = super().create(request, *args, **kwargs)
        return Response(ResponseFacialAnalysisSerializer().to_representation(serializer.data), status=HTTP_201_CREATED)
```

Figure 4.8: Facial Analysis Viewset declaration.

In this stage, all the API routes were declared which contains all the features available to the client. Each viewset has its unique behavior to accomplish the system's needs. Using the Swagger tool is possible to document all the API Available routes as shown in the figure 4.9.

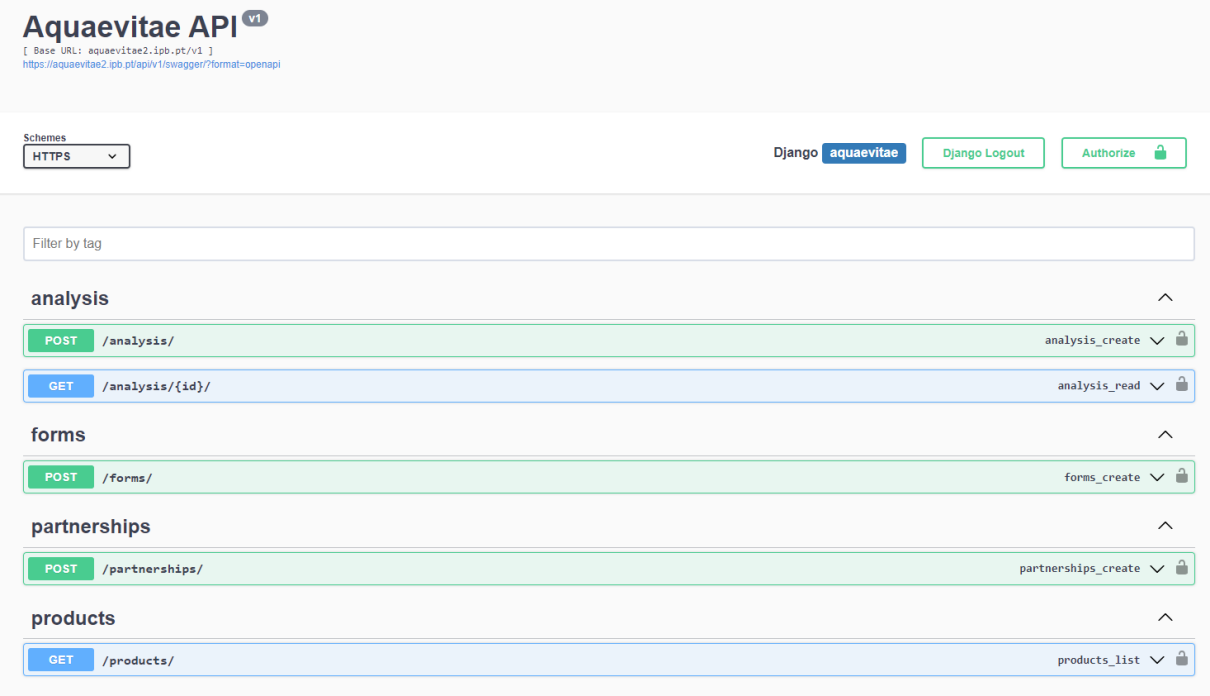


Figure 4.9: Swagger Documented routes.

4.1.6 Django Admin

The Django Admin is a built-in feature of the Django web framework that provides a powerful, out-of-the-box interface for managing the data and models of a Django application. Automatically generated based on the application's models, the Django Admin allows administrators to perform CRUD (Create, Read, Update, Delete) operations on database records without requiring additional coding.

This interface is highly customizable, enabling developers to tailor the admin panel's look and functionality to meet specific requirements. The Django Admin is invaluable for its ease of use, reducing development time, and offering robust features such as search, filtering, sorting, and inline editing, making it an essential tool for efficiently managing and maintaining web applications.

In this project, Django Admin is mainly used to manage the received partnership requests, products, and company registers. The Django default authentication method is being used, which saves some time in the development process and already provides

a robust and safe authorization mechanism. The admin user is able to access the panel with its username and password.

Some customization was needed in order to give the partnership flow the wanted behavior. The form's filled information is automatically attributed to a Company register, allowing to the administrator proceed with one click. The created company and the partnership request will be related to keeping it recorded. While on a company's page, is easy to see all the company's products as well as remove or add new ones.

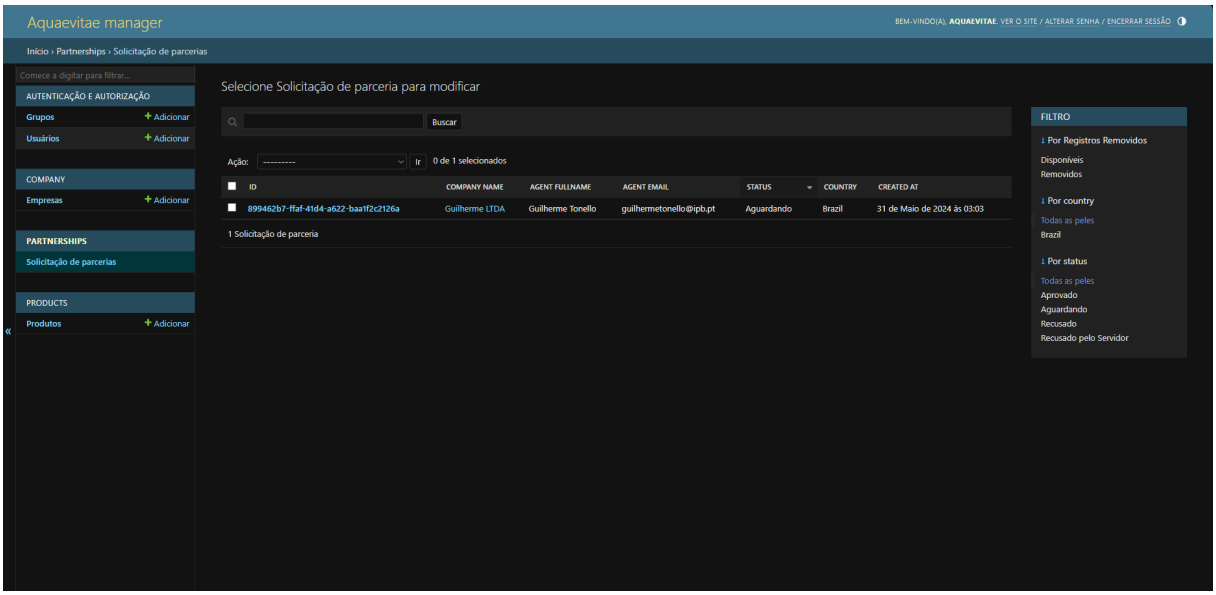


Figure 4.10: Partnership Requests Admin page.

4.1.7 Signals and Mail Sending

Django includes a "signal dispatcher" which helps decoupled applications get notified when actions occur elsewhere in the framework. In a nutshell, signals allow certain senders to notify a set of receivers that some action has taken place. They're especially useful when many pieces of code may be interested in the same events. In addition, Django models trigger a set of signals in common behaviors to add or update a value in the database.

In this work, Django signals are being used mainly to be aware of any changes in the partnership request status. For that, was configured a receiver for the "post_save" signal, sent by the PartnershipRequest model. This signal is triggered after a model instance is successfully saved in the database, both in insert or update operations. With that is possible to identify any status changes and act accordingly.

Every time a partnership request is created without being denied by the server, an email is sent to the application's administrator, and to the email informed in the partnership form, notifying them about the newly created request. If it's denied by the server, the email informed in the partnership form will be notified as well. After created, any status update can be identified and the informed email will be notified as well. To make the email-sending process faster was created a Python Coroutine, using the built-in AsyncIO Python Library, which sends the emails asynchronously. When a form is submitted to the server a correspondent link, leading to the recommended products list, is sent to the user by email (if available).

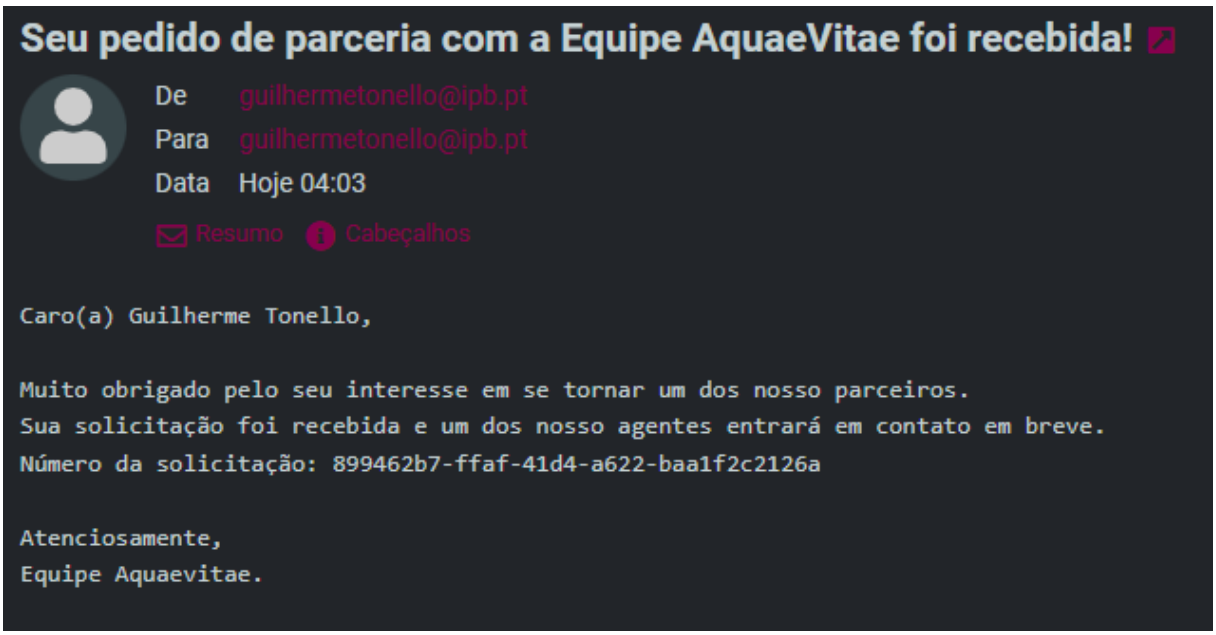


Figure 4.11: Email notification of partnership request received.

4.1.8 Order, Filter and Pagination

REST APIs often include functionalities for order, filter, and pagination to enhance data retrieval and user experience. Ordering allows clients to request data in a specific sequence, such as by date or relevance, making the results more meaningful. Filtering enables clients to refine the data based on specific criteria, which is essential for handling large datasets efficiently. Pagination breaks down the data into manageable chunks, preventing the client from being overwhelmed with too much information at once and reducing server load. These features improve the performance, usability, and scalability of REST APIs, ensuring that users can access precisely the data they need quickly and efficiently.

In this project, mostly filter, pagination, and order properties were given to the product's ViewSet. In this way, is possible to access the system's list of products by filtering by the product's category, product's type, product's skin type, and product's provided treatments. For ordering, besides the default creation date order, it's possible to order by the product's price in ascendent and descent ways. As well it's also possible to search products by their name or by their ingredient's name. A default pagination was configured but is not being used since the page length is 20 (twenty) elements and the system does not exceed this number of products yet.

A custom filter was created to list only the recommended products given a specified form. In this way, it's possible to provide a form identifier in the same route used to list the products. If done, the system will retrieve the form data and generate the product's score. Each product has their own score generated by the Product Model, as shows the Figure 4.12 , and after that, product's are ordered from the highest to the lowest score. In this process, all other filters are being ignored and it's not possible to paginate it.

4.1.9 Server Side Image Processing

The facial analysis process has its unique set of routes featured by the FacialAnalysis ViewSet, one to request an analysis, which receives the user's image as the request body, and one to access the analysis results, which can contain a list of results, an error message

```python
def get_score(self, form):
    score = 0
    product_skin_needs = set([skin_need.skin_need for skin_need in self.skin_needs.all()])

    for disease in form.skin_diseases.all():
        if disease.level == FormSkinDiseasesLevelChoices.NONE:
            continue

        form_skin_needs = set(DISEASE_NEEDS_MAP[disease.skin_disease])

        related_skin_needs = product_skin_needs.intersection(form_skin_needs)

        score += len(related_skin_needs) * disease.level

    score += round(int(9 in product_skin_needs) * form.aging_level)

    form_skin_types = set([skin_type.skin_type for skin_type in form.skin_types.all()])
    product_skin_types = set([skin_type.skin_type for skin_type in self.skin_types.all()])
    related_skin_types = product_skin_types.intersection(form_skin_types)

    score += (len(related_skin_types) + 1) ** 2 if related_skin_needs else 0

    return score
```

Figure 4.12: Product's Score calculation function.

or an empty list in case of the analysis is not completed yet.

When requesting an analysis, the system will save the analysis request in the database, save the sent image in the server, trigger two Celery tasks at the same time to be executed asynchronously, and then return to the client an identifier related to the analysis process.

One of the triggered tasks makes use of a tool called DeepFace [60], described as a lightweight face recognition and facial attribute analysis framework for Python, capable of recognizing faces and extracting data from facial pictures like age, gender, emotion, and race. In this work, the tool is used to predict the user's age based on the submitted image, to create more accurate recommended product lists, and then store the result in the database. This result will be later compared with the user's informed age to determine the user's aging level providing more quality while recommending anti-aging products. The DeepFace tool was chosen given its high capability and efficiency proved in multiple studies [60], as well as its high usability that accomplishes the project's needs with low

effort.

The other task is responsible for the wrinkle recognition process. It starts pre-processing the facial image by cropping the interest zone and leaving it in the expected format by the machine learning model. This step is the same as the neural network pre-processing steps, explained in section 3.8.2 and section 3.8.3. The treated image is then analyzed by the developed model, which returns the probability of containing wrinkles from 0 (zero) to 1 (one). The result is stored in the "analysis_prediction" table with the prediction type, which in this case is 6 (six) corresponding to the wrinkles disease code. A basic Celery Task was developed which loads the developed ML model, exported from the development process as a "keras" file, the first time it's being called and then stored it in memory, to improve the analysis speed.

When both tasks have been completed, it is verified if any of them has ended with an error, if so, the error message will be stored. If more than one task has been ended with an error, the error messages will be concatenated and then stored. After that, is verified the user's storage image authorization, if not provided, the received image is deleted, regardless of error or success. To conclude, the analysis process is set as done.

Using the retrieve route, the client can continuously check the request status until get some result. When not done, the API will simply return an empty result list and an attribute "is_done" set as false. If there is an error stored in the analysis process and there are no valid predictions, an error message will be returned to the user and there's no more reason to keep retrieving the analysis status. If the analysis is already done and there are valid predictions, a list of results will be returned, containing a disease code (6 for the wrinkles) and the disease level to prefill the user's form. The level is calculated as follows, if the result is lower than 0.6, the disease level is set as "None"; lower than 0.70 is set as "Low"; lower than 0.85 is set as "Medium" and higher than 0.85 is set as "High".

4.2 Front-end Implementation

This section will describe the front-end development process, contemplating the website accessible to the final user. Firstly the used tools will be described, followed by the components, pages, and implementation of the system's developed features. Important note that this section does not concern the admin panel, which is being explained in section 4.1.6.

The front-end was developed to be fully usable from a smartphone or a computer, in both, it will be accessed from the browser and it will adapt some behaviors according to the current screen size. From the implemented website is possible to access the page of all the project contributors such as the Aquaevitae, Polytechnic Institute of Bragança (IPB), and Federal University of Technology of Paraná (UTFPR) webpages.

4.2.1 Main Tools

In the rapidly evolving landscape of web development, a variety of powerful tools and frameworks have emerged to streamline and enhance the development process. This introduction provides an overview of several key technologies that were essential for the front-end development process highlighting their functionalities and benefits.

Each of these tools contributes uniquely to the web development ecosystem, providing enhanced functionality, improved developer experience, and more efficient workflows. By leveraging these technologies, developers can build high-quality, maintainable, and scalable web applications, which was essential during the system's development process.

JavaScript and TypeScript

JavaScript is a versatile and widely-used programming language that powers dynamic behavior on websites [61]. It allows developers to create interactive user interfaces, handle events, and manipulate the DOM (Document Object Model). TypeScript, a superset of JavaScript, introduces static typing to the language [62], enabling developers to catch errors early through type checks. This leads to more robust and maintainable code,

especially in large-scale applications. For this reason, it was chosen as the front end programming language.

Bun

Bun is a modern JavaScript runtime like Node.js but designed to be faster and more efficient [63]. It provides a minimalistic and performing environment for running JavaScript code on the server side, facilitating quick development cycles.

React

React is a popular JavaScript library for building user interfaces [64]. It uses a component-based architecture, allowing developers to create reusable UI components that manage their own state. This makes it easier to build complex and dynamic web applications with a clear and maintainable structure.

Tailwind

Tailwind CSS is a utility-first CSS framework that enables developers to design responsive and customizable user interfaces quickly [65]. By using utility classes, developers can apply styles directly in their HTML, resulting in a more streamlined and efficient development process. Tailwind's approach promotes consistency and reduces the need for custom CSS, making it ideal for rapid prototyping and large projects alike. It was used given it's high compatibility with React and Shadcn/ui components.

Shadcn/ui

Shadcn/ui is a design system and component library that provides a cohesive set of UI components for building modern web applications [66]. It integrates seamlessly with Tailwind CSS, allowing developers to quickly assemble aesthetically pleasing and consistent user interfaces. This tool enhances productivity by offering pre-designed components that can be easily customized to fit specific design needs.

Tanstack Query and Tanstack Router

Tanstack Query is a powerful library for managing server-state in React applications [67]. It simplifies data fetching, caching, synchronization, and updates, enabling developers to build responsive and efficient applications with ease. Tanstack Router complements this by providing a declarative routing solution, allowing for smooth navigation and improved user experience [68].

Zod

Zod is a TypeScript-first schema declaration and validation library [69]. It helps developers define schemas for their data models and validate input data against these schemas. This ensures data integrity and helps catch errors early in the development process, leading to more reliable applications. It was mainly used for validate the informed data in the partnerships request form.

Axios

Axios is a promise-based HTTP client for JavaScript that simplifies making HTTP requests to APIs [70]. It supports features like interceptors, automatic JSON transformation, and request cancellation, making it a robust tool for handling network communication in web applications. Its intuitive API and versatility make it a good choice for integrating backend services and REST APIs.

4.2.2 Form Submission

To accomplish the skin form submission requirements, it was created a "form" page. This feature has two possible flows, chosen by the user in the first form step (Figure 4.13), together with the term's acceptance. To implement that and the other steps, it was used a Wizard Form, which is a multi-step form on the same page, allows the user to come back and forward in the process and exit it at any time.

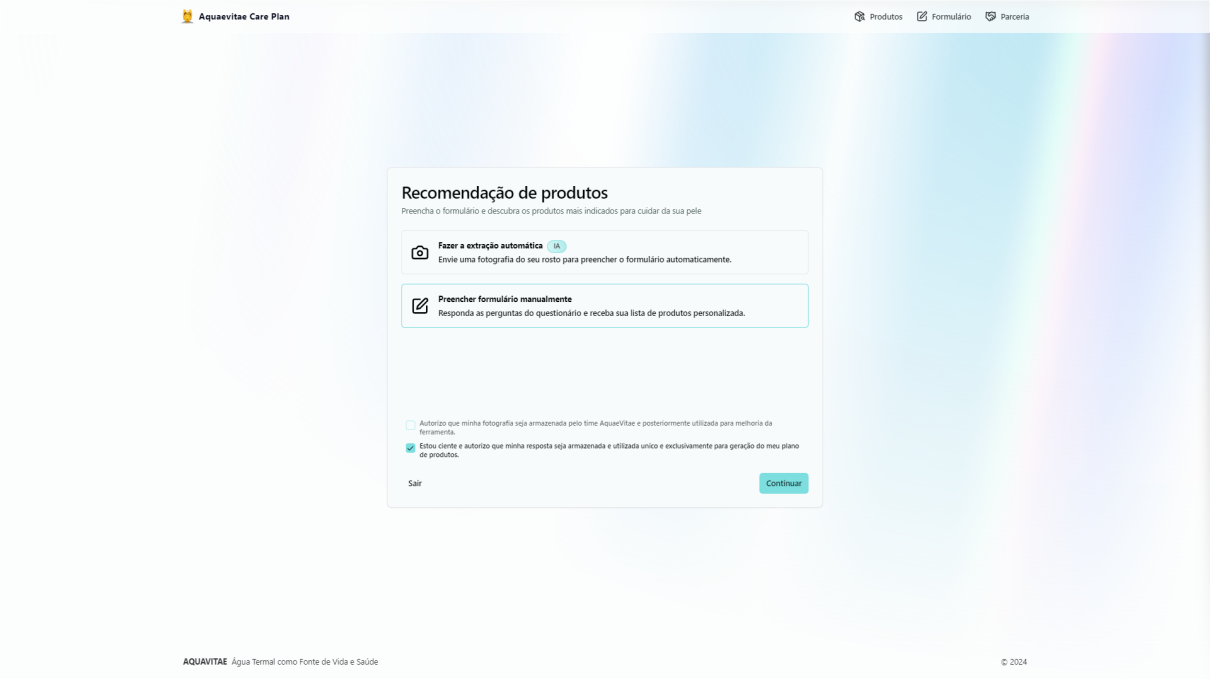


Figure 4.13: Form submission page first step.

A Form component was created to manage the steps and information during the process. Any typed information is saved and can be retrieved or edited before submitting the form. If the user chooses to proceed with the manual process, it will be moved directly to the Skin Type steps able to continue from there, as shown in Figure 4.14. Each skin type is shown as a check button, and when the maximum amount of skin types (3) is exceeded, the first selected one will be deselected. If the "Not Sure" option is marked, all the other options will be unchecked as well.

Moving forward to the skin diseases step, a list of diseases is shown where the user can select up to 5 (five), as shown in Figure 4.15. The user can freely remove any chosen disease. When a disease is selected, a bar is shown below, giving the user the possibility to inform the correspondent disease level between "Low", "Medium" and "High". The last step is to provide the user's personal information, such as age, name, and email (not required). Then the user will be automatically redirected to their recommended products list page.

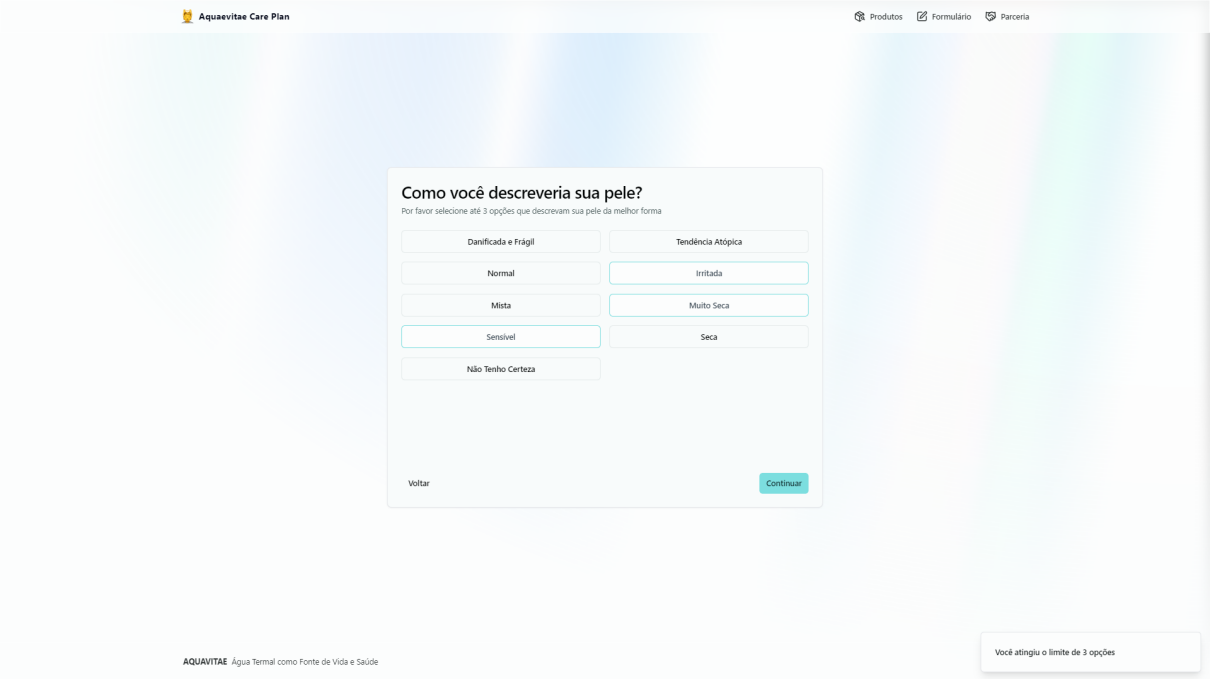


Figure 4.14: Skin Type step in Form submission page.

For the users who chose to proceed with the image submission, a page with the picture instruction is displayed and it's automatically dismissed after a few seconds. Then, the user's frontal camera or webcam is accessed and, using the MediaPipe tool for Javascript, validates in real-time, the number of faces captured in the image, the face size, and the face position, only allowing the picture to be taken when the user fit the displayed mark, as show Figure 4.16. After taken, the photo is sent to the server and the form proceeds as normal to the skin type step. When reaching the skin diseases step, the client will try to retrieve the analysis results, if not available it will keep trying until gets a valid result list, an error, or reaches 40 (forty) tries. When getting a valid result, consisting of a set of disease codes with a respective disease level, it will prefill the disease step with the retrieved data. In this case, setting the detected user's wrinkle level, which can be freely updated or removed by the user before the form submission.

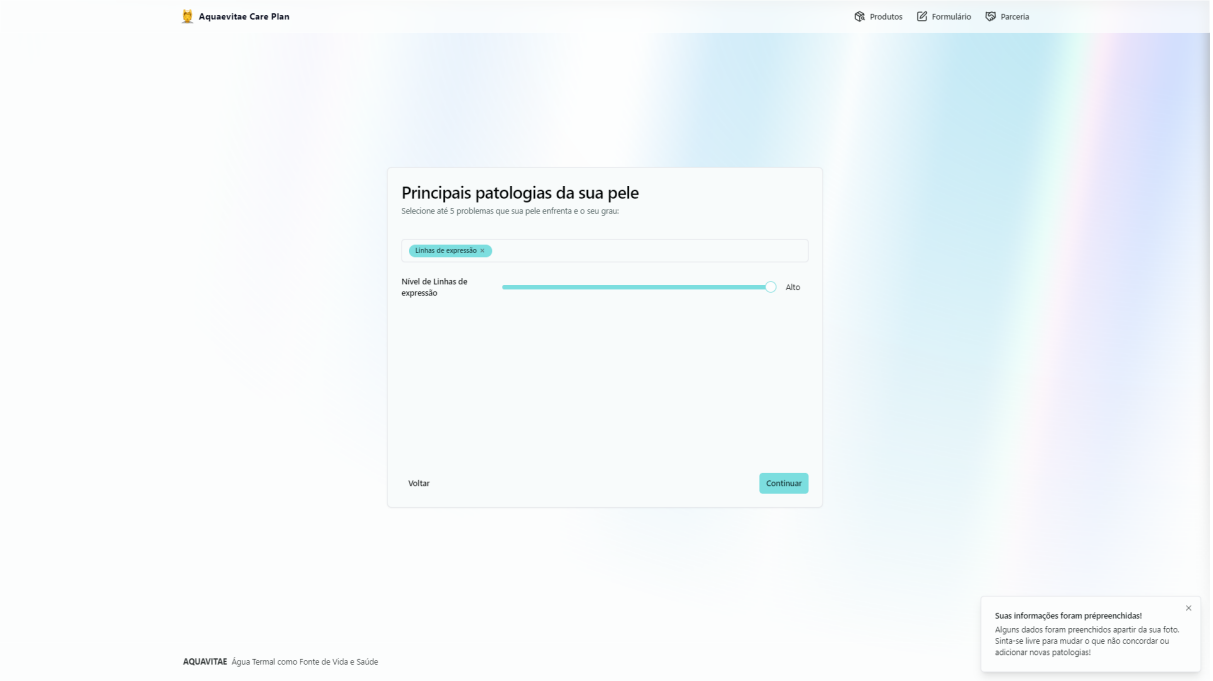


Figure 4.15: Pre-filled Skin Diseases step in Form submission page.

4.2.3 Products List

A product page was created to contemplate both the traditional product list and the recommended products list. Each product is displayed as a card containing some crucial information such as the product name, description, and provided cares. If a card is selected, the other product's information is shown in a lateral drawer component, e.g., the price and the buy button, as shown in Figure 4.17. The products list is fetched from the API when the page is accessed.

For the recommended product list, a form identifier has to be informed in the page URL in the "form_id" parameter. In this way, the application can identify that it's a recommended product list and, while fetching the products from the server, pass the informed ID as a filter parameter, as explained in section 4.1.8. The server will then generate the product list and return it to the client ordered from the highest to the lowest score. Each product will be tagged as "Important", "Recommended" and "Optional" based on its position in the list, as shown in Figure 4.18. As it has no pagination in

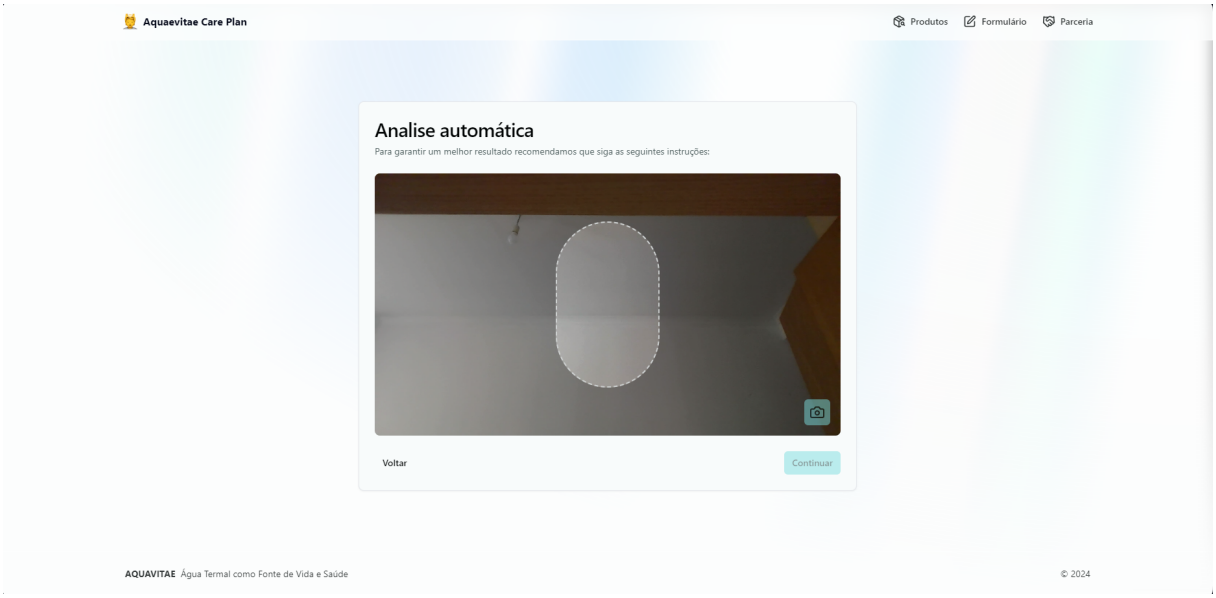


Figure 4.16: Photo Capturing step in Form submission page.

the recommended list, it will be a maximum of 20 (twenty) products. The "Important" products will represent the first 20% listed products, the "Recommended" products, the next 30%, and the "Optional" products, the last 50%. In this way, even with a small number of products or with no highly compatible products, the users will still receive a valid recommend list.The user will receive a link to access the full list at any time by email (if provided) and for each recommended product is possible for the user to access the product's original page using the "Go to store" button, to buy it or get more information about it, as shown in Figure 4.17.

4.2.4 Partnership Request

To give the user the possibility to request a partnership, a Partnership Request Page was created, containing the partnership request form. It solicited the company's name, country, user's name, email, phone number, the user's role inside the company, and a text field so the user could send a cover letter together with the request. This data is then sent to the server and the user can monitor the request by email. The developed page

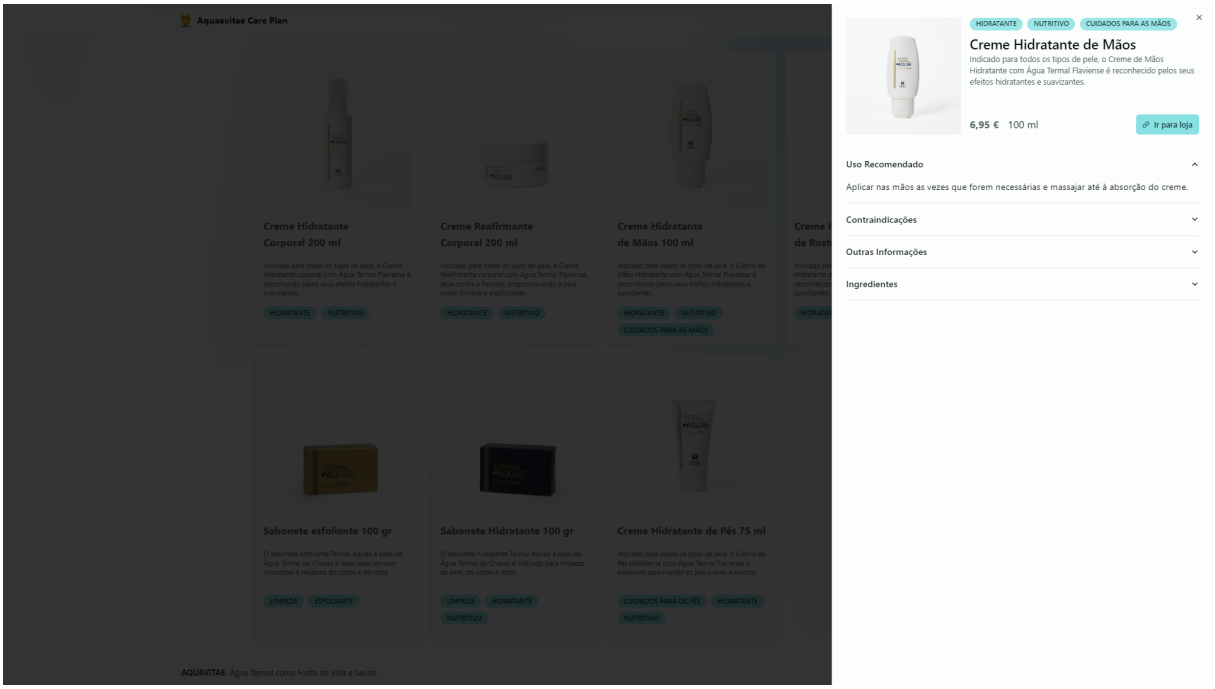


Figure 4.17: Product's details.

can be seen in Figure 4.19.

4.3 Machine Learning Model Development

This section contemplates the development process of the machine learning model for wrinkle recognition. Initially, the used tools will be described, as well as an explanation of the auxiliary neural networks used. The training process will be covered specifying the parameters that impacted the final result. Lastly, the trained model will be evaluated and the results will be discussed.

4.3.1 Main Tools

MediaPipe is an open-source framework developed by Google for building multi-modal, cross-platform machine learning pipelines [49]. It provides a comprehensive suite of pre-built solutions for computer vision tasks such as face detection, hand tracking, and pose

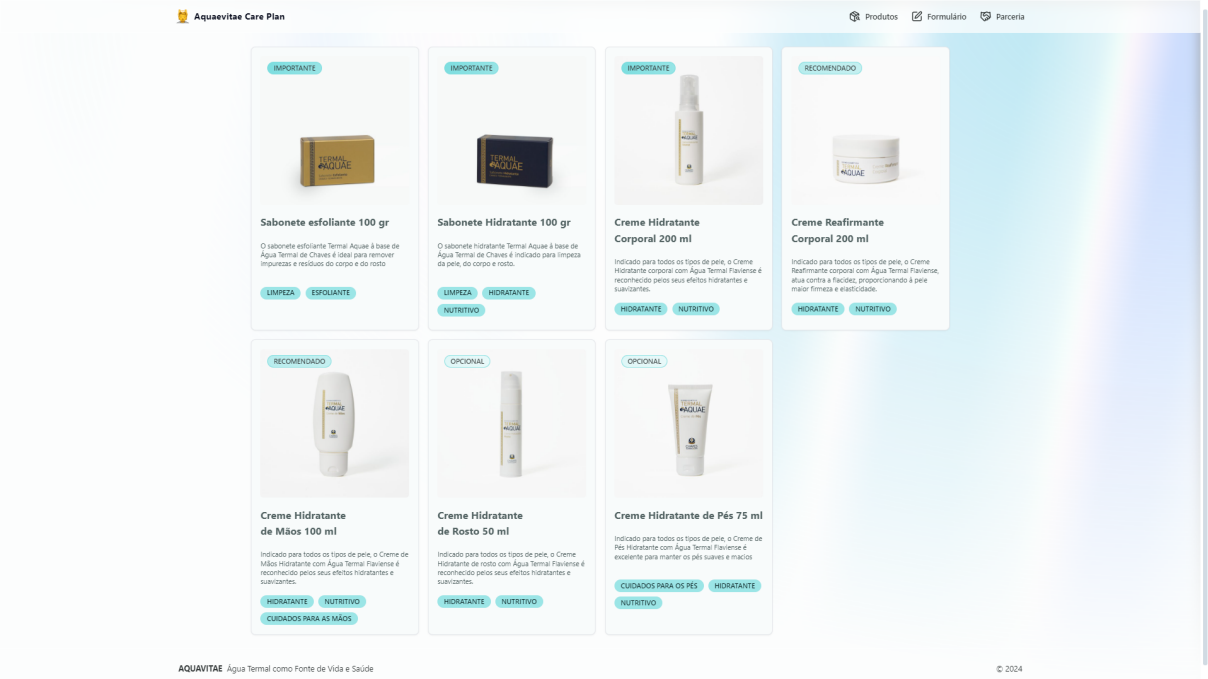


Figure 4.18: Recommended products list.

estimation. For this project, the tool identified the facial landmarks as shown in Figure 3.8c, to be later used as base points to crop the forehead.

For the model CNN model development Keras and Tensorflow framework were used. Keras [71] is a high-level neural networks API, written in Python, capable of running on top of deep learning frameworks like TensorFlow. It provides an easy way to access pre-trained models, such as VGG16, and apply training strategies like transfer learning. TensorFlow [72], on the other hand, is an open-source machine learning framework developed by Google that provides a comprehensive ecosystem for building, training, and deploying machine learning models, needed to treat the data in a Keras-compatible way.

In addition, OpenCV, an open-source computer vision and machine learning software library that provides a wide array of tools for image and video processing, was used to manipulate (e.g. balance the brightness among the pictures) the images during the training process.

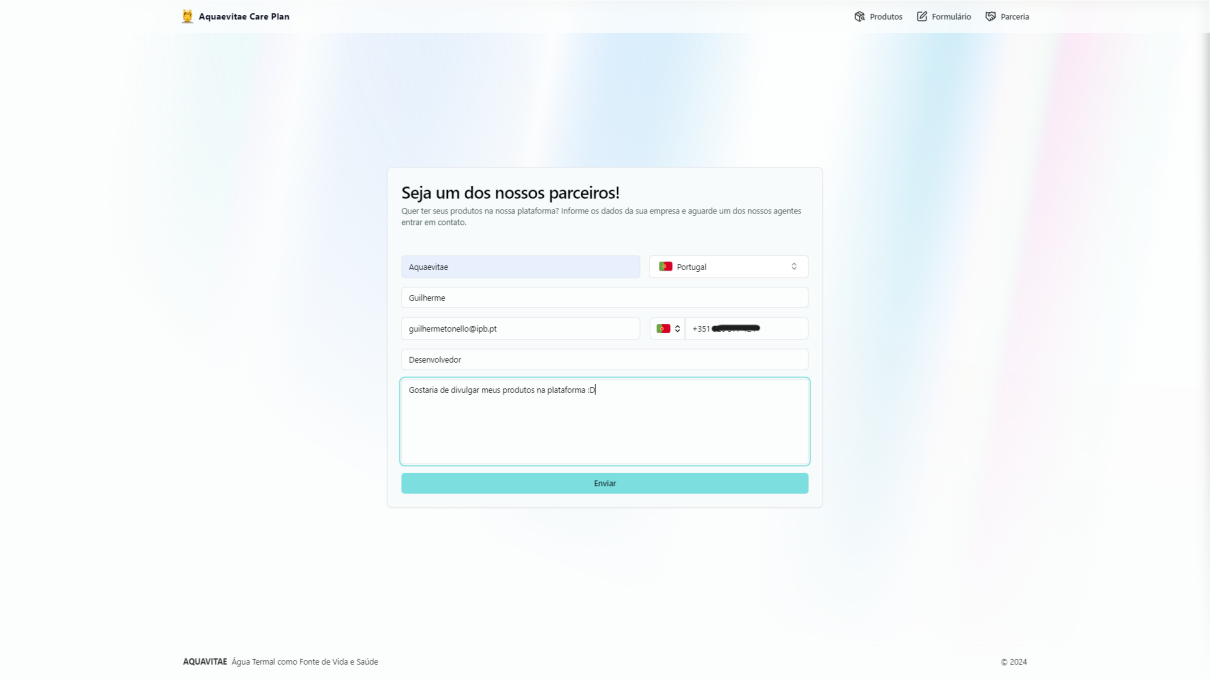


Figure 4.19: Partnership Request form.

4.3.2 Multi-Task Cascaded Convolutional Neural Network

The MTCNN is a sophisticated deep learning model designed to detect faces, identify bounding boxes, and pinpoint five key facial landmarks within an image [27]. This model operates in three distinct stages, each enhancing the accuracy and detail of the facial recognition process. Kaipeng Zhang's development of the MTCNN framework revolutionized with a method of forward cascade detection founded by integrating three phases that effectively detect bounding boxes, facial points, and landmarks.

As an image progresses through each MTCNN phase, the precision of the detection significantly improves. Initially, the image is input into the Convolutional Neural Network (CNN), which assigns a confidence score and delineates a bounding box. In the first stage, the image is scaled down, and the CNN constructs an image pyramid to facilitate multi-scale detection. During the subsequent stages, the model extracts image patches corresponding to each bounding box, resizing them progressively. By the third stage, the model not only refines the bounding boxes and updates their scores but also calculates the

five essential facial landmarks for each detected face, ensuring comprehensive facial feature detection and analysis. For this work, the MTCNN model is being used to accurately identify and crop the facial bounding box from the dataset images.

4.3.3 VGG16

VGG16 is a CNN developed by the Visual Geometry Group at the University of Oxford for the ImageNet Challenge 2014. The number "16" in VGG16 refers to the fact that it has 16 weight layers, including 13 convolutional layers and three fully connected (or dense) layers. There are other versions of VGG, including VGG13 and VGG19, which have 13 and 19 weight layers, respectively.

The VGG16 architecture focuses on the high depth of the algorithm, with the 16 weight layers, a greater number compared to other algorithms until then. Furthermore, it stands out for its use of 3x3 filters, the smallest possible for the perception of left/right, bottom/top, and center, and a 2x2 kernel, as a parameter for the MaxPooling function, both with a stride of 1 pixel.

Exploring three chained convolutional layers with a 3x3 filter allows us to obtain the same perception as a single layer with a 7x7 filter, present in other models. This layer decomposition increases the specificity of the algorithm. In addition to reducing the necessary computational effort, it considerably reduces the number of weights [50]. Its efficiency was proven by winning second place in the classification tasks at the ImageNet Challenge. Since then, it has been widely used as a basis for this type of activity. Compared to other more current algorithms, its reduced filter size and relatively simple architecture have already proved as a great fit for small dataset problems [73], being an interesting option for this study.

4.3.4 Training Settings

Training a model stands out as a costly and time-consuming step. Extended hours are needed for the network to learn to identify patterns in a database. This subsection lists

the configurations used to train the developed network:

- **Optimizer:** The choice of optimizer impacts the model's learning flow. The SGD (Stochastic Gradient Descent) optimizer showed better results than the commonly used ADAM. Its superior performance has already been proven when fine-tuning other types of models, as shown in the study [74]. In this case, using the ADAM optimizer results in an almost constant loss graph, preventing the network from learning efficiently. Given the high number of parameters in the VGG16 [50] network used as a base, it is assumed that this occurs.

- **Learning Rate:** The learning rate works with the optimizer to determine the variation of weights during training. Very high values can make the model's learning process inconsistent, making it difficult to achieve the best result. Low values can delay learning, preventing the network from reaching its maximum capacity within the available training time. A range of values between 10^{-1} and 10^{-5} was tested in model training, while the best result was obtained with 10^{-3}.

- **Number of epochs**: 100 training epochs was needed to obtain the best result. Higher values resulted in constant overfitting, reducing the model's accuracy in new situations.

- **Validation and training split**: 70% of the images extracted from the Color FERET database were used in the model training, and the remaining 30% were used for validation. After, the model was tested with another set of images taken from the internet.

4.3.5 Discussion about the training process

Figure 4.20 shows the model loss and accuracy graphs over each epoch. The inconsistency of validation values is notorious, given the low amount of data available and the relatively high learning rate. However, the training line, despite being chaotic, maintains greater consistency. Its jumps in values can be justified by the low amount of data, high learning

rate, and the Dropout layer, which, due to its random execution, can generate this type of distortion. The training time for the final model was ≈ 33.2 seconds, with an NVIDIA RTX3060 video card, on the Ubuntu 22.04 operating system.

Lower learning rate values resulted in a constant but linear graph, which prevented the model from reaching its highest capacity in the number of epochs used. The increase in the number of epochs, in turn, resulted in over-fitting.

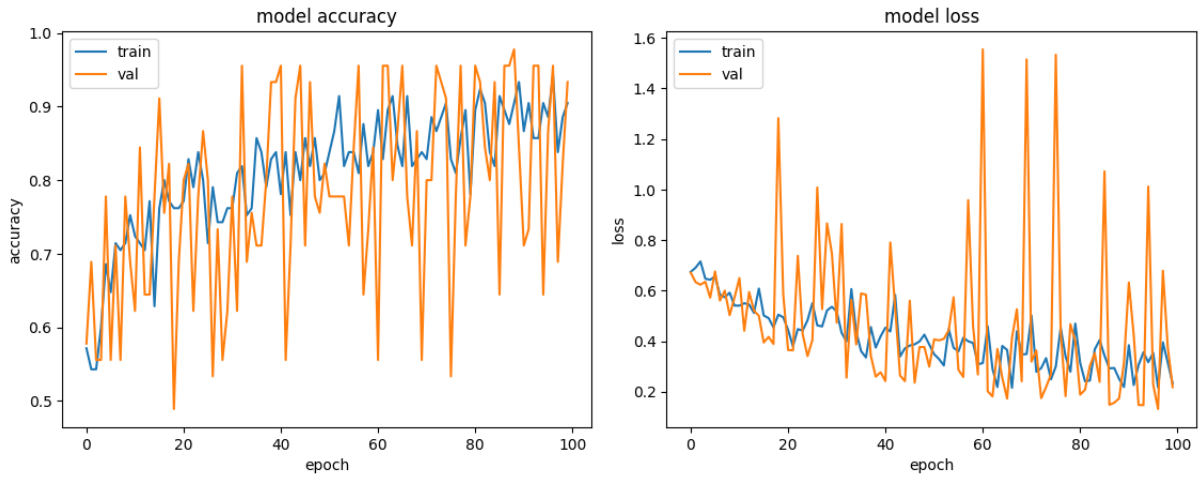


Figure 4.20: Accuracy and loss graphs about a single training epoch. The behavior represented is repeated for the other epochs.

4.3.6 Wrinkles Detection Results

Accuracy and F1-score metrics were used to evaluate the experimental results. These two metrics are calculated based on the following numbers:

- True Positives (TP): numbers of images with Wrinkles in which Wrinkles was detected;
- False Positives (FP): number of images with Wrinkles in which Wrinkles was not detected;

- False Negatives (FN): numbers of images without Wrinkles in which Wrinkles was detected;
- True Negative (TN): numbers of images without Wrinkles in which Wrinkles was not detected;

Accuracy is the proportion of correct predictions relative to the total number of predictions. The Accuracy computation as in Equation 4.2.

$$accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{4.1}$$

F1-score is a precision measure that considers both the model's precision (the fraction of relevant instances among the retrieved instances) and recall (the fraction of the total relevant instances that were actually retrieved). The precision, recall and F1-Score are computed with the Equations 4.2, 4.3, 4.4.

$$precision = \frac{NTP}{NTP + NFP} \tag{4.2}$$

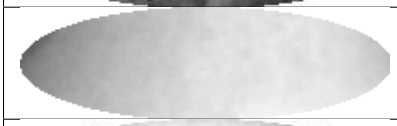
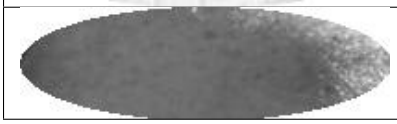
$$recall = \frac{NTP}{NTP + NFN} \tag{4.3}$$

$$f1 - Score = 2 * \frac{precision * recall}{precision + recall} \tag{4.4}$$

The trained model achieved an accuracy of 92% and an f1-score value of 92%. This is an adequate result considering the limitations in the size of the experiment dataset. Table 4.1 presents examples of the model's prediction results for four input images. The

confusion matrix of the results is presented in Figure 4.21 – it is possible to visualize that the model also had only two errors in each class, showing a good distribution across the cutoff point. The final model is extracted as a "keras" file and uploaded in a specific folder inside the API, so it can be used in the web system, therefore any new developed model can be easily changed or added to.

Table 4.1: Model predictions example

Input Image	Wrinkle Percentage	No Wrinkle Percentage
	93.52%	06.48%
	03.18%	96.82%
	80.93%	19.07%
	15.53%	84.47%

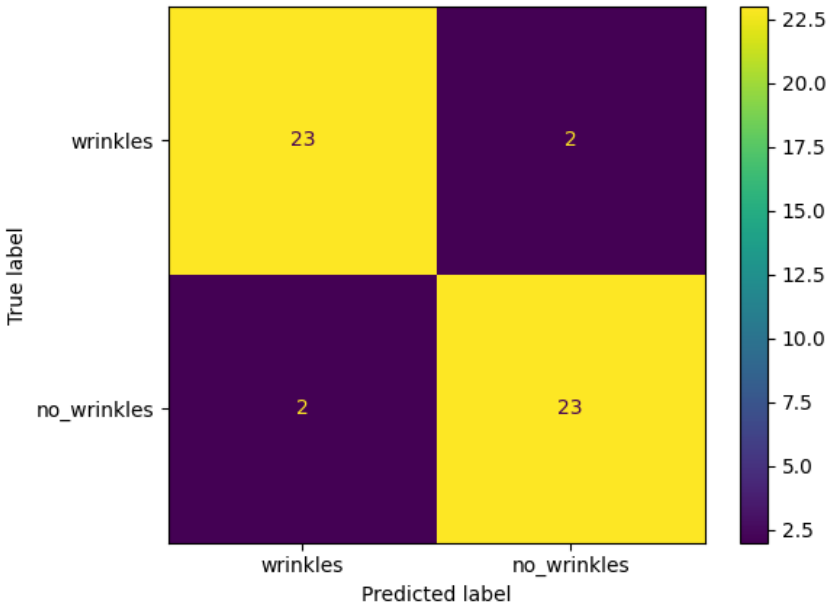


Figure 4.21: Confusion matrix

4.4 Conclusion

This chapter presented the development process of the back-end, front-end, and the machine learning model respectively. Each one has described its set of needed tools to implement the system following the proposed methodology.

Django was used as the main framework during the back-end end development, together with the Celery integration to process the submitted facial images asynchronously. Some of the most relevant part for understanding the API REST implementation and behavior, such as the models and serializers, was presented. The way the product's score is calculated, the user's facial images are analyzed and the recommendation product list is generated was also discussed in this section.

As for the front end, React as chose to create the interface using the Bun runtime, to keep high performance and usability. The developed components, pages, and implementation of system features were described. Finally, the training and evaluation process of the developed CNN model is approached. Specifying the tools, training parameters, evaluation metrics, and results.

Chapter 5

Conclusion

Inside the scope of the Aquaevitae project and following the trends in skincare and natural products, this work presents the creation of a web platform for recommending thermal water-based cosmetic products given the specific conditions of each user. The platform makes use of a developed wrinkle recognition machine learning model and other neural network tools, such as DeepFace for age detection, to assist in the process of filling out information and improve the user's experience. Information such as the user's skin type and skin diseases, together with their level, is sent to the server and crosses with the product's database, each product receives a score based on its compatibility that will be later used to generate a personalized list of recommended products, based on the user's needs.

The platform was developed in conjunction with an administrator panel, giving project managers the ability to manage the data in the system, such as adding and removing products, without any prior computer knowledge. In addition, the system offers a partnership flow where interested institutions can request a partnership to have their products promoted on the platform, which will be evaluated by an administrator in the panel. The lists of recommended products generated become more complete and accurate as the catalog of available products grows.

The developed neural network approach uses machine learning models to analyze a

user's submitted facial picture, then prefill the form information, and improve the recommended list quality. The provided pictures can be stored, given the user's permission, as well as the predicted results so the developed model can be retrained and improved in the future with a more accurate, complete, and bigger dataset.

Furthermore, this work presents an approach based on a machine learning model that recognizes wrinkles on the forehead from facial photographs, exploring the VGG16 architecture with an accuracy of 92%. The CNN was trained with just 150 images (45 used for validation) and was tested with a set of 50 completely different images. Image preprocessing, transfer learning, and fine-tuning proved essential for the model's performance due to the small number of images. The challenges documented in the state-of-art works were proven to be true, since the team has faced difficulties finding available databases to work with, resulting in the small used dataset.

Overall, the proposed solutions have proven to be effective both in the detection of expression lines and in the generation of the list of recommended products. The main issue was trying to find an adequate dataset for the machine learning model training and understand the expected features for a cosmetic recommendation system integrated with computer vision algorithms. In addition, the present work results in an article entitled "A Neural Network-based Approach to Identifying Wrinkles and Recommending Cosmetic Products" accepted into the International Conference on Optimization, Learning Algorithms and Applications (OL2A 2024) [75].

5.1 Future Works

Several potential scenarios for future work arise from this project. Initially, for the web system, more features can be developed giving the users further information about the facial analysis process and results and giving it more options to customize their recommended list after being generated. The system was developed to make easier the addition of new machine learning models capable of extracting additional data from the submitted image, improving the process quality.

With the collected data from the web system, it's possible to retrain the developed model with a larger amount of data and more up-to-date data in the future. With that, more complex CNN models can be used in the fine-tuning process. Also, different strategies of data augmentation can be tested, such as the generation of synthetic data.

Bibliography

[1] P. J. Phillips, H. Wechsler, J. Huang, and P. J. Rauss, "The feret database and evaluation procedure for face-recognition algorithms," *Image and vision computing*, vol. 16, no. 5, pp. 295–306, 1998.

[2] P. J. Phillips, H. Moon, S. A. Rizvi, and P. J. Rauss, "The feret evaluation methodology for face-recognition algorithms," *IEEE Transactions on pattern analysis and machine intelligence*, vol. 22, no. 10, pp. 1090–1104, 2000.

[3] N. Yusuf, C. Irby, S. K. Katiyar, and C. A. Elmets, "Photoprotective effects of green tea polyphenols," *Photodermatology, photoimmunology & photomedicine*, vol. 23, no. 1, pp. 48–56, 2007.

[4] H.-F. Chen, Y.-H. Lee, Y.-C. Tu, and Y.-F. Chao, *Consumer purchase intention for skincare products*, 2012.

[5] J. Chin, B. C. Jiang, I. Mufidah, S. F. Persada, and B. A. Noer, "The investigation of consumers' behavior intention in using green skincare products: A pro-environmental behavior model approach," *Sustainability*, vol. 10, no. 11, p. 3922, 2018.

[6] I. Kokoi, "Female buying behaviour related to facial skin care products," 2011.

[7] Aquavalor, *Aquae vitae – água termal como fonte de vida e saúde*, 2022. [Online]. Available: `https://aquavalor.pt/2022/03/29/aquae-vitae-agua-termal-como-fonte-de-vida-e-saude/`.

[8] Dermalogica, *Dermalogica face mapping*. [Online]. Available: https://www.dermalogica.co.uk/pages/face-mapping.

[9] Revieve, *Ai skin diagnostics*. [Online]. Available: https://www.revieve.com/platform/skincare/ai-skin-diagnostics.

[10] Garnier, *Skin coach ai*. [Online]. Available: https://www.garnier.com.au/skin-coach-skin-analysis.

[11] L. Groupe, *Skin genius*. [Online]. Available: https://www.loreal.com/fr/articles/science-et-technologie/skin-genius-loreal-paris/.

[12] L. Groupe, *Behind the scenes of the skin analyzer apps*, 2021. [Online]. Available: https://www.loreal.com/en/articles/science-and-technology/ri-behind-the-scenes-of-the-skin-analyzer-apps/.

[13] Modiface, *Modiface*. [Online]. Available: https://modiface.com.

[14] Ignae, *Ai skin analysis*. [Online]. Available: https://ignae.com/pages/ai-skin-analysis.

[15] SkinQ, *Skinq ai mirror*. [Online]. Available: https://skinq.com/pages/ai-mirror.

[16] Deeptag, *Deeptag ai*. [Online]. Available: https://deeptag.ai/DeepTagPage.html.

[17] Lululab, *Lumini software development kit*. [Online]. Available: https://www.lulu-lab.com/.

[18] C.-C. Ng, M. H. Yap, N. Costen, and B. Li, "Wrinkle detection using hessian line tracking," *IEEE Access*, vol. 3, pp. 1079–1088, 2015.

[19] A. Savran, B. Sankur, and M. T. Bilge, "Comparative evaluation of 3d vs. 2d modality for automatic detection of facial action units," *Pattern recognition*, vol. 45, no. 2, pp. 767–782, 2012.

[20] C.-C. Ng, M. H. Yap, N. Costen, and B. Li, "Automatic wrinkle detection using hybrid hessian filter," in *Asian Conference on Computer Vision*, Springer, 2015, pp. 609–622.

[21] S. Kim, H. Yoon, J. Lee, and S. Yoo, "Semi-automatic labeling and training strategy for deep learning-based facial wrinkle detection," in *IEEE International Symposium on Computer-based Medical Systems (CBMS)*, IEEE, 2022, pp. 383–388.

[22] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in *Medical Image Computing and computer-Assisted Intervention (MICCAI)*, Springer, 2015, pp. 234–241.

[23] N. Batool and R. Chellappa, "Fast detection of facial wrinkles based on gabor features using image morphology and geometric constraints," *Pattern Recognition*, vol. 48, no. 3, pp. 642–658, 2015.

[24] Z. Liu, Q. Qi, S. Wang, and G. Zhai, "A novel approach to the detection of facial wrinkles: Database, detection algorithm, and evaluation metrics," *Computers in Biology and Medicine*, p. 108 431, 2024.

[25] A. Alrabiah, M. Alduailij, and M. Crane, "Computer-based approach to detect wrinkles and suggest facial fillers," *International Journal of Advanced Computer Science and Applications*, vol. 10, no. 9, 2019.

[26] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 2818–2826.

[27] K. Zhang, Z. Zhang, Z. Li, and Y. Qiao, "Joint face detection and alignment using multitask cascaded convolutional networks," *IEEE Signal Processing Letters*, vol. 23, no. 10, pp. 1499–1503, 2016.

[28] A. Cockburn, *Writing Effective Use Cases*, 1st. USA: Addison-Wesley Longman Publishing Co., Inc., 2000, ISBN: 0201702258.

[29] J. Rumbaugh, I. Jacobson, and G. Booch, *Unified Modeling Language Reference Manual, The (2nd Edition)*. Pearson Higher Education, 2004, ISBN: 0321245628.

[30] R. Elmasri and S. Navathe, "Fundamentals of database systems," *World*, 2000.

[31] P. J. Leach, R. Salz, and M. H. Mealling, *A Universally Unique IDentifier (UUID) URN Namespace*, RFC 4122, Jul. 2005. DOI: 10.17487/RFC4122. [Online]. Available: https://www.rfc-editor.org/info/rfc4122.

[32] D. Garlan and M. Shaw, "An introduction to software architecture," in *Advances in software engineering and knowledge engineering*, World Scientific, 1993, pp. 1–39.

[33] A. Berson, *Client/server architecture*. McGraw-Hill, Inc., 1996.

[34] M. Glinz, "On non-functional requirements," in *15th IEEE International Requirements Engineering Conference (RE 2007)*, 2007, pp. 21–26. DOI: 10.1109/RE.2007.45.

[35] M. Mikowski and J. Powell, *Single page web applications: JavaScript end-to-end*. Simon and Schuster, 2013.

[36] IBM, *What is an api?* [Online]. Available: https://www.ibm.com/topics/api.

[37] L. Gupta, *What is rest?* [Online]. Available: https://restfulapi.net/rest/.

[38] D. Box, D. Ehnebuske, G. Kakivaya, *et al.*, *Simple object access protocol (soap) 1.1*, 2000.

[39] T. G. Foundation, *A query language for your api*. [Online]. Available: https://graphql.org.

[40] Google, *A high performance, open source universal rpc framework*. [Online]. Available: https://grpc.io.

[41] E. F. Codd, "Relational database: A practical foundation for productivity," in *ACM Turing award lectures*, 2007, p. 1981.

[42] S. Betal, *Task queues and why do we need them.* [Online]. Available: https://dev.to/sarbikbetal/task-queues-and-why-do-we-need-them-26mj.

[43] Celey, *What's a task queue?* [Online]. Available: `https://docs.celeryq.dev/en/main/getting-started/introduction.html#id2`.

[44] B. D. Ripley, *Pattern recognition and neural networks*. Cambridge university press, 2007.

[45] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[46] J. Gu, Z. Wang, J. Kuen, *et al.*, "Recent advances in convolutional neural networks," *Pattern Recognition*, vol. 77, pp. 354–377, 2018.

[47] G. Vrbančič and V. Podgorelec, "Transfer learning with adaptive fine-tuning," *IEEE Access*, vol. 8, pp. 196 197–196 211, 2020.

[48] E. Cetinic, T. Lipic, and S. Grgic, "Fine-tuning convolutional neural networks for fine art classification," *Expert Systems with Applications*, vol. 114, pp. 107–118, 2018.

[49] C. Lugaresi, J. Tang, H. Nash, *et al.*, "Mediapipe: A framework for perceiving and processing reality," in *Workshop on Computer Vision for AR/VR at IEEE Computer Vision and Pattern Recognition (CVPR)*, vol. 2019, 2019.

[50] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[51] TensorFlow, *An end-to-end platform for machine learning*. [Online]. Available: `https://www.tensorflow.org/`.

[52] G. Van Rossum and F. L. Drake Jr, *Python reference manual*. Centrum voor Wiskunde en Informatica Amsterdam, 1995.

[53] Ł. Langa and contributors to Black, *Black: The uncompromising Python code formatter*. [Online]. Available: `https://github.com/psf/black`.

[54] S. W. Adrian Holovaty, *Django 4.2*. [Online]. Available: `https://www.djangoproject.com`.

[55] T. Christie, *Django rest framework*. [Online]. Available: https://www.django-rest-framework.org.

[56] P. G. D. Group, *Postgresql: The world's most advanced open source relational database*. [Online]. Available: https://www.postgresql.org.

[57] *Celery - distributed task queue*. [Online]. Available: https://docs.celeryq.dev/en/stable/index.html.

[58] J. L. Carlson. "Redis in action." (2013), [Online]. Available: https://www.amazon.de/Redis-Action-Josiah-L-Carlson/dp/1617290858/.

[59] W. Reese, "Nginx: The high-performance web server and reverse proxy," *Linux J.*, vol. 2008, no. 173, Sep. 2008, ISSN: 1075-3583.

[60] S. I. Serengil and A. Ozpinar, "Hyperextended lightface: A facial attribute analysis framework," in *International Conference on Engineering and Emerging Technologies (ICEET)*, IEEE, 2021, pp. 1–4. DOI: 10.1109/ICEET53442.2021.9659697. [Online]. Available: https://ieeexplore.ieee.org/document/9659697.

[61] *Javascript documentation*. [Online]. Available: https://developer.mozilla.org/pt-BR/docs/Web/JavaScript.

[62] *Typescript*. [Online]. Available: https://www.typescriptlang.org/?.

[63] *Bun is a javascript runtime.* [Online]. Available: https://bun.sh.

[64] *React: The library for web and native user interfaces.* [Online]. Available: https://react.dev.

[65] *Rapidly build modern websites without ever leaving your html.* [Online]. Available: https://tailwindcss.com.

[66] *Beautifully designed components that you can copy and paste into your apps.* [Online]. Available: https://ui.shadcn.com.

[67] *Powerful asynchronous state management for ts/js, react, solid, vue, svelte and angular.* [Online]. Available: https://tanstack.com/query/latest.

[68] *Modern and scalable routing for react applications.* [Online]. Available: `https://tanstack.com/router/latest`.

[69] *Typescript-first schema validation with static type inference.* [Online]. Available: `https://zod.dev`.

[70] *Promise based http client for the browser and node.js.* [Online]. Available: `https://axios-http.com/docs/intro`.

[71] F. Chollet *et al.*, *Keras*, `https://keras.io`, 2015.

[72] Martín Abadi, Ashish Agarwal, Paul Barham, *et al.*, *TensorFlow: Large-scale machine learning on heterogeneous systems*, Software available from tensorflow.org, 2015. [Online]. Available: `https://www.tensorflow.org/`.

[73] M. Shu, "Deep learning for image classification on very small datasets using transfer learning," 2019.

[74] R. Poojary and A. Pai, "Comparative study of model optimization techniques in fine-tuned cnn models," in *International Conference on Electrical and Computing Technologies and Applications (ICECTA)*, 2019, pp. 1–4. DOI: `10.1109/ICECTA48151.2019.8959681`.

[75] *International conference on optimization, learning algorithms and applications (ol2a 2024).* [Online]. Available: `https://ol2a.ipb.pt/ui/home`.

Appendix A

Products Categories

```
CATEGORY_CHOICES = [
    ("B", _("Body")),
    ("F", _("Facial")),
    ("H", _("Hair")),
    ("P", _("Perfume")),
    ("C", _("Childrens")),
    ("S", _("Suns")),
    ("M", _("Makeup")),
    ("O", _("Other")),
]
```

Appendix B

Products Types

```
PRODUCT_TYPE_CHOICES = [
    (1, _("Soap")),
    (2, _("Syndet")),
    (3, _("Bath-Gel")),
    (4, _("Bath-cream")),
    (5, _("Shampoo")),
    (6, _("Oil")),
    (7, _("Lotion")),
    (8, _("Cream")),
    (9, _("Balm")),
    (10, _("Waters")),
    (11, _("Micellar-Water")),
    (12, _("Water-Essences")),
    (13, _("Face-Mist")),
    (14, _("Mask")),
    (15, _("Serum")),
    (0, _("Others")),
]
```

Appendix C

Products Skin Needs

```
SKIN_NEEDS_CHOICES = [
    (1, _("Clean")),
    (2, _("Makeup Remover")),
    (3, _("Hidrate")),
    (4, _("Nourish")),
    (5, _("Soothe")),
    (6, _("Complexion Care")),
    (7, _("Repairers")),
    (8, _("Exfoliating")),
    (9, _("Aging")),
    (10, _("Acne")),
    (11, _("Marks")),
    (12, _("Irritation")),
    (13, _("Antiperspirant")),
    (14, _("Scalp")),
    (15, _("Intimate Care")),
    (16, _("Hands")),
    (17, _("Feet")),
```

```
    (0, _("Others")),
]
```

Appendix D

Products Skin Solar Needs

```
SOLAR_CARES_CHOICES = [
    (1, _("Body Protection")),
    (2, _("Hair Protection")),
    (3, _("Face Protection")),
    (4, _("Aftersun")),
    (5, _("Suntun")),
    (0, _("Others")),
]
```

Appendix E

Product Skin Types

```
@unique
class ProductSkinTypeChoices(IntegerChoices):
    DRY = 1, _("Dry")
    SENSIBLE = 2, _("Sensible")
    EXTRA_DRY = 3, _("Extra-Dry")
    COMBINED = 4, _("Combined")
    IRRITATED = 5, _("Irritated")
    NORMAL = 6, _("Normal")
    ATOPIC_TENDENCY = 7, _("Atopic-Tendency")
    FRAGILE_DAMAGED = 8, _("Fragile-Damaged")
    ALL = 0, _("All")
```

Appendix F

Form Skin Types

```
@unique
class FormSkinTypeChoices(IntegerChoices):
    DRY = 1, _("Dry")
    SENSIBLE = 2, _("Sensible")
    EXTRA_DRY = 3, _("Extra-Dry")
    COMBINED = 4, _("Combined")
    IRRITATED = 5, _("Irritated")
    NORMAL = 6, _("Normal")
    ATOPIC_TENDENCY = 7, _("Atopic-Tendency")
    FRAGILE_DAMAGED = 8, _("Fragile-Damaged")
    NOT_SURE = 0, _("Not-Sure")
```

Appendix G

Form Skin Diseases

```
@unique
class FormSkinDiseasesChoices(IntegerChoices):
    TRANSLUCENCY = 1, _("Translucency")
    EYE_AREA = 2, _("Eye Area")
    UNIFORMNESS = 3, _("Uniformness")
    REDNESS = 4, _("Redness")
    SAGGING = 5, _("Sagging")
    WRINKLES = 6, _("Wrinkles")
    ACNE = 7, _("Acne")
    HYDRATION = 8, _("Hydration")
    PIGMENTATION = 9, _("Pigmentation")
    BLACKHEAD = 10, _("Blackhead")
    PORES = 11, _("Pores")
```

Appendix H

Product's Sheet

Ficha de caraterísticas de produto

Nome do produto	**Creme Hidratante de Pés**
Categoria do produto	Corporal__x_ Facial ___ Cabelo___ Perfume___ Produto Infantil___ Solar___ Maquilhagem___ Outro_____
Apresentação do produto (Indique a quantidade de produto na embalagem)	75 mL
Tipo de produto (Consulte a tabela de definições cosméticas anexa, se considerar necessário)	Sabonete ___ Syndet ___ Gel de banho___ Creme de banho___ Champô Óleo ___ Loção ___ Creme_x__ Bálsamo___ Águas___ Águas Micelares___ Essências de água___ Bruma___ Máscara___ Sérum___ Stick___ Outro_____
Tipo de pele	Seca___ Sensível___ Muito seca___ Mista a oleosa___ Irritada___ Normal___ Normal a seca ___ Normal a mista___ Com tendência atópica ___ Fragilizada e Danificada ___ Todos os tipos de pele_x__
Necessidades (produtos rosto)	Limpar ___

	Desmaquilhar ____ Cuidados Hidratantes e nutritivos__ Cuidados Apaziguantes___ Cuidados para a tez___ Cuidados reparadores___ Cuidados esfoliantes___ Cuidados antienvelhecimento___ Cuidados anti-imperfeições/ acne___ Cuidados anti-manchas___ Cuidados anti-envelhecimento___ Cuidados anti-irritações___ Outro_____
Necessidades (produtos corpo)	Limpar ___ Cuidados Hidratantes e nutritivos_x__ Cuidados reparadores___ Cuidados esfoliantes___ Cuidados anti-transpirantes___ Cuidados anti-irritações___ Cuidados para o couro cabeludo___ Cuidados para a higiene íntima___ Cuidados para as mãos___ Cuidados para os pés_x__ Outro_____
Necessidades (produtos solares)	Proteção solar___ Cuidados Pós-solar___ Cuidados autobronzeadores___ Proteção solar do rosto e corpo____ Proteção solar cabelo ____ Outro_____
Índice de proteção (apenas aplicável aos produtos solares de rosto e corpo)	SPF30___ SPF50+___ Outro_____
Caraterísticas do produto	Indicado para todos os tipos de pele, o Creme de Pés Hidratante com Água Termal Flaviense é excelente para manter os pés suaves e macios
Uso recomendado	Aplicar de manhã e/ou à noite na pele previamente limpa e seca.
Contra-indicações	Uso externo.
Lista de ingredientes (INCI)	Ingredients: Aqua (water); glycerin; cetearyl octanoate; caprilic-capric triglyceride; glyceryl stearate; cetyl alcohol; urea; aloe

	barbadensis (aloe vera extract); lanolin; tocopheryl acetate; allantoin; triethanolamine; sodium laurylsulphate; sodium lauryl sulphosuccinate; methylparaben; potassium sorbate; propylparaben; propyl gallate; BHA; fragance (parfum); benzyl alcohol; citral; tree moss extract; geraniol; hexyl cinnamal; limonene; linalool.
Imagem