BACHELOR‘S THESIS ASSIGNMENT

I. Personal and study details

Student's name: **Ovsyannikova Veronika** Personal ID number: **507653**

Faculty / Institute: **Faculty of Electrical Engineering**

Department / Institute: **Department of Computer Science**

Study program: **Software Engineering and Technology**

II. Bachelor’s thesis details

Bachelor’s thesis title in English:

Augmented reality facial styling app

Bachelor’s thesis title in Czech:

Rozšířeně realitní aplikace pro líčení obličeje

Guidelines:

Learn about augmented reality techniques and applications for facial augmentation. Design and implement an application that will allow the creation and editing of multi-layered facial image masks, adding 3D objects, and sharing them among other users of the application. At a minimum, the mask editor should be able to: finger draw with various brushes, stamps, animated GIFs, basic transformations, layers combining. Follow the UCD methodology when designing and implementing the application. Test the final application with at least five users.

Bibliography / sources:

1] Steve Aukstakalnis. Practical Augmented Reality: A Guide to the Technologies, Applications, and Human Factors for AR and VR (Usability). Addison Wesley 2017.
2] Dieter Schmalstieg, and Tobias Hollerer. Augmented Reality: Principles and Practice (Usability), Addison Wesley 2016
3] T. Lowdermilk, User-Centered Design, O’Reilly Media, 2013

Name and workplace of bachelor’s thesis supervisor:

Ing. David Sedláček, Ph.D. Department of Computer Graphics and Interaction FEE

Name and workplace of second bachelor’s thesis supervisor or consultant:

Date of bachelor’s thesis assignment: **01.02.2024** Deadline for bachelor thesis submission: **24.05.2024**

Assignment valid until: **21.09.2025**

Ing. David Sedláček, Ph.D. Supervisor's signature	Head of department's signature	prof. Mgr. Petr Páta, Ph.D. Dean's signature

III. Assignment receipt

The student acknowledges that the bachelor's thesis is an individual work. The student must produce her thesis without the assistance of others, with the exception of provided consultations. Within the bachelor's thesis, the author must state the names of consultants and include a list of references.

.	
Date of assignment receipt	Student’s signature

Czech Technical University in Prague
Faculty of Electrical Engineering
Department of Computer Science

Bachelor Thesis
Augmented reality facial styling app

Author: Veronika Ovsyannikova
Supervisor: Ing. David Sedláček, Ph.D.

2024

Acknowledgements

I would like to thank several people whose support helped me successfully complete my bachelor's thesis.

First and foremost, I am grateful to my parents for their unwavering support and for giving me the opportunity to move to the Czech Republic to study.

My deep appreciation goes to my advisor, Ing. David Sedláček, Ph.D, for his invaluable help and inspiration throughout the development of my thesis. I would also like to thank my friends and classmates at CTU for testing the application, as the design and functionality of my work greatly benefited from their feedback.

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis. I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular that the Czech Technical University in Prague has the right to conclude a license agreement on the utilization of this thesis as school work under the provisions of Article 60(1) of the Act.

In Prague, 24.05.2024

.....
Veronika Ovsyannikova

Citation of this thesis

Ovsyannikova, Veronika. Augmented reality facial styling app. Bachelor's thesis. Czech Technical University in Prague, Faculty of Electrical Engineering, 2024

Abstrakt

Hlavním cílem této bakalařské práce je analyzovat možné řešení aplikace pro styling obličeje a implementovat takovou aplikaci pomocí tzv. user-centred design metodiky. Po výzkumu bylo rozhodnuto implementovat mobilní aplikaci v jazyce Kotlin pro chytré telefony s operačním systémem Android, s využitím ARCore pro sledování obličeje a Sceneform pro vykreslování. Aplikace prošla dvěma iteracemi testování použitelnosti a získala pozitivní zpětnou vazbu od uživatelů.

Keywords: Rozšiřená realita, AR, Filtry na obličej, Virtuální zkušební kabinka, ARCore, Sceneform, Mobilní aplikace, Android, Kotlin

Abstract

The main goal of this thesis is to analyze the possible solutions of a face styling application and implement such an application utilizing user-centered design principles. After conducting thorough research, the decision was made to implement the mobile application in Kotlin for Android smartphones, using ARCore for face tracking and Sceneform for rendering. The application underwent two usability testing iterations and received positive user feedback.

Keywords: Augmented Reality, AR, Face filters, Virtual fitting room, ARCore, Sceneform, Mobile application, Android, Kotlin

Contents

Chapter 1

Introduction and motivation

Augmented Reality (AR) [1, pp. 91-92] is a technology that synthesizes the real-world environment with digital elements, predominantly featuring more real-world components than virtual ones. This definition highlights the potential of this technology, as it enriches a person's sensory perception by adding augmented information to their actual environment. The development of smartphone technologies is the main force behind the rapid expansion of AR accessibility. The ability to make displays and microelectronics smaller has made it possible to use mobile AR on smartphones and tablets, removing the necessity for specialized equipment in creating and using AR applications [2].

Augmented reality has made its mark in different spheres of everyday life, but the average person most clearly feels its impact in social media and e-commerce. Snapchat and Instagram have played significant roles in popularizing AR in everyday life through AR filters, which change a user's appearance in real-time through the camera.

AR filters have emerged as a new way of creative self-expression. They strongly appeal to the human desire to play with, idealize, and improve their looks. These filters serve not only entertainment purposes but are also used for advertising. For example, Gucci incredibly succeeded with its Virtual Shoe Try-on [3] Snapchat campaign.

The intersection of AR and the beauty industry has provided new ways to enhance customer's experience. During the COVID-19 pandemic, the beauty industry quickly adopted AR to allow users to try products virtually due to limits on in-person trials. Popular brands like Sephora, L'Oréal, Chanel, and MAC have integrated "virtual mirrors" into their websites and apps.

This thesis will cover the process of developing a facial styling application. It will use augmented reality's interactive and immersive features to give users a fun and practical tool for experimenting with their appearance.

Chapter 2

Analysis

This chapter analyzes current augmented reality (AR) technologies and their applications, focusing on face tracking and virtual try-on solutions. It begins by examining existing solutions in Section 2.1.1, detailing some of the applications in Subsection 2.1.1 and AR editors in Subsection 2.1.2. Subsequently, Section 2.2 provides an overview of technologies essential for developing a facial styling application, including a review of frameworks and libraries in Subsection 2.2.1 and an assessment of rendering technologies compatible with the ARCore Android SDK in Subsection 2.2.2.

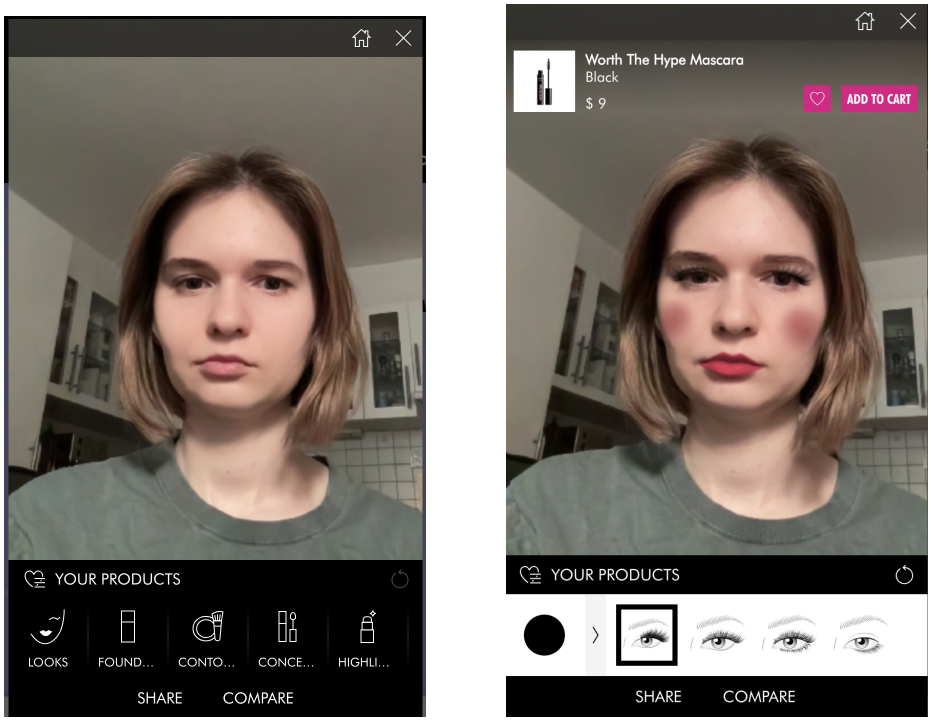
2.1 Existing solutions

2.1.1 Applications

ModiFace

ModiFace [4], a Canadian company owned by L'Oréal, specializes in augmented reality (AR) makeup, nail, hair try-ons, and skin and face analysis. They have developed an advanced face tracker algorithm that detects facial features and applies virtual makeup to the correct zones. ModiFace offers its SDK and API for commercial solutions, enabling virtual try-ons on multiple platforms, including mobile apps and web applications.

The company holds various patents. One notable patent, "System and Method for Light Field Correction of Colored Surfaces in an Image" [5], involves a computer-implemented method for correcting the rendering of makeup or skin effects. Another patent, "End-to-End Merge for Video Object Segmentation (VOS)" [6], describes a method for tracking and segmenting objects in a video sequence using a single-frame reference annotation.



(a) Before (b) After

Figure 2.1: NYX virtual try on makeup web application

Popular makeup retailers like NYX, Urban Decay, L'Oréal, and Sephora use ModiFace technologies for their virtual try-ons. These technologies allow customers to experiment with various makeup combinations, offering various colors and textures. Figure 2.1 shows a web application powered by ModiFace on the NYX website.

Warby Parker Glasses Try-On

Prescription glasses retailer Warby Parker [7] offers a web and mobile application for iOS and Android that allows users to try on glasses online. The app uses the iPhone's camera to map users' faces and provide style recommendations suited to each individual. It also utilizes the front camera to render glasses on the user's face in real-time, as shown in Figure 2.2. The iOS version of the Warby Parker application utilizes ARKit and Apple's "TrueDepth" [8] camera system, which combines sensors for accurate face measurements.

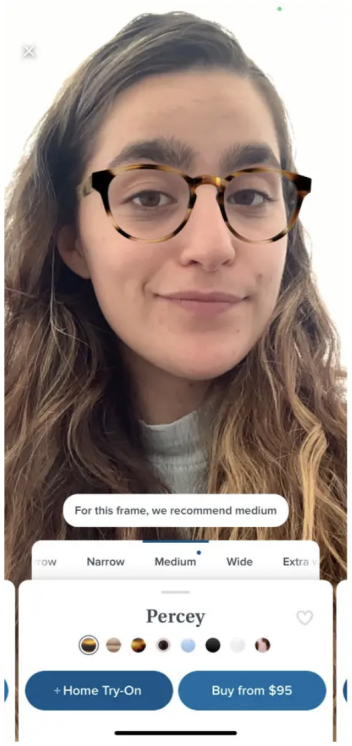
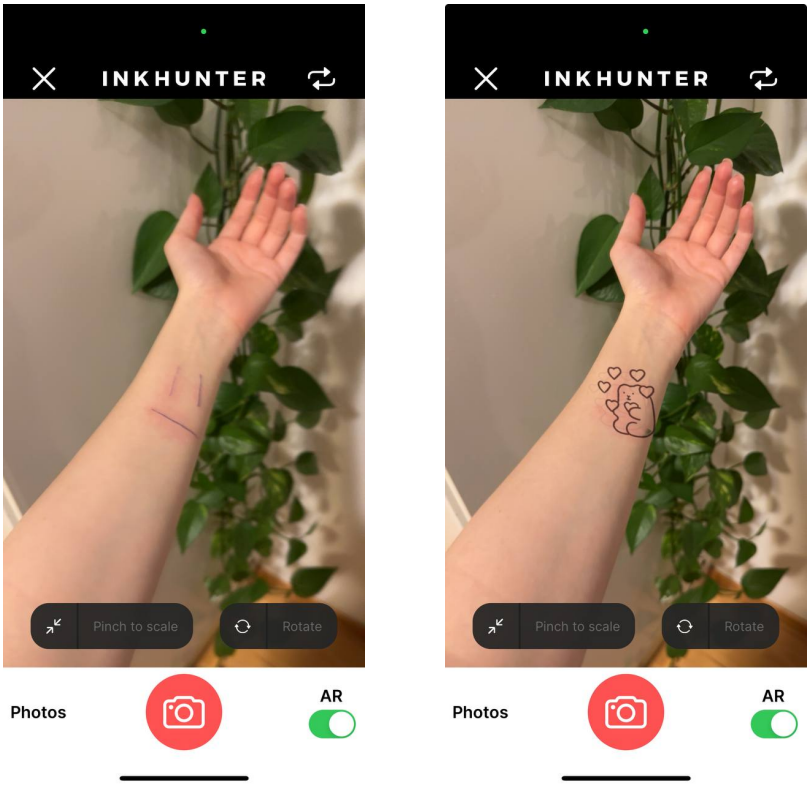


Figure 2.2: Warby Parker try-on iOS app. Source [9]

Inkhunter

Inkhunter [10] is an application for trying on tattoos. Users begin by placing a mark on the part of their body where they want to see the tattoo. The tattoo image is then attached to this mark, and its position is updated accordingly (Figure 2.3). Users can share their photos with tattoos and add new designs to the gallery for others to try out.



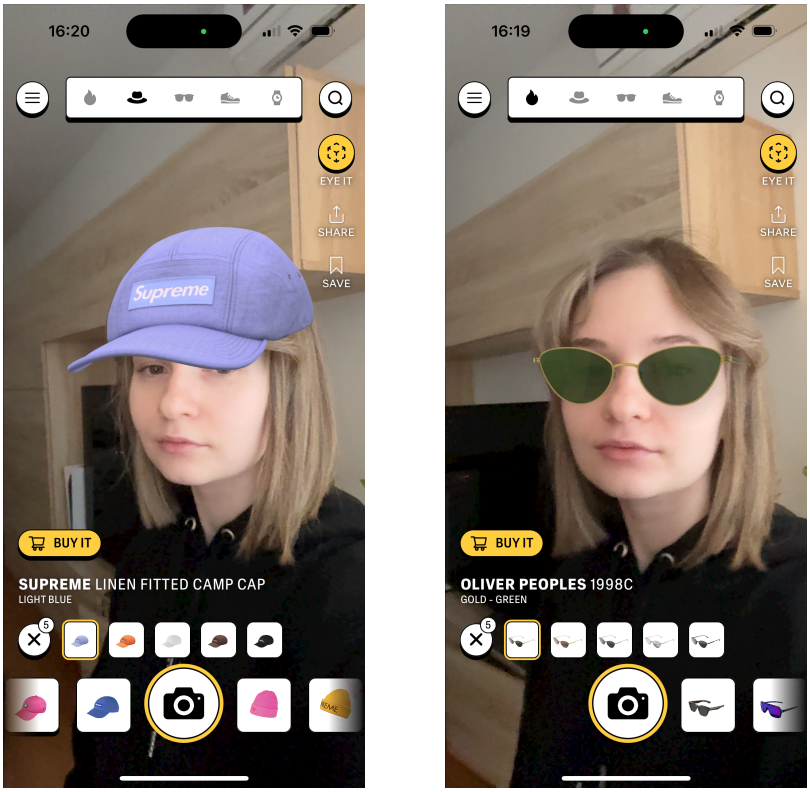
(a) Anchor mark (b) Tattoo applied

Figure 2.3: Inkhunter iOS application

TRYO

TRYO [11] is a virtual try-on application developed by QReal that is available on iOS. TRYO allows users to virtually try on sneakers, hats, watches, and glasses, providing an AR shopping experience that enables consumers to purchase with a single touch, as demonstrated in Figure 2.4.

The application also allows users to rotate and zoom in on a 3D model in a separate window. Additionally, it provides a link-sharing feature, enabling users to share items with others, who can then use the link to try them.



(a) Hat (b) Glasses

Figure 2.4: TRYO iOS application

2.1.2 Editors

Meta Spark Studio [12], previously known as Spark AR Studio, is a platform developed by Meta that enables users to create and share AR effects for Instagram, Facebook, and Messenger. Areas of application of those effects include AR face filters, advertisements, or video calls across Meta's suite of apps.

Another platform for creating AR experiences is Lens Studio [13], developed by Snap Inc. Content created with Lens Studio is sharable on Snapchat, Spectacles, and web and mobile apps through Camera Kit [14]. Camera Kit is a JDK specifically developed to create web, iOS, or Android AR applications. Filters created with Camera Kit can later be distributed to Snapchat through the Camera Kit Portal. This JDK is currently in early access.

Both editors are desktop-based and offer similar functionalities for creating face filters. They provide capabilities to track facial, hand, and body movements and facial gestures and interactions. Additionally, they can separate the camera input and change the background or color of the user's hair. They also support facial distortions, which allow changing the proportions of the face. The scripting systems in both platforms enable

the creation of interactive face filters, incorporating gaming elements for enhanced user engagement.

2.1.3 Conclusion

Although robust solutions for creating AR filters, such as Lens Studio and Meta Spark Studio, exist, these editors are only available for desktop computers. Given the popularity of mobile platforms, there is a niche for mobile-based editors that allow users to create and try on face masks, serving as simplified pocket versions of the programs mentioned above.

2.2 Technologies overview

This section provides an overview of technologies essential for developing facial styling applications.

2.2.1 Frameworks and libraries

iOS ARKit

ARKit [15], Apple's proprietary augmented reality platform, is specifically tailored for iOS devices. It leverages the advanced hardware capabilities of iPhones and iPads to provide an environment for AR development and is available for free as part of Apple's Xcode.

Face tracking with facial expression recognition uses a "TrueDepth" camera with a depth sensor. A mesh with weighted parameters representing muscle movements is attached to the user's face, allowing the application of custom 3D objects or textures to the face mesh for rendering effects. Once a face is detected, the AR session generates `ARFaceAnchor` [16] objects that provide the position and orientation of the tracked face, along with the 3D topology and parameters of the current facial expression.

Additionally, ARKit implements blend shapes [17]. These are sets of parameters that describe the movements of specific facial features. Each parameter has a corresponding value, which is a floating point number indicating the current position of that feature relative to its neutral position, ranging from 0.0 to 1.0. Figure 2.5 demonstrates the changing of the blend shape parameter corresponding to the right eye blinking.

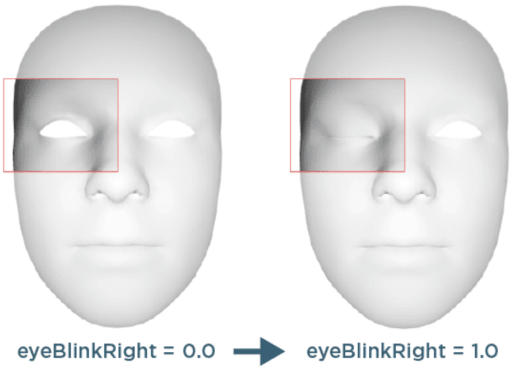


Figure 2.5: ARKit - Blend Shapes. Source [18]

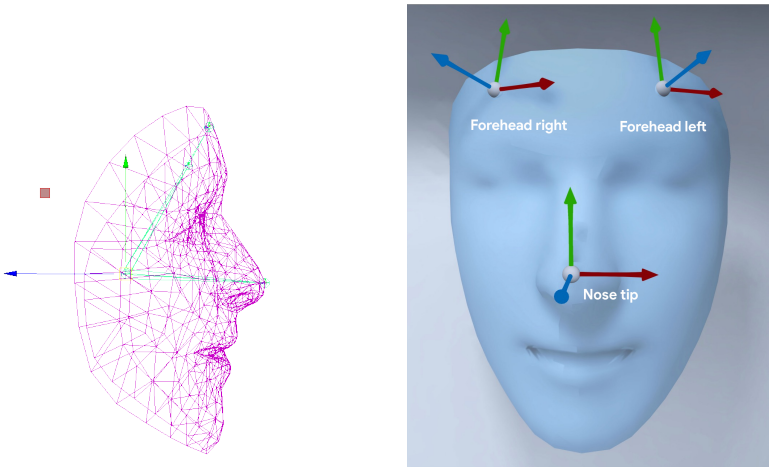
ARKit enhances face tracking capabilities by supporting concurrently tracking multiple faces and combining user face tracking with world tracking. Thus, ARKit can provide information from the rear and front cameras simultaneously, enabling features such as copying a user's facial expressions to a 3D avatar. However, face tracking with the rear camera is not supported.

ARCore

ARCore [19] is Google's platform for building AR experiences across different platforms, including Android, iOS, Web, Adobe Aero, and Unity. ARCore is open-source and free to use.

Unlike ARKit, ARCore does not require special hardware with a depth sensor. Instead, it uses frames from the camera and machine learning to detect and track a user's face, allowing applications to run even on devices that are not high-end. When a face is detected, ARCore attaches a face mesh to it. The AR session provides an `AugmentedFace` [20] object that holds information about the face mesh and key points of the tracked face, such as the center pose (Figure 2.6a) located behind the nose and marking the center of the face mesh. It also provides region poses (Figure 2.6b) that identify the nose tip and sides of the forehead. ARCore can use these poses to render assets on them, and textures can be applied to the face mesh to create effects.

Nevertheless, ARCore has some limitations: it supports tracking only one face at a time, and face tracking can be done only through the front camera. While using the front camera, ARCore does not support plane detection or anchor placement.



(a) Center pose (b) Region poses

Figure 2.6: ARCore - Augmented face. Source [21]

Unity AR foundation

AR Foundation [22] is a framework for AR development in Unity, multi-platform game engine. AR Foundation includes core features from ARKit, ARCore, Magic Leap, and HoloLens, as well as unique Unity features. AR Foundation is free but requires a Unity license for publishing. Table 2.1 provides lists of the available features in each Unity-supported provider plug-in.

Feature	ARCore	ARKit		OpenXR		XR Simulation
	Android	iOS	visionOS	HoloLens	Meta Quest	Unity Editor
Session	✓	✓		✓	✓	✓
Device tracking	✓	✓		✓	✓	✓
Camera	✓	✓			✓	✓
Plane detection	✓	✓		✓	✓	✓
Bounding Box detection					✓	
Image tracking	✓	✓				✓
Object tracking		✓				
Face tracking	✓	✓				
Body tracking		✓				
Point clouds	✓	✓				✓
Raycasts	✓	✓		✓	✓	✓
Anchors	✓	✓		✓	✓	✓
Meshing		✓		✓	✓	✓
Environment probes	✓	✓				✓
Occlusion	✓	✓				✓
Participants		✓				

Table 2.1: Supported features in AR Foundation

Unity's strength lies in its integration with ARKit and ARCore, which makes cross-platform AR applications possible [23]. It supports various plugins and extensions, such as Vuforia for image and object recognition and the XR Interaction Toolkit for immersive

AR experiences. Unity's versatility is further highlighted by its support for various input types, including touch, voice commands, and motion controllers, allowing diverse and interactive user experiences in AR.

MediaPipe

MediaPipe [24], an open-source framework from Google, is designed for building multimodal (video, audio, and time-series) machine-learning pipelines. A part of this framework, MediaPipe Solutions [25] is a suite of libraries and tools for various tasks, such as face landmark detection, which is available for Android, Web, and as a Python script. It utilizes machine learning models to process single images, videos, or live camera feeds.

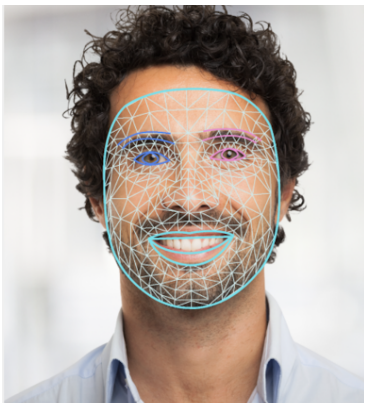


Figure 2.7: MediaPipe - face landmarks. Source [26]

The results of the face landmark detection include bounding boxes for detected faces, complete face meshes (Figure 2.7) for each detected face with blend shape scores representing facial expressions, and coordinates for facial landmarks. MediaPipe also offers solutions for gesture, hand, and pose recognition, which can be used to create AR applications.

8th Wall

8th Wall is an AR solution for mobile browsers that allows developers to create, publish, and distribute WebAR and WebVR projects within a web browser. The 8th Wall platform is a paid subscription-based service with built-in hosting, a cloud code editor, and an integrated AR engine.

8th Wall detects the user's face and attaches a face mesh. It can return various important points on the face where content can be anchored and provides the geometry of the face mesh as a whole or in parts (such as eyes or mouth). This platform can also detect and track ears, which can be helpful in try-on applications for earrings. It supports tracking and detecting up to three faces simultaneously, along with iris tracking, facial expression recognition, and face morphing to change the user's face shape.

Conclusion

After evaluating various technologies, the choice fell on Android for the mobile development platform and Kotlin as the programming language due to its familiarity. At the same time, ARCore was selected for face tracking because it provides all the features the future facial styling app will need and allows native integration with Android.

2.2.2 Rendering technologies for ARCore Android SDK

The rendering engine further processes the tracking data provided by the technologies discussed in Subsection 2.2.1. This subsection provides an overview of rendering technologies compatible with the ARCore Android SDK.

OpenGL ES

OpenGL has long been widely used in desktop environments, while mobile devices utilize its embedded variant, OpenGL ES [27, Introduction]. The introduction of OpenGL ES's first version started the era of 3D graphics on mobile platforms. It is a low-level graphics API that enables direct programming of graphics hardware using shader programs [27, pp. 8-15]. The typical setup involves initializing the GL context [27, pp. 28-36] and creating a rendering surface.

Several examples in the official Google ARCore SDK repository [28] utilize OpenGL ES with an abstraction layer created by Google developers. These examples can assist in ARCore development.

Vulkan

Vulkan [29] is a low-level graphics API that employs parallel computing principles for its computational model. In older APIs like OpenGL, drivers manage memory and synchronization and handle error management at runtime, thus requiring significant CPU resources. Vulkan shifts the responsibility for nearly all synchronization, memory

management, and state tracking to the developer. While applications developed with Vulkan may have a higher likelihood of crashing, the trade-off is greater control over the device and improved CPU performance. The primary use case of Vulkan is high-performance, real-time 3D graphics applications like video games and interactive media.

Vulkan rendering support came to ARCore in version 1.37, allowing it to update an Android hardware buffer that can be bound to Vulkan's `VkImage` [30], a storage type responsible for storing the actual texture data, including the texture's pixels and its primary memory. An example of a Vulkan application is available in the official Google ARCore SDK repository [31].

Filament

Filament [32] is a cross-platform physically based rendering engine designed to be small and efficient. It ensures smooth operation on Android systems without requiring extensive resources and uses different backends to render graphics on various platforms. For example, it can use Vulkan 1.0, OpenGL ES 3.0+, or WebGL 2.0 on Android. Filament's integration with ARCore lacks an official demo at the moment. However, an open-source example [33] demonstrates Filament's integration with plane detection and simple model manipulation.

Sceneform

Sceneform [34] was developed as a high-level scene graph API to simplify rendering with ARCore on Android, eliminating the need for prior OpenGL knowledge. It features a realistic, physically-based renderer provided by Filament. Sceneform offers an API for creating 3D models and attaching them to tracking objects, such as a user's face or an anchor in world space. ARCore provides tracking and lighting information used to render the attached objects. As a layer between ARCore and the rendering engine, Sceneform significantly lowers the barriers to entry for Android AR development, making it easier for developers to create AR experiences.

As of version 1.16.0, Sceneform was archived [35] and no longer maintained by Google. It has since been forked and maintained as an open-source project [36], but it has been a year since it has seen new releases.

SceneView

SceneView [37], created in Kotlin by the community, has emerged as a replacement for Sceneform and is currently under active development. However, it is not yet suitable for facial styling applications due to incomplete support for augmented faces, as indicated by unresolved issue [38].

Conclusion

After considering different rendering options for use with ARCore, the choice fell on the version of Sceneform maintained by the community. Although Sceneform is not actively updated, its high similarity to SceneView will make it easy to update the application later when SceneView begins to support augmented faces.

Chapter 3

Solution plan

This chapter covers the entire application designing process. It begins with defining requirements in Section 3.2, followed by use cases derived from the functional requirements in Section 3.3. It continues with a description of the application architecture in Section 3.4 and sequence diagrams in Section 3.5 and concludes with the user interface design in Section 3.6.

3.1 User-centred design

In his book "User-Centered Design" [39], Travis Lowdermilk explains that user-centered design is an iterative process involving users from beginning to end. This approach relies heavily on user feedback, both in the early stages of design and in the final phases. Making the user the centerpiece at every step ensures that the product meets users' needs.

The application will be designed using user-centered design principles. A focus group of 5-8 potential users, specifically, young people aged 15 to 30 who are active on social media will be involved in the design process. Based on their feedback, the application's requirements, use cases, and user interface will be adjusted to ensure they align with their needs and preferences.

3.2 Requirements

This section describes functional and non-functional requirements for the application. Functional requirements in Subsection 3.2.1 describe a system's specific behaviors, actions, and functions. Non-functional requirements in Subsection 3.2.2 describe system attributes such as performance, usability, and scalability.

3.2.1 Functional requirements

The focus group provided feedback on the first version of the functional requirements, specifically that the functionalities of uploading their GIFs and images would help creativity in the 2D editor. They also expressed that they wanted to be able to revisit saved drawings and make changes to them. Figure 3.1 features a list of new functional requirements based on the feedback.

1. **FRQ-12 Image Upload:** The application shall enable the user to upload images and use them in the 2D editor.
2. **FRQ-13 GIF Upload:** The application shall enable the user to upload and use GIFs in the 2D editor.
3. **FRQ-14 Saved Drawing Modification:** The application shall enable the user to open saved drawings in the 2D editor and modify them.

Figure 3.1: Added functional requirements

After further analysis and time consideration, the 3D editor, mentioned in the first version, was omitted in the final version of functional requirements, so the main focus of the application will be on the 2D editor. Figure 3.2 contains a list of functional requirements not included in the final version.

1. **FRQ-01 3D Editor:** The application shall enable the user to create an account.
2. **FRQ-01-01 3D Model Transformation:** The 3D editor shall enable the user to change 3D model position, rotation, and color.
3. **FRQ-01-02 3D Model Upload:** The 3D editor shall enable the user to upload 3D models.
4. **FRQ-01-03 3D Drawing:** The 3D editor shall enable the user to draw doodles in 3D.

Figure 3.2: Rejected functional requirements

Figures 3.3 and 3.4 show the final functional requirements.

1. **FRQ-01 Sign Up:** The application shall enable the user to create an account.
2. **FRQ-02 Log In:** The application shall enable the user to log in to the created account.
3. **FRQ-03 Sign Out:** The application shall enable the user to sign out.
4. **FRQ-04 Makeup:** The application shall enable the user to try on different makeup types and combine them.
5. **FRQ-05 Accessories:** The application shall enable the user to try on accessories and combine them.
6. **FRQ-06 Look Creation:** The application shall enable the user to create a combination of 3D models, makeup, and drawn face mask and save it.
7. **FRQ-07 Look Deletion:** The application shall enable the user to delete a look that they created.
8. **FRQ-08 Look Sharing:** The application shall enable the user to share a created combination of 3D models, makeup, and drawn face masks among users.
9. **FRQ-09 2D Editor:** The application shall enable the user to create a drawing that can be applied to the user's face as face mask.
10. **FRQ-09-01 Multi-Layered Editor:** The editor shall support layers.
11. **FRQ-09-02 Finger Drawing:** The editor shall enable finger drawing with different brush types and sizes.
12. **FRQ-09-03 Images:** The editor shall enable users to add images.
13. **FRQ-09-04 GIFs:** The editor shall enable users to add GIFs.
14. **FRQ-09-05 Element Editing:** The editor shall enable element editing such as scaling, color changing, rotating, moving, and deleting.
15. **FRQ-09-06 Drawing History:** The editor shall provide a history with undo and redo actions to manage changes.

Figure 3.3: Final functional requirements. Part 1

10. **FRQ-10 Photo:** The application shall enable the user to take a photo with applied makeup, accessories, and face mask.

11. **FRQ-11 Video:** The application shall enable the user to film a video with applied makeup, accessories, and face mask.

12. **FRQ-12 Image Upload:** The application shall enable the user to upload images and use them in the 2D editor.

13. **FRQ-13 GIF Upload:** The application shall enable the user to upload and use GIFs in the 2D editor.

14. **FRQ-14 Saved Drawing Modification:** The application shall enable the user to open saved drawings in the 2D editor and modify them.

Figure 3.4: Final functional requirements. Part 2

3.2.2 Non-functional requirements

Figure 3.5 includes a list of non-functional requirements.

1. **NFQ-01 Device Support:** The application shall be optimized for Android smartphones.
2. **NFQ-02 Operating System:** The application shall be compatible with Android version 11 and above.
3. **NFQ-03 2D Editor Performance, Layers:** The 2D editor shall support creating and managing of up to 20 layers.
4. **NFQ-04 2D Editor Performance, Elements:** The 2D editor shall support creating and managing up to 100 drawn elements (finger drawing, stamps) and up to 10 images and GIFs per project without significant freezing.
5. **NFQ-05 2D Editor Performance, History:** The 2D editor shall support holding up to 50 revertable user actions in memory.
6. **NFQ-06 Error Handling:** The application shall gracefully handle errors and provide meaningful error messages to the user.
7. **NFQ-07 User-Friendly Interface:** The application shall have an intuitive user interface.
8. **NFQ-08 Content Scalability:** The application design shall easily support adding new 3D models, makeup options, and images.

Figure 3.5: Non-functional requirements

3.3 Use cases

This section covers interactions derived from functional requirements 3.2.1 between user and the application.

After the presentation of the first version of the use cases, the focus group suggested that making the looks they created invisible to other users would be helpful. Additionally, they found sharing looks only through the in-application menu inconvenient. They wanted to share specific looks with other users via a direct link. Figure 3.6 depicts new use cases added after consultation with a focus group.

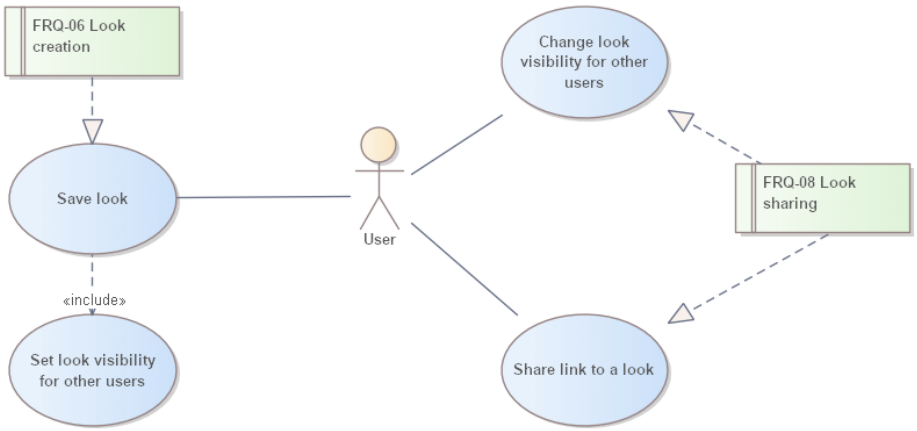


Figure 3.6: Added UC

Figures 3.7, 3.8, 3.9, 3.10, 3.11 shows the final use cases.

Figure 3.7 illustrates use cases for user registration and authentication. Figure 3.8 depicts use cases related to makeup and accessory combinations in the application. Use cases for the application's main feature, the 2D editor, are shown in Figure 3.9.

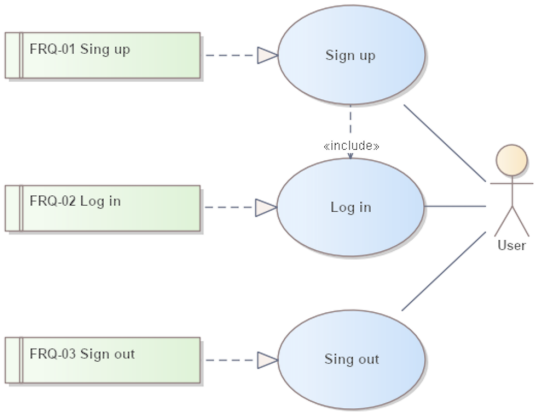


Figure 3.7: User account UC

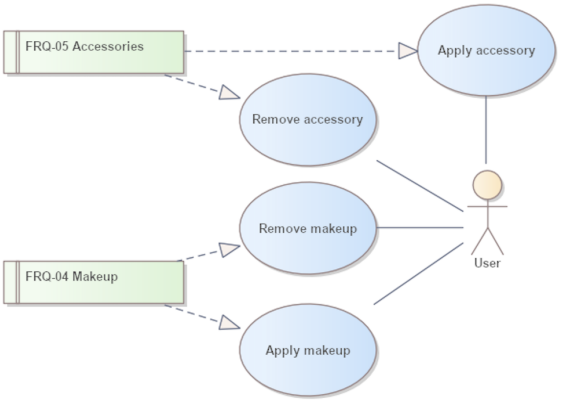


Figure 3.8: Makeup, accessories combination UC

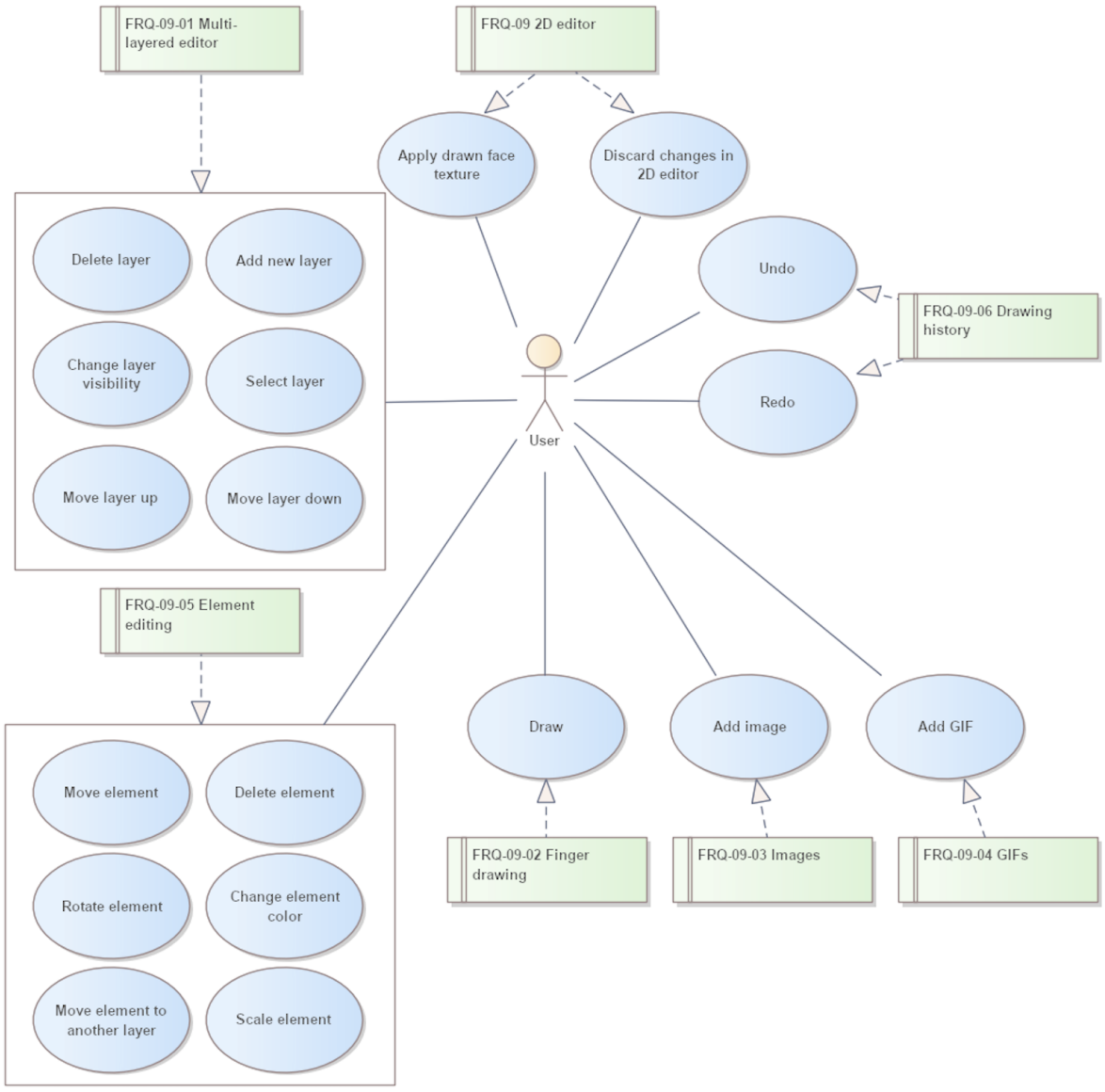


Figure 3.9: 2D editor UC

Figure 3.10 details use cases associated with creating, deleting, and sharing looks (a combination of accessories, makeup, and face mask drawn in the 2D editor).

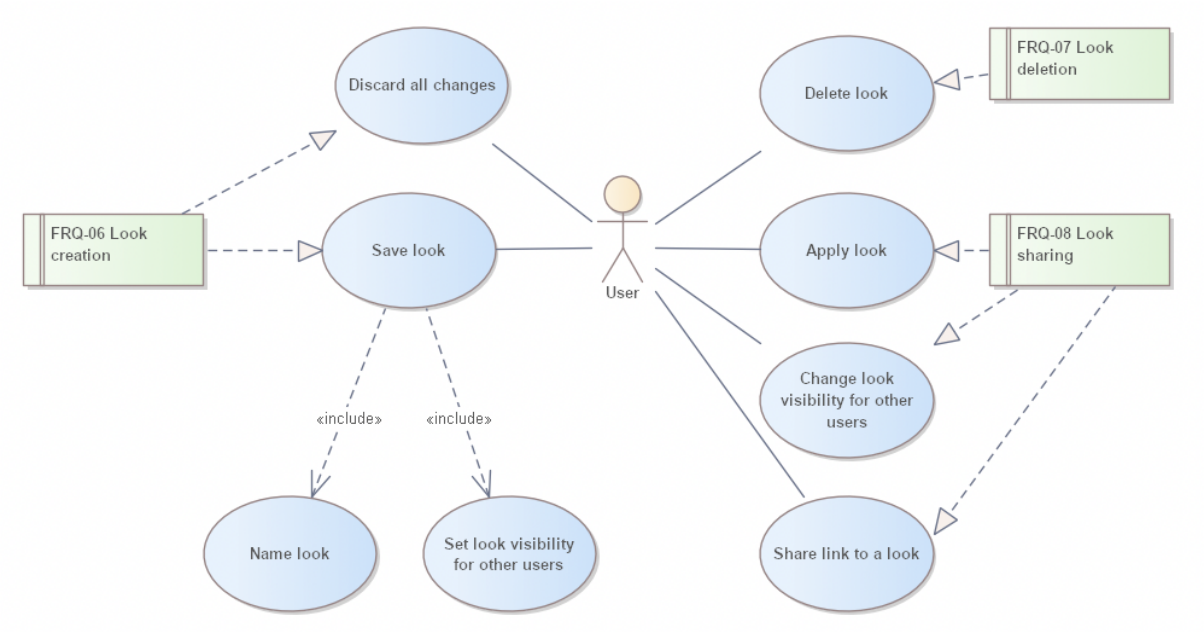


Figure 3.10: Look UC

The final diagram in Figure 3.11 presents other use cases, such as uploading images and GIFs, modifying saved drawings (face masks within look), and capturing photos and videos within the application for sharing on social media.

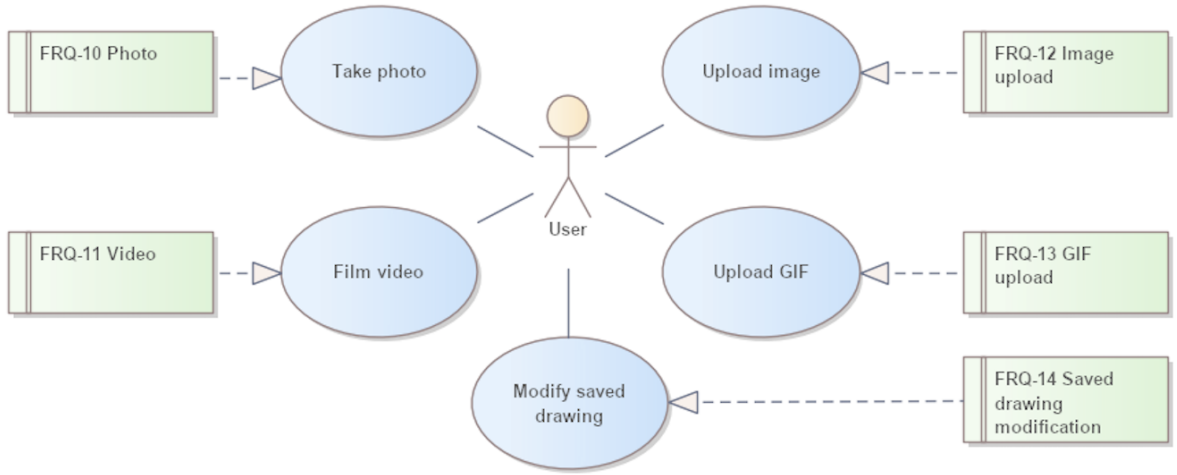


Figure 3.11: Other UC

3.4 Application architecture

This section will cover the application's architecture (Subsection 3.4.1), including the initial design considerations, the final architecture decision, and the class diagram (Subsection 3.4.2).

3.4.1 Architecture diagrams

In the initial design phase of the application, a traditional architecture, depicted in Figure 3.12, was considered, which included a client-side Android application supported by a Spring Boot [40] backend service. The backend's tasks include managing user registration, authentication, and CRUD with the help of a PostgreSQL [41] database and storing 3D models and images using Amazon S3 [42]. At the same time, an internal SQLite [43] database on the client's device would store data that are too expensive to hold in application memory and that the app should reuse.

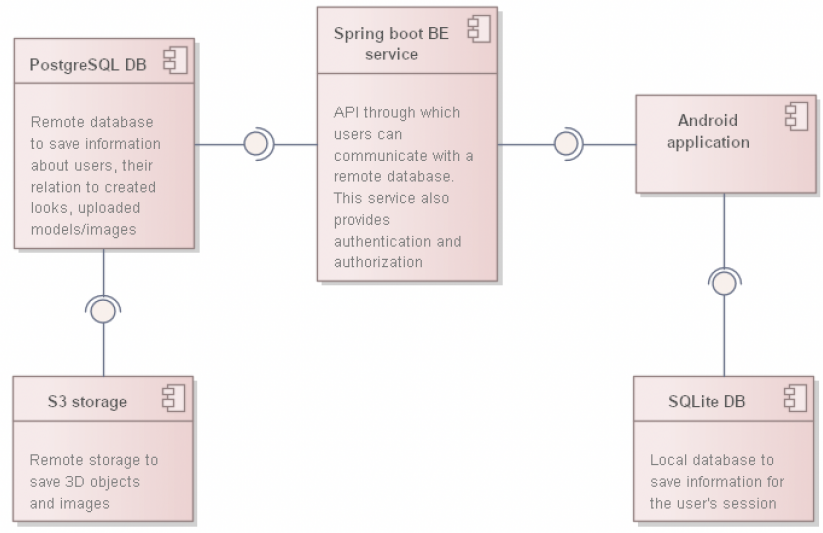


Figure 3.12: First version of the architecture diagram

However, after evaluating the project's requirements and constraints, the decision was made to use Firebase's backend cloud computing services [44] instead. Firebase provides all the necessary functionalities, including user registration, authentication, a document-based NoSQL database, and file storage. Using Firebase eliminates the need to create and host a custom backend service and database.

Additionally, Firebase Storage offers efficient management and storage solutions for images and 3D models. Firebase also provides FirebaseUI [45] for storage, which integrates with Glide [46], a popular Android library for image uploading and caching.

Furthermore, it was decided not to use an SQLite local database but to use the client device storage, as images are the only objects expected to be stored locally. Figure 3.13 shows the final version of the application architecture.

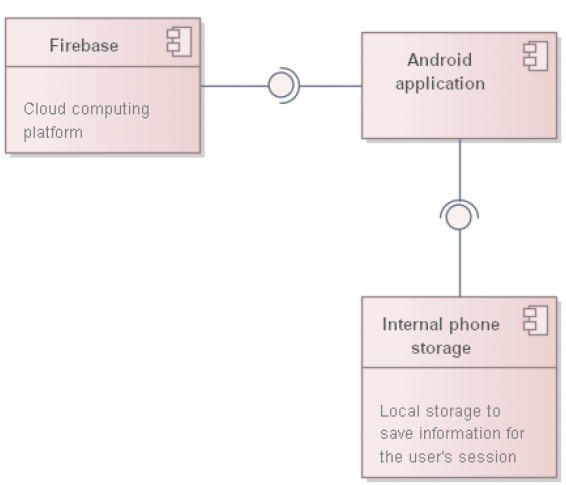


Figure 3.13: Final version of the architecture diagram

3.4.2 Class diagram

The foundation of the Android application builds upon activities [47] and fragments [48]. Activities serve as the main entry points for user interactions, representing individual screens within the application. Each activity is responsible for managing the user interface and handling user inputs.

Fragments are reusable portions of the user interface, embeddable within activities or other fragments. They allow more modular and flexible UI designs, enabling the dynamic management of different UI parts within a single activity. Fragments can also manage their life cycle and handle user interactions independently of the activity to which they are attached.

As a large amount of data cannot be passed directly between activities, the future application should aim to keep the state of applied models and textures on a user's face while allowing quick transitions between different UI components. That leads to almost single-activity architecture with a fitting room activity at the core, as depicted in Figure 3.14.

Mask Editor Fragment is designed to handle drawing and editing tasks, with references to its `DrawView` component to manage the drawing on canvas. AR Front Facing Fragment is a fragment from the Sceneform library that configures application sessions to use augmented face features.

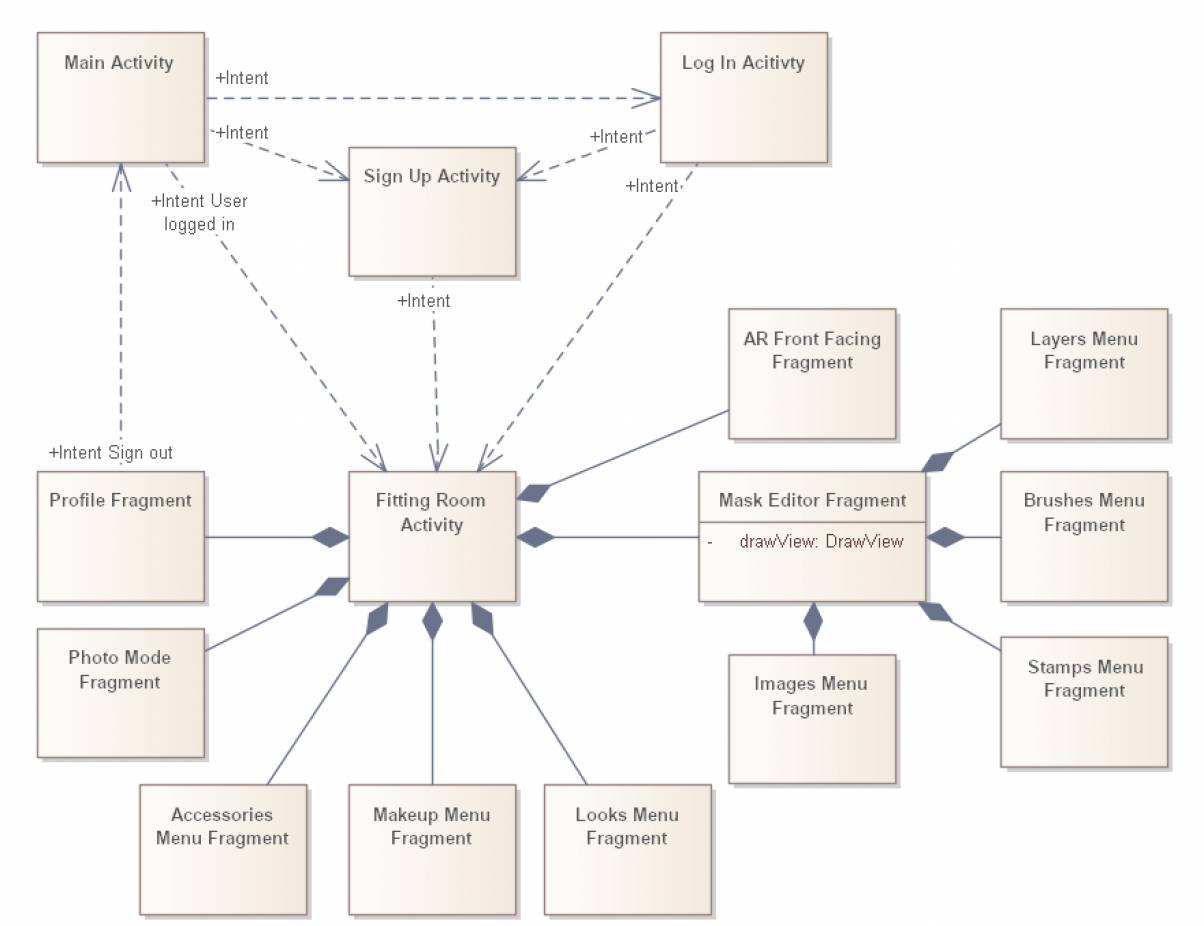


Figure 3.14: Class diagram

3.5 Sequence diagrams

The sequence diagrams in this section detail the communication between the application's parts, as defined in Section 3.4. These diagrams will focus on essential use cases, showcasing how the application's activities, fragments, and external services interact.

Figure 3.15 displays the flow for the use case "Apply an accessory" defined previously in Figure 3.8.

The diagram in Figure 3.16 shows the possible flow for use cases "Draw", "Add an image", and "Apply the drawn face mask", mentioned in Figure 3.9.

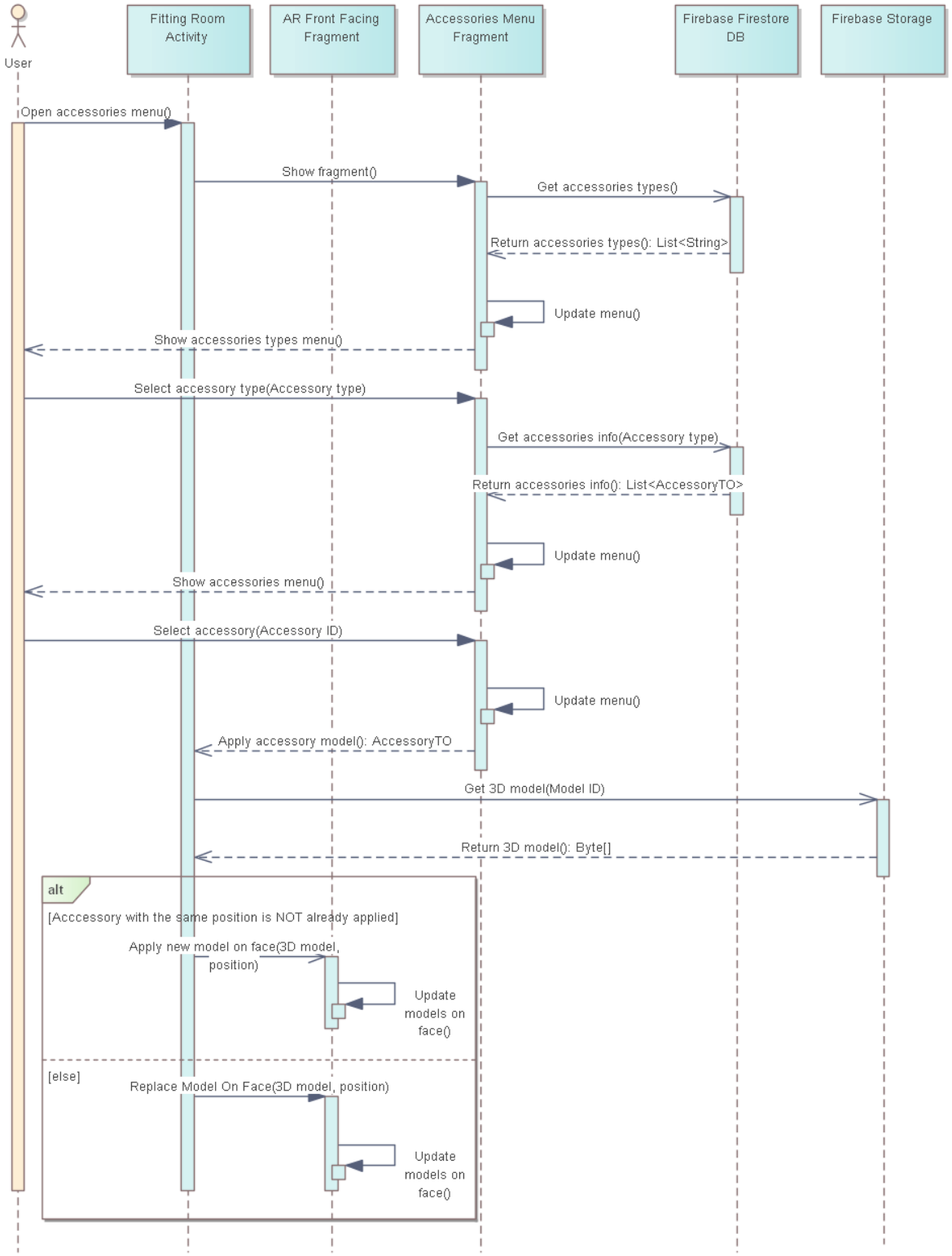


Figure 3.15: Sequence diagram - Accessories combination

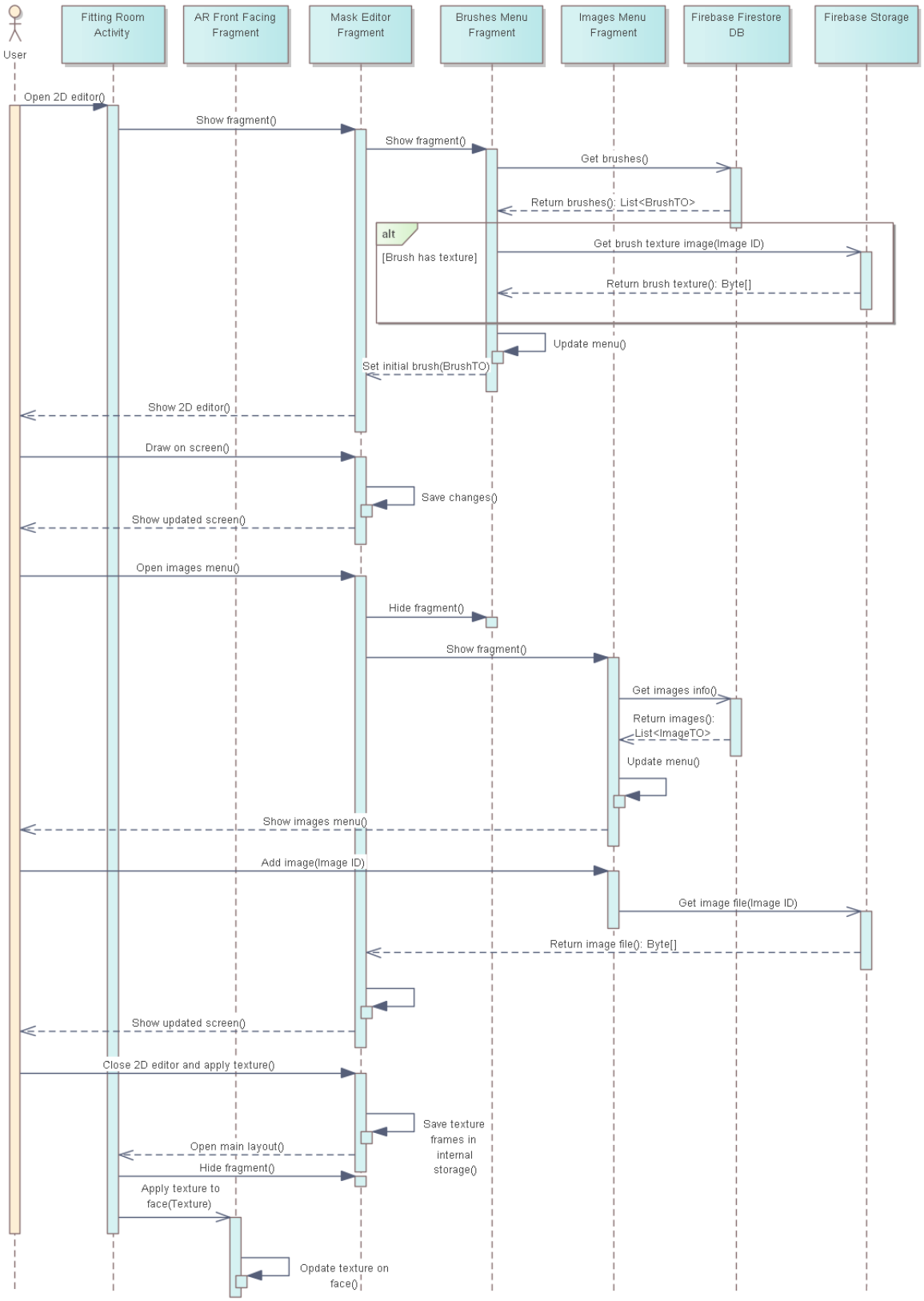


Figure 3.16: Sequence diagram - 2D editor

3.6 User interface

The initial user interface version, designed as High-Fidelity wireframes, was presented to a focus group for evaluation.

Figure 3.17: First version of the main screen

The focus group identified several issues with the initial design of the main screen, as shown in Figure 3.17. One of the concerns was the button featuring a camera icon that should open the camera mode screen where users can take pictures or film a video. Function of the button needed to be more apparent, some participants assumed that it would flip the camera from front to back. They then considered two more variants of the button.

Figure 3.18: Camera - first new variant

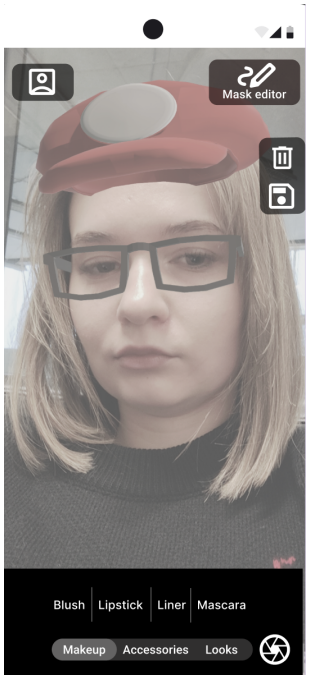


Figure 3.19: Camera - second new variant

First Variant (Figure 3.18): This design featured a camera button whose purpose users correctly identified. However, its placement stretched out the bottom menu, disrupting the layout. Second Variant (Figure 3.19): This design attempted a different layout, but it appeared even more confusing to the potential users. Given the feedback, the placement of the camera button was decided not to change. Instead, the label "Camera mode" was added to it to help clarify its purpose.

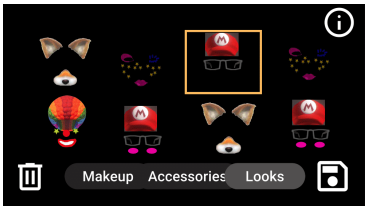


Figure 3.20: First version of the looks menu

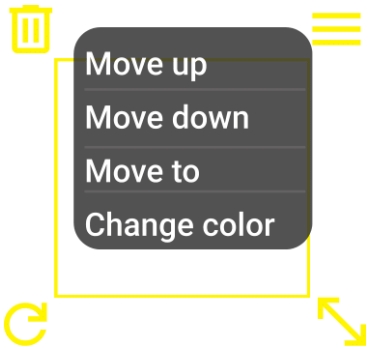
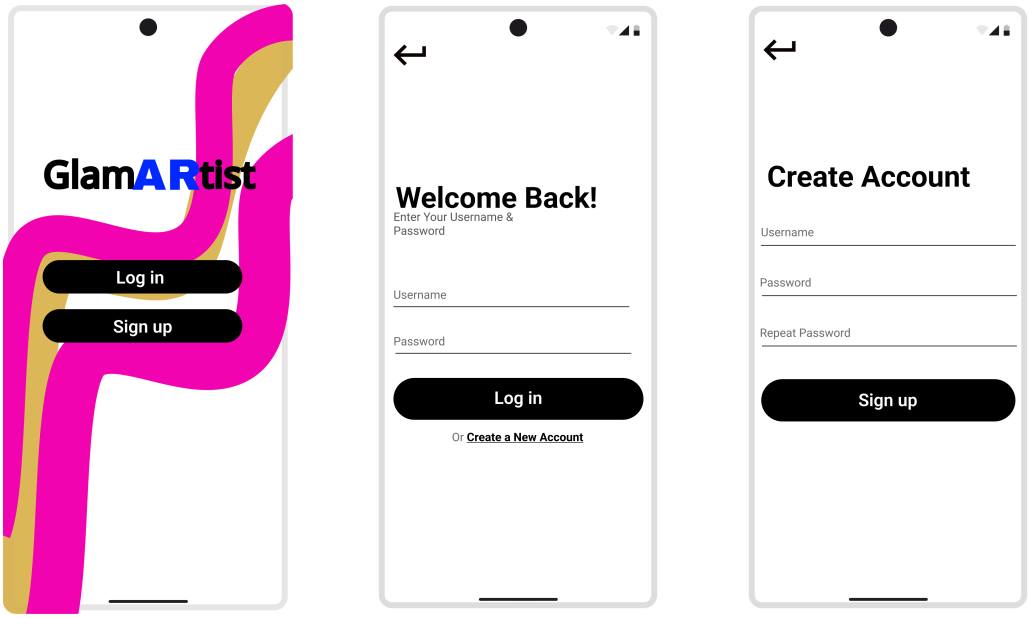


Figure 3.21: First version of the element menu

The focus group also recommended renaming certain buttons in the element menu (Figure 3.21) to reflect their functions better. Specifically, the buttons labeled "Move up", "Move down", and "Move to" were found to be ambiguous. They suggested renaming

these buttons "Layer up", "Layer down", and "To layer", respectively. This change makes it clear that these actions move elements within layers, move an element up a layer, down a layer, or move element to the specific layer. Additionally, the focus group suggested adding a filter to the looks menu (Figure 3.20). This filter would allow users to view only the looks they created, helping them find their content more easily.

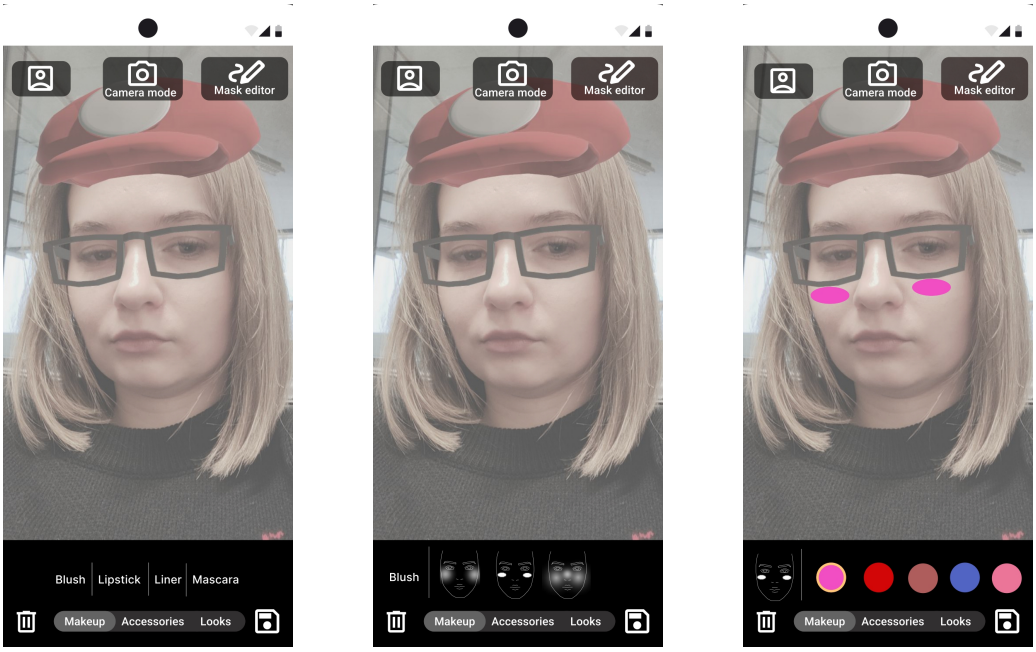
The final UI prototype is presented on Figures 3.22, 3.23, 3.24, 3.25, 3.26.



(a) Initial screen (b) Log In screen (c) Sign Up screen

Figure 3.22: UI - user account

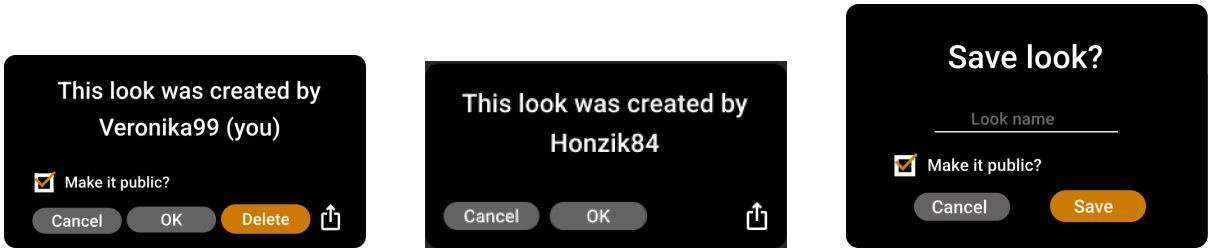
Figure 3.23 displays the three levels of the makeup menu. First, the user chooses the type of makeup. Then, the user can select from various options and apply them to their face. Finally, the user can change the color of the applied makeup.



(a) Makeup menu (b) Blush menu (c) Blush menu - colors

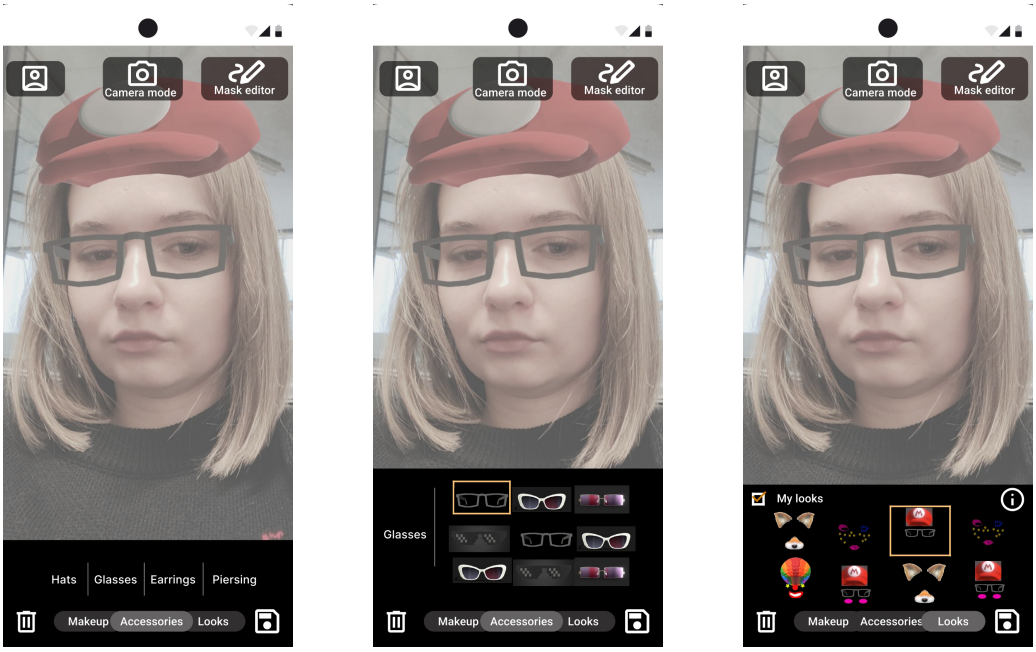
Figure 3.23: UI - makeup menu

Figure 3.25 shows two levels of the accessories menu. The first level presents accessory types, and the second displays 3D models of the selected type. Figure also includes a menu with user-created looks, featuring a filter checkbox to view only their looks, as suggested by the focus group.



(a) Look info window for the author (b) Look info window (c) Save look window

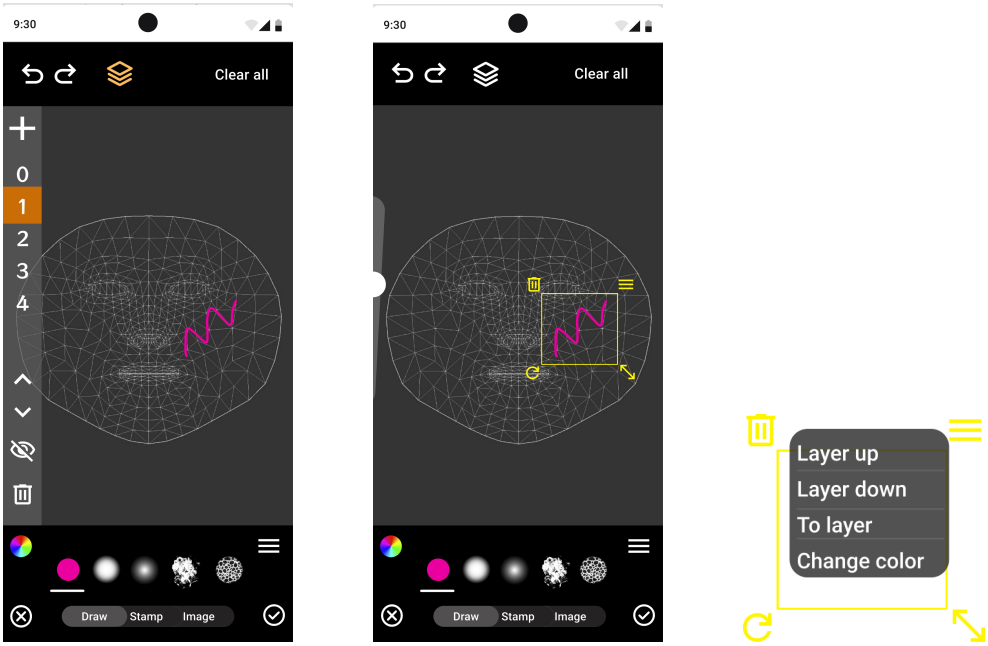
Figure 3.24: UI - look related windows



(a) Accessories menu (b) Glasses menu (c) Looks menu

Figure 3.25: UI - accessories and looks menus

Figures 3.24a and 3.24b are pop-up windows that appear after the user selects a look and presses the round "info" button on the looks menu (Figure 3.25c). After pressing the floppy disk button, the save look window (Figure 3.24c) appears.



(a) Layers menu (b) 2D editor screen (c) Element menu

Figure 3.26: UI - 2D editor

The frames in Figure 3.26 show the UI of the 2D editor. The round "OK" button saves changes and returns to the previous screen, while the "Cancel" button discards all changes and returns to the previous screen. The Layers menu (Figure 3.26a) contains buttons for managing layers: add a new layer, select a layer, move a layer up, move a layer down, hide, and delete. Figure 3.26c shows the element menu, which was improved based on feedback from the focus group.

Chapter 4

Implementation

This chapter describes the technical details of the implementation, covering key functionalities. Section 4.1 provides an overview of how makeup and accessories are combined and displayed on the user's face. In Section 4.2, the implementation of the 2D editor is detailed, including finger drawing and textured brushes in Subsection 4.2.1, and how layers work in Subsection 4.2.2. The process of saving and managing drawing history is detailed in Subsection 4.2.3, followed by an explanation of how the editor state is serialized in Subsection 4.2.4. Subsection 4.2.5 describes the format in which masks are saved, followed by Section 4.3, that explains how looks are serialized and saved. Finally, Section 4.4 provides a guide on how to add new content to the application.

4.1 Makeup and accessories combination

The `AugmentedFaceNode` [49] class, provided by Sceneform, allows rendering effects on the user's face when attached to the `AugmentedFace` [50]. To display a texture mapped to the user's face, a face mesh texture should be applied to it. 3D models that overlay the user's face should be applied as renderables.

Makeup options are rendered as texture dynamically generated by combining individual makeup bitmaps, colored based on user selections. The state of effects applied to the face is stored in the `FaceNodesInfo` data class. Accessories are applied by loading and attaching 3D models to nodes based on their defined slots.

```
data class FaceNodesInfo(
    var augmentedFace: AugmentedFace? = null,
    val slotToFaceNodeMap: MutableMap<String, AugmentedFaceNode>,
)
```

Listing 4.1: FaceNodesInfo data class

The `StateService` class manages the models and textures displayed on the user's face. It uses a slot system to organize and apply makeup and accessories. Each slot is associated with a specific type of effect, allowing for the combination or replacement of elements as needed.

4.2 2D editor

At the core of the 2D editor is the `DrawView` class. It captures user input and interprets it to perform various drawing and editing operations based on the current mode of the editor, detailed on Listings 4.2. The `Canvas` [51] is used in the `onDraw()` method to render the contents of the editor. Editor is updated on demand, using the `invalidate()` method that forces the view to be redrawn.

```
enum class EEditorMode {
    BRUSH,
    EDITING,
    PIPETTE,
    STAMP,
    ;
}
```

Listing 4.2: Editor Modes

Each element that can be drawn implements the `Drawable` interface and specifies how it should be drawn on Canvas in the `drawSpecific()` method.

4.2.1 Finger drawing

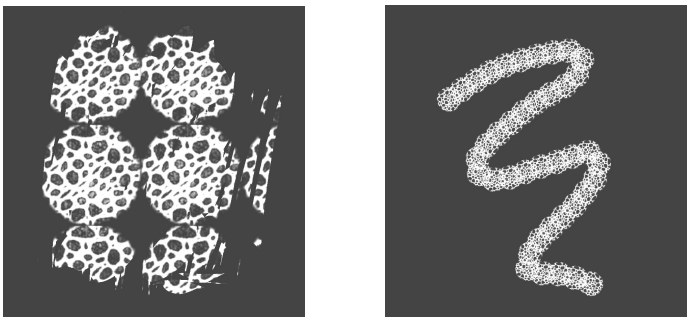
The implementation of finger drawing involves handling touch events to create curves on a canvas. The curves are drawn on `Canvas` using the `Path` [52], that allows for various drawing operations such as lines, moves, and quadratic Bezier curves. When the user

touches the screen, the coordinates are captured to define the curve's path. Android allows to recognize different motion events [53] from the user's input.

The curve creation process starts with the `ACTION_DOWN` event, where the initial touch coordinates are saved. As the user moves their finger, the `ACTION_MOVE` event is triggered, updating the path with a quadratic Bezier curve segment. This segment is defined by the previous touch point, the current touch point, and the midpoint between them. This approach ensures smooth and continuous curves. When the user lifts their finger, `ACTION_UP` finalizes the path by drawing a straight line to the last touch point.

Textured brushes

The Android `Paint` [54] class allows using `BitmapShader` [55] to draw textures. However, as seen in Figure 4.1a, the result is not suitable for simulating a textured brush. Therefore, a custom implementation of textured brushes was developed.



(a) `BitmapShader` (b) Custom textured brush

Figure 4.1: Comparison of `BitmapShader` and custom textured brush implementation

To achieve the desired effect with the custom textured brush (Figure 4.1b), the path is divided into segments, and a brush texture bitmap is drawn for each segment. This bitmap is positioned so that it aligns its center with the segment's location on the canvas. The bitmap is then rotated according to the tangent of the path to ensure the texture aligns with the path's direction. To create a seamless pattern, the bitmaps are drawn with overlapping.

4.2.2 Layers

The `Layer` class is responsible for managing the elements within a specific layer, maintaining their drawing order. Initially, in the editor's `onDraw()` method each layer was rendered one by one as a separate Bitmap on the Canvas. During each `draw()`

call, the layer would call the `draw()` method for each of its elements. This approach resulted in substantial performance issues, including completely unresponsive application, when more than seven images were added simultaneously and one of them was moved continuously.

To optimize the process of drawing layers, the number of `draw()` method calls for each element was reduced. At any given time, only one layer can be active. Due to performance reasons, all layers below the active one are merged into a single bitmap, and all layers above are merged into another bitmap. When the editor needs to be updated, it draws these bitmaps on the canvas and the active layer between them.

The main challenge for rendering performance is when a selected element is moved, as it requires the `DrawView` to be constantly updated so the element will move smoothly. The solution involved separating elements in the active layer into two bitmaps based on whether they were drawn before or after the selected element. This reduced the number of `draw()` method calls to only three for the active layer: one for the bitmap with elements below the selected element, one for the selected element, and one for the bitmap with elements above the selected element.

This solution significantly improved the editor's performance, but it has some downsides, such as pixelation of the drawn elements, that can be noticeable when the editor is zoomed in.

4.2.3 Drawing history

History in editors can be implemented using different approaches based on the given requirements and specifics. This subsection provides a general overview of some approaches and then describes how it was implemented in the application.

Solutions overview

Memento pattern

This pattern is used to store snapshots of the entire state of the editor after each action is performed. It is memory intensive, but with this approach, actions where information was lost and cannot be restored with inverse actions can easily be reverted. For example, blurring an image results in the loss of pixel information, and this action can only be reverted by reloading the image from before the blur was applied. When the user wants to undo, the current state is simply swapped out with the previous state. Optimizations can be made with this approach, such as saving only parts of the editor screen by dividing

it into smaller squares and keeping information only on the parts of the screen affected by the action.

Command pattern

Instead of maintaining the states themselves, the actions performed are saved in the form of a command pattern. When the user wants to undo an action, the reverse of that action is executed. Thus, the previous state is not stored in memory but is generated as needed. This approach requires fewer resources but can only be implemented for commands that do not require retaining the previous state for reverting back, as mentioned in the previous paragraph.

Linear history

Editing history can be implemented linearly, meaning there is just one timeline of actions. Users can go back and forth with undo/redo actions one by one. A problem occurs when a user undoes an action and then performs new actions, causing the loss of all future actions' information. This can be solved by using a non-linear, tree structure for history or by simply deleting all redo actions in this case to keep the history consistent.

History tree

Some more sophisticated editors use a tree approach to history. A new branch is created when the user undoes actions and performs new ones. This branching mechanism allows the editor to maintain a complete record of all actions, including divergent paths that result from undoing and redoing at different points in the history.

Implementation

In the application, history was implemented using the linear approach as a command pattern. It is not memory-intensive and is suitable for the vector graphics approach used in the application. Every action in the editor, such as moving an element, rotating an element, adding an element, is wrapped in an instance of a class that implements the `Command` interface (Listing 4.3).

```
interface Command {
    fun execute()
    fun revert()
}
```

Listing 4.3: Command interfaxe

The history is stored in `DrawHistoryHolder` as two linked lists: one for completed actions and one for reverted actions. When the user presses "undo", the last added command is

reverted and is moved to the appropriate list. When the user wants to redo an action, the last undone command reverted. To avoid an inconsistent state of the history, that list is cleared whenever users perform any action in the editor.

4.2.4 Saving the editor state

To fulfill the requirements for look modification, the state of the editor should be serialized and stored. The editor's state is saved as a part of the look in the form of `EditorStateTO`. This class holds serialized elements that were drawn, along with the information about layers and the elements belonging to them.

To effectively restore elements and maintain their appearance across different screen sizes and resolutions, the coordinates and sizes of the elements are converted to relative values based on the dimensions of the `DrawView` they are displayed on. This involves dividing the x and y coordinates of an element by the width and height of the view respectively and subsequently recalculating its size.

4.2.5 Mask saving

Drawings created in the editor should be stored to later map to the user's face as a mask. Masks can be animated if any GIFs are added to it. When multiple GIFs with different numbers of frames are added to the mask, the question arises what should be the final number of frames for the animated mask. Calculating the least common multiple of all frame numbers would yield a perfect looped animation, but this might result in too many frames to handle efficiently. Therefore, the biggest frame number among all existing GIFs is used for the animated mask.

The GIF format supports a maximum of 256 colors per frame and only binary transparency: pixels are either fully transparent or fully opaque. Converting images with masks to GIFs results in the loss of semi-transparent color information. Since the bitmaps have a wide range of colors, converting them to GIF requires quantizing the colors to fit within this limit, which can lead to a loss of color fidelity.

To address this issue, animated masks are stored as multiple PNG files, with each frame saved as a separate file. These frames are then applied sequentially to the `AugmentedFaceNode` as a face mesh texture within a `runnable` with a frame delay to create the animation effect. Non-animated masks are saved as a single image.

4.3 Look saving

To fulfill the requirement that users can share created looks with other users, looks should be serialized and saved in the Firebase Database in the `looks` collection. Listing 4.4 shows the structure of the respective document. Information about applied makeup options and its colors should be stored as list of `MakeupTO` objects. All applied accessories are stored as list of `ModelTO`. Look frames are stored in Firebase Storage in folder with looks' ID.

```
{
  "applied_makeup": MakeupTO[],
  "applied_models": ModelTO[],
  "editor_state": EditorStateTO,
  "is_animated": Boolean,
  "is_public": Boolean,
  "look_id": UUID,
  "preview_ref": A reference to a preview image in the
     storage,
  "name": String,
  "author": String,
  "created_at": Timestamp
}
```

Listing 4.4: Look document structure

4.4 Assets creation

This section provides a guide on how to add new makeup, accessories, brushes, and images to the application.

4.4.1 Makeup

To create a texture that will be rendered on top of a user's face, the canonical face model [56] should be imported into a 3D model editor, for example, Blender. Using the UV Texture Editor, a UV-mapped texture [57, pp. 39-53] can be created and saved in the PNG format for future use in the application. Figure 4.2 shows the lipstick drawn on the UV texture, and Figure 4.3 depicts how this texture appears in the application.

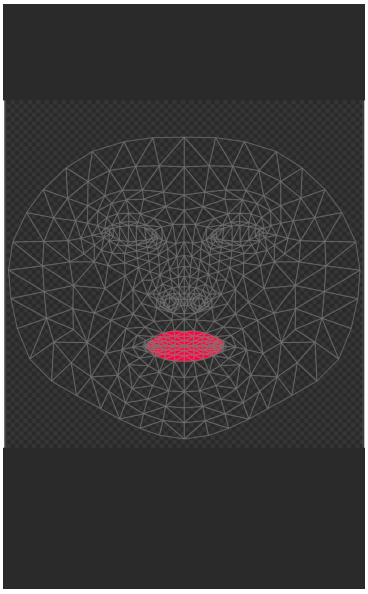


Figure 4.2: Texture in Blender

Figure 4.3: Texture in the application

After uploading the image with a makeup texture to the storage, a document with the structure shown in Listing 4.5 should be added to the `makeup` collection. The "type" attribute should be an ID from the `makeup_types` collection. If a new makeup type is needed, the corresponding document must first be added to the `makeup_types` collection.

```
{
  "ref": A reference to a texture image in the storage,
  "type": Makeup_type ID
}
```

Listing 4.5: Makeup document structure

4.4.2 Accessories

To create a 3D model that will be rendered on the user's face, the canonical face mesh must also be initially imported into a 3D model editor. The model should be positioned according to the canonical face mesh (Figure 4.4) and then exported as glTF or GLB.

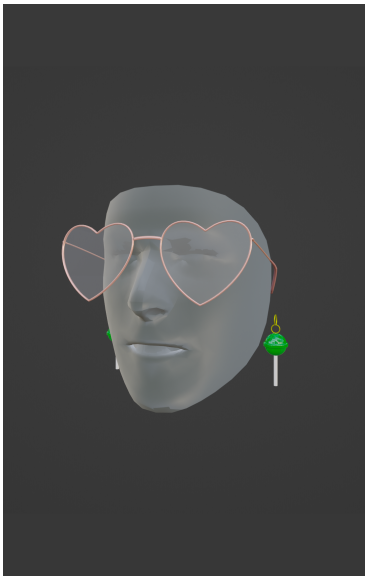


Figure 4.4: Models in Blender

Figure 4.5: Models in the application

Files for the accessory model and its image preview should be stored in Firebase Storage. A corresponding document, structured as shown Listing 4.6, should be created in the Firestore Database under the `models` collection.

Each accessory must have a "type" attribute, whose value should be the document ID from the `accessory_types` collection. These types define the menu in which one the accessories will be displayed. If a new type of accessory needs to be created, a new document needs to be added to the `accessory_types` collection.

Only one model can be applied at a time for a specific slot. If two models share the same slot, applying a new model will replace the previous one. If the "slot" attribute is left empty, the model will use a unique slot and will not be replaced by any other model.

```
{
  "preview_ref": Reference to preview image in storage,
  "ref": A reference to a model in the storage,
  "slot": String,
  "type": Accessory_type ID
}
```

Listing 4.6: Model document structure

4.4.3 Brushes

Brushes are dynamically created from documents in the `brushes` collection and can have attributes described in Listing 4.7. If a brush has a texture, the image with this texture on transparent background should be uploaded to the storage. Attributes "stroke_cap", "stroke_joint", "style" and "blur_type" use enumerated values from the class Paint [54].

```
{
  "stroke_cap": ROUND|BUTT|SQUARE,
  "stroke_joint": MITER|ROUND|BEVEL,
  "style": STOKE|FILL|FILL_AND_STROKE,
  "blur_radius": Int,
  "blur_type": NORMAL|SOLID|OUTER|INNER,
  "texture_ref": A reference to a brush texture in the
     storage
}
```

Listing 4.7: Brush document structure

4.4.4 Images and GIFs

Images and GIFs can be uploaded in the application or directly via Firebase. The image or GIF should first be uploaded to the storage, and a corresponding document should be created in the `images` collection; its structure is shown in Listing 4.8.

```
{
  "created_at": Timestamp,
  "ref": A reference to an image in the storage,
  "is_animated": Boolean
}
```

Listing 4.8: Image document structure

Chapter 5

Testing

This chapter details the process of testing the application, employing a usability testing methodology. The testing consisted of two iterations; participants followed scenarios written in Subsection 5.1.1. The first iteration, described in Subsection 5.1.2, involved two users whose feedback resulted in several changes to the application. The second iteration, described in Section 5.1.3, was conducted with five additional participants who tested the enhanced version of the application. Each participant in the second iteration completed a survey described in the conclusion (Section 5.2). The conclusion also features the changes made after the second iteration and possible future improvements or new functionalities proposed by the users.

5.1 Usability testing

In his book "Don't Make Me Think" [58, Chapter 9], Steve Krug describes usability testing as a process involving the observation of real users as they interact with an application to identify any issues or areas for improvement. Unlike focus groups, which provide opinions and feelings about the design, as described in Chapter 3, usability testing focuses on practical interaction. It helps determine if users can effectively use the application to accomplish specific tasks.

The application was tested in two phases to identify and address the most notable issues early on before performing usability testing with a larger user base. Before each test, participants were given a brief overview of the application concept. They were instructed to follow the steps described in the next section and think out loud.

Participants in the first iteration were the members of the focus group and were familiar with the application from the design process. The testers in the second iteration were seeing the application for the first time.

5.1.1 Scenarios

Scenarios for usability tests describe specific tasks that testers should complete because they cover the most significant use cases from Section 3.3.

Scenario 1: Look Creation

1. **Sign Up and Log In:**
   (a) Open the application and sign up with a new account.
   (b) Log in with the newly created account.
2. **Apply a Makeup:**
   (a) Apply bottom lashes mascara in green color.
   (b) Add and combine with other types of makeup.
3. **Add Accessories:**
   (a) Apply star-shaped glasses.
   (b) Add and combine with other types of accessories.
4. **Create and Edit a Face Texture:**
   (a) Open the 2D editor to draw the texture for the look.
   (b) Draw a line in blue with a textured brush.
   (c) Change the position and size of the line.
   (d) Delete the drawn line.
   (e) Add a star shape in red color, not filled. Place it on the eye; its size should be slightly greater than the eye.
   (f) Add an animated image of a burger to the nose.
   (g) Rotate it by 90 degrees.
   (h) Create a new layer.

(i) Draw a rectangle on the new layer.

(j) Move the image of a burger to the newly created layer.

(k) Swap the order of the layers.

(l) Hide the layer with the image of a burger.

5. **Take and Save a Photo:**

   (a) Take a photo with the created look.

   (b) Save the look and make it non-visible to other users.

   (c) Share the link to the created look.

Scenario 2: Look Modification

1. **Find and Apply a Look:**

   (a) Find and apply the look with red and blue stars on the eyes.

2. **Edit the Look:**

   (a) Change the color of the red star to match the blue star and make both stars not filled, just outlined.

   (b) Upload a custom image and place it inside the drawing.

   (c) Return to the previous screen without saving changes made in the editor.

Scenario 3: Look Deletion

1. **Delete Existing Elements:**

   (a) Delete all accessories, makeup, and texture added to the face.

2. **Find and Apply a Look:**

   (a) Find the look created in the first scenario.

   (b) Apply it.

3. **Edit the Look:**

   (a) Change its visibility for other users to public.

4. **Delete the Look:**

   (a) Delete the previously modified look.

5.1.2 First iteration

User #1

Device: OnePlus 9 Pro - Android 11
The user reduced the size of elements so much that they disappeared from the editor's screen. He suggested implementing a minimum size for elements to prevent this issue. Additionally, the user found the scaling of drawn lines odd, expecting the line thickness to change with the line length, while in the current version, the thickness remained the same.

The user was confused when an image placed outside the face grid in the editor was not displayed on his face. He suggested providing a warning about this or cropping the image in the editor when it is out of bounds. Another area of confusion was manipulating the order of layers; he suggested making layer order changes more obvious. He also mentioned that keeping the button for look sharing in the look info menu could be more intuitive and expected it to be somewhere on the main screen for easier access.

Revealed bugs: Several bugs were found during testing, such as a deleted GIF still appearing on the user's face and look elements remaining visible in the editor after removal. After this test, the mentioned problems were fixed.

User's suggestions on future development: The user suggested integrating Giphy to allow GIFs to be added directly from it into the application's 2D editor.

User #2

Device: Samsung Galaxy S22 - Android 13
The user easily navigated the accessories and makeup menus but noticed that the preview images of black models were barely visible due to the dark menu background.

In the 2D editor, the user did not recognize the size slider and suggested changing its design. Furthermore, the user mentioned difficulty making the brush smaller due to the slider's maximum value being too large. She also noted that smaller brush sizes would be used more often than the massive ones.

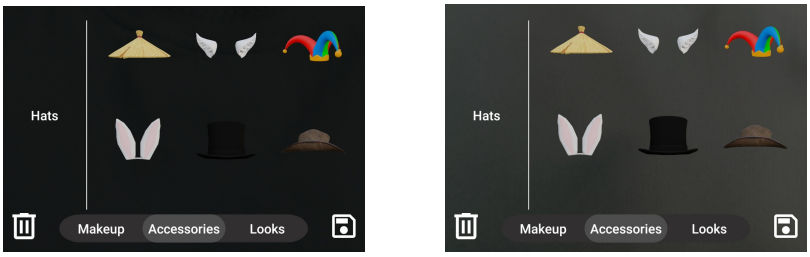
The test participant found uploading a custom image frustrating. She suggested moving the upload button from the "burger menu" button in the editor to the image menu directly and displaying the last uploaded image first.

User's suggestions on future development: The user suggested adding the ability to preview 3D models in a separate window with options to zoom in, rotate, and send links to the models to other users to try them on.

Improvements

Several main usability flaws of the application surfaced during the first testing phase. Addressing them before continuing with testing would allow participants to focus on another area in the second phase and notice different problems.

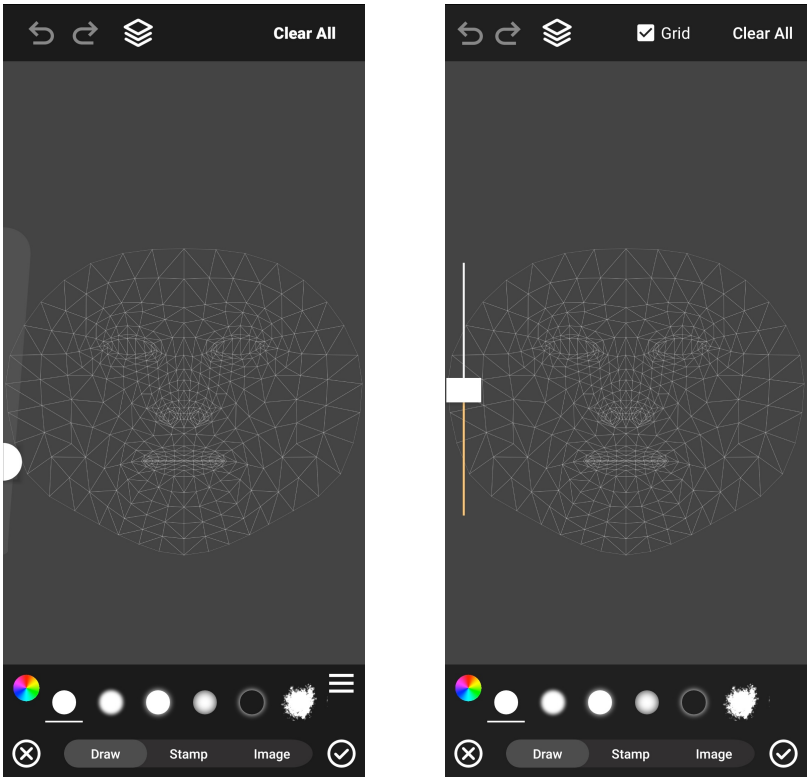
Figure 5.1 shows how the transparency of the menu was decreased to help the visibility of dark preview images.



(a) Original menu opacity (b) Updated menu opacity

Figure 5.1: Comparison between the original and the updated menu opacity

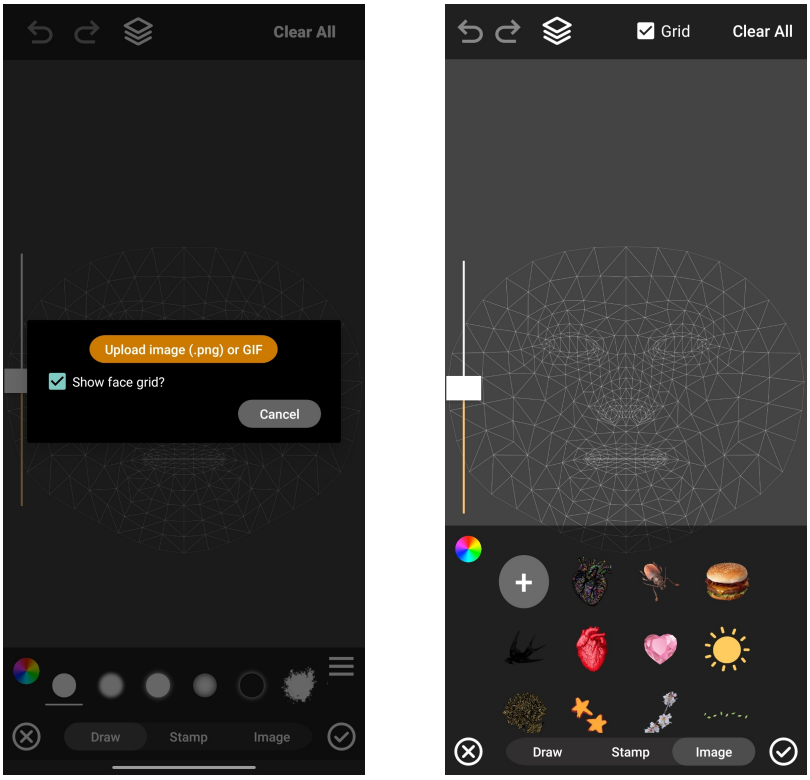
Using the application became more comfortable thanks to the expansion of the active area of the buttons. Drawn lines now scale in thickness as well as length. The redesign of the size slider in the 2D editor, as depicted in Figure 5.2, made it easier to use and appear more recognizable. Sliders have a larger thumb and a reduced max value. Setting minimum width and height for each element made it impossible to make them too little so that they disappear.



(a) Original size slider (b) Redesigned size slider

Figure 5.2: Comparison of the original and redesigned size sliders

The editor's UI became tidier thanks to removing the burger menu button. Rather than opening a menu that offers two unrelated functionalities, a newly added round plus button at the beginning of the images menu allows uploading an image (Figure 5.3). The option to turn off the face grid in the editor was moved on the top of the screen. The order of images in the menu has also been changed from random, based on automatically generated IDs in the database, to being sorted by upload date.

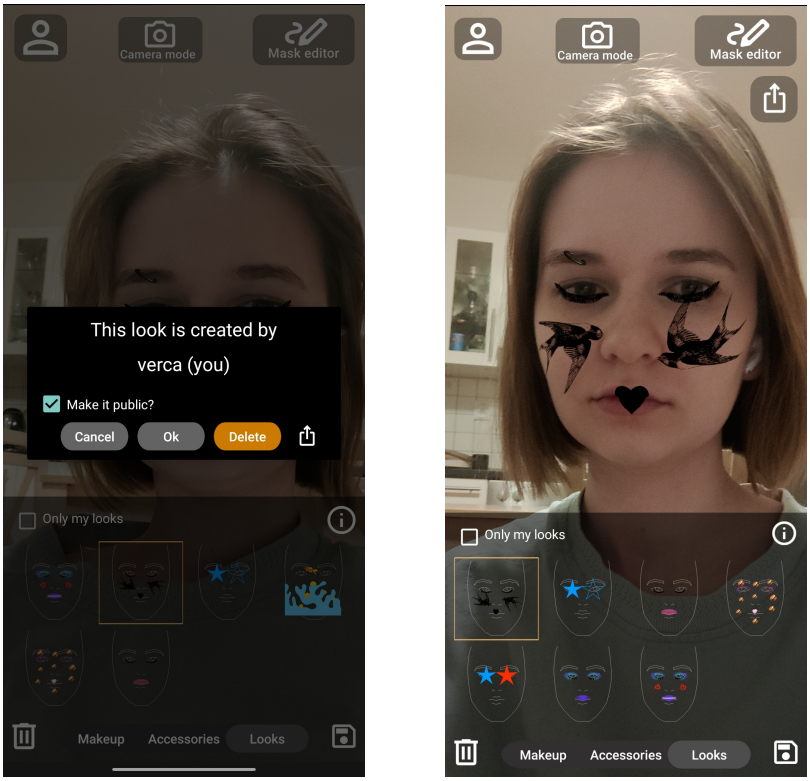


(a) Original image upload (b) Redesigned image upload

Figure 5.3: Comparison of the original and redesigned image upload menus

Another overhaul relates to the layers menu: previously, the numbers representing different layers were based on their index positions and displayed from highest to lowest. These numbers did not change when layers switched their positions, making it unclear to users which layer they controlled. The application started assigning a unique number to each layer in the new version to enhance this functionality. The number increments with each new layer created, and when the order of layers is changed, those numbers are updated accordingly, making their new position in the order more apparent to the user.

The last non-intuitive feature reported by the testers from the first iteration was how to share a link to the look. The share button was initially placed inside the look info pop-up window, making it difficult to find. In the updated version, the share button appears directly on the screen whenever a look is applied, making it more visible and easier for users to find and share, as shown in Figure 5.4.



(a) Original link sharing (b) Redesigned link sharing

Figure 5.4: Comparison of the original and redesigned link sharing

5.1.3 Second iteration

In the second phase of usability testing, the new application version, incorporating changes based on feedback from the first iteration, was tested.

User #1

Device: Samsung Galaxy A41 - Android 12
The user noted that he did not like that the first thing he sees when reopening the application while already logged in is the window with the front camera. He would prefer a home screen before that. The user easily navigated through the makeup and menu with accessories and understood how to select or edit elements. However, the most challenging part was understanding how to move an element between layers. He tried using the layer reordering buttons in the layers menu and did not understand their purpose. The user also needed assistance when tasked with changing the color of an existing element. He suggested that the most obvious way would be to select the element and choose a new color from the palette rather than opening the element menu and selecting the ”Change color” option.

User's suggestions on future development: The user would like to change the background while taking a photo and suggested adding support for multiple faces to try looks on with his friends.

User #2

Device: Samsung Galaxy A52 - Android 14
Navigating the makeup and accessories menu was troublesome for the user; he needed help understanding how to go back and that the horizontal menus were scrollable. He struggled to select elements for editing and suggested implementing a gesture, such as tapping on the element with two fingers and holding or having a separate button to enable element selection instead of drawing. The user also struggled with changing the color of an existing element, similar to User #1. Additionally, he found it challenging to locate the button that deletes a look; he found its placement inside the 'info' menu irrational and suggested a dropdown menu popping out after pressing the look icon for some time. Overall, while completing most of the steps, the user required guidance due to the complexity of interacting with the application.

User's suggestions on future development: The user suggested merging lines drawn close to each other into a group.

User #3

Device: OnePlus 9 Pro - Android 11
This user encountered issues similar to those of Users #1 and #2. The most challenging parts were deleting a look and understanding how to select an element for editing. He tried to find the button that would allow him to select elements. The user did not like the screen with camera mode in the application, suggesting showing a small preview of the photo after taking it and providing a link to the gallery instead.

User's suggestions on future development: The user sees potential in integrating the application with social media, such as sharing created looks on Instagram as a lens. He also suggested expanding the editor's functionality with layer modifications, such as changing individual layer opacity and adding filters like "multiply" or "overlay".

User #4

Device: Samsung Galaxy A54 - Android 13
This user quickly understood how to manage layers and move elements between them. She encountered the same problems mentioned earlier and noticed that when the layers

menu is open, the size slider is invisible, which could result in users forgetting about it altogether. She also found that the location of the checkbox that changes the stamp style from fill to outline inside the palette was not intuitive, and she struggled to find this option.

User's suggestions on future development: The user would like to incorporate social networking features into the application, such as the ability to rate looks, sort looks by trending, view the profile of the look's author, and check out other creations from the same author.

User #5

Device: OnePlus 9 Pro - Android 11
Similar to User #4, managing layers was intuitive for this participant. The main areas of confusion were the same as for the previous testers: deleting a look and selecting elements. The user commented that the "Delete" button in the look info menu should not be on the right side or be orange because its color does not match the corresponding action, which is removal. He also noticed that when the images menu is open, the "Hide layer" button in the layers menu is not visible, and he suggested fixing this issue.

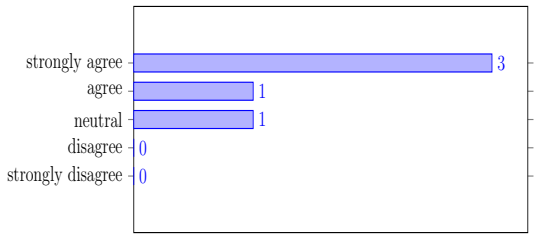
User's suggestions on future development: The user proposed adding the ability to upload their 3D models to try them on in the application.

5.2 Conclusion

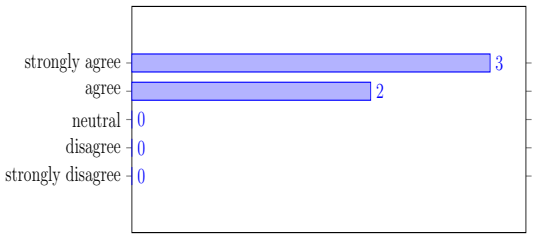
5.2.1 User survey results

After the testing phase, each participant from the second iteration was asked to complete the satisfaction survey. They could answer how much they agreed with the statement in the question on a Likert scale.

The results from the satisfaction survey for Q1 (Figure 5.5a) and Q2 (Figure 5.5b) indicate that most users found trying on different types of makeup and accessories easy and intuitive. However, one user mentioned that the menu navigation needed to be more intuitive. The positive feedback suggests that the test group received the accessory and makeup application features well.



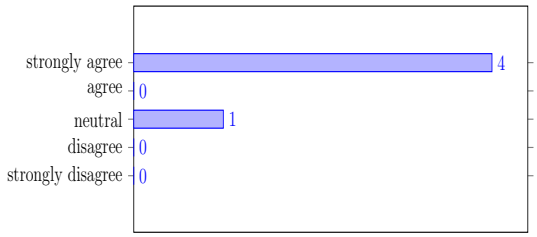
(a) Q1 - Trying on different types of makeup was easy and intuitive



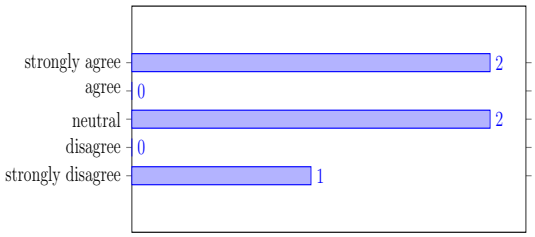
(b) Q2 - Trying on different types of accessories was easy and intuitive

Figure 5.5: User satisfaction survey results (Q1 - Q2)

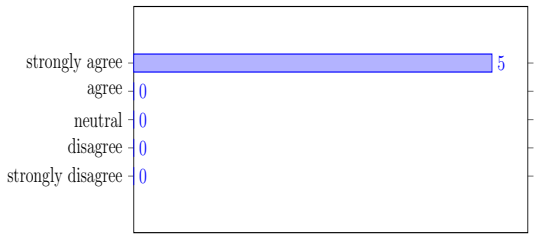
For Q3, the majority of participants found saving the look to be straightforward (Figure 5.6a). However, most testers struggled with deleting a look, indicating significant room for improvement regarding this feature (Figure 5.6b).



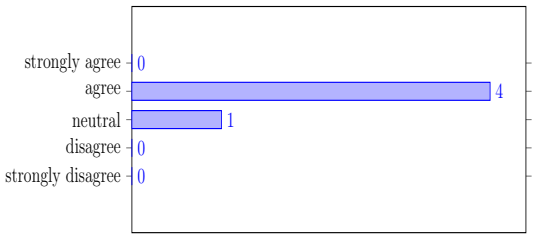
(a) Q3 - Saving a created look was straightforward



(b) Q4 - Deleting a created look was easy to do



(c) Q5 - Sharing a created look with others was easy to do

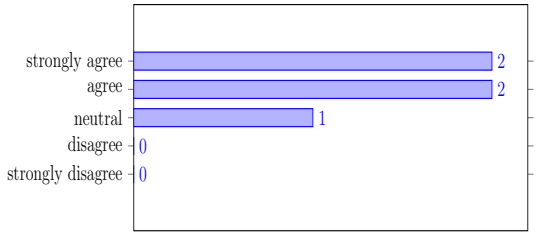


(d) Q6 - Creating drawings in the 2D editor was intuitive

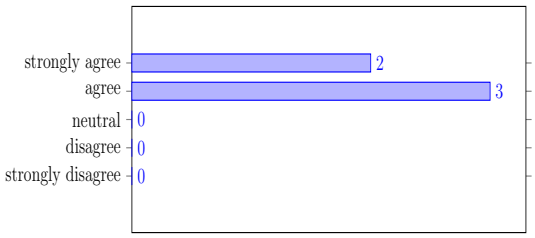
Figure 5.6: User satisfaction survey results (Q3 - Q6)

According to Q5, the participants unanimously agreed that sharing a created look posed no issue, which indicates that the improvements in the UI after the first iteration of testing were successful (Figure 5.6c).

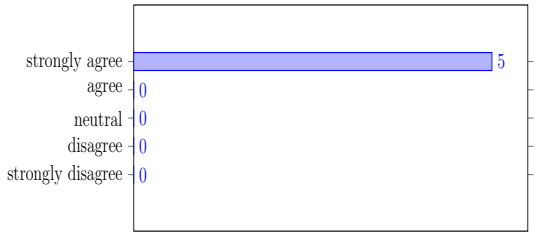
Q6 presented a mixed feedback on the drawing in the 2D editor. The negative feedback primarily revolved around the element selection process, indicating a potential area for improvement (Figure 5.6d).



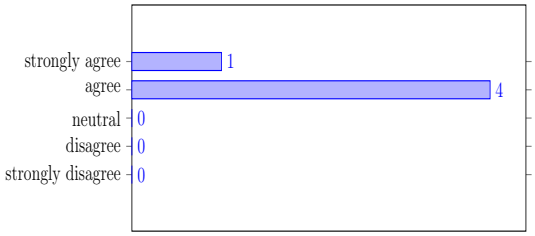
(a) Q7 - Managing layers in the 2D editor was easy



(b) Q8 - Editing elements (rotating, moving, scaling, color changing) was intuitive



(c) Q9 - Uploading images and GIFs to use in the 2D editor was easy



(d) Q10 - Taking a photo within the application was straightforward

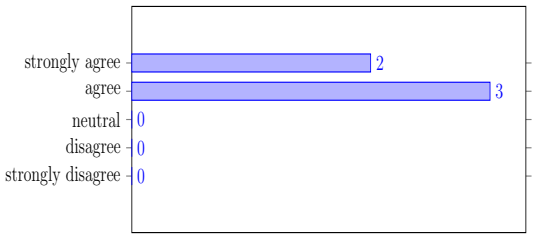
Figure 5.7: User satisfaction survey results Q7 - Q10

Q7 results indicate that most participants perceived managing layers in the 2D editor as simple. It suggests that the improved version of the layers menu after the first iteration became more intuitive for users (Figure 5.7a).

Q8 shows high satisfaction with the element editing features (Figure 5.7b). For Q9, all participants strongly agreed that uploading images and GIFs in the 2D editor was easy, demonstrating that the adjustments between the testing iterations were effective (Figure 5.7c).

Q10 indicates that users positively received the photo-taking feature. However, all users suggested enhancements to make it more similar to the standard camera application, such as creating a small preview with a link to the gallery after taking a photo and adding a shutter sound (Figure 5.7d).

Finally, Q11 results indicate overall satisfaction with the application, with an average rating of 4.4 out of 5. Participants were generally happy with the application's functionality and usability but noted room for improvement (Figure 5.8a).

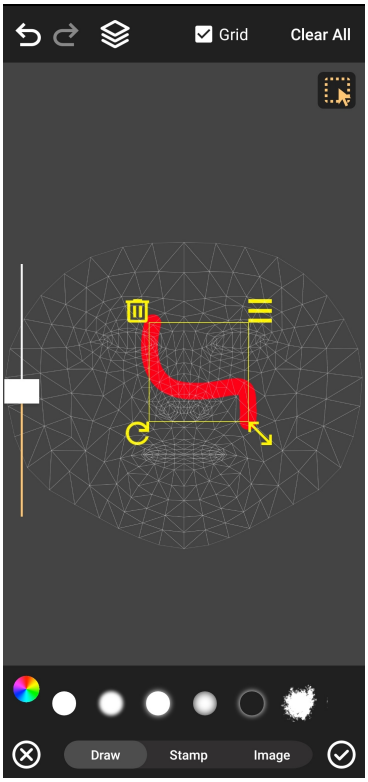


(a) Q11 - Overall, I am satisfied with the application

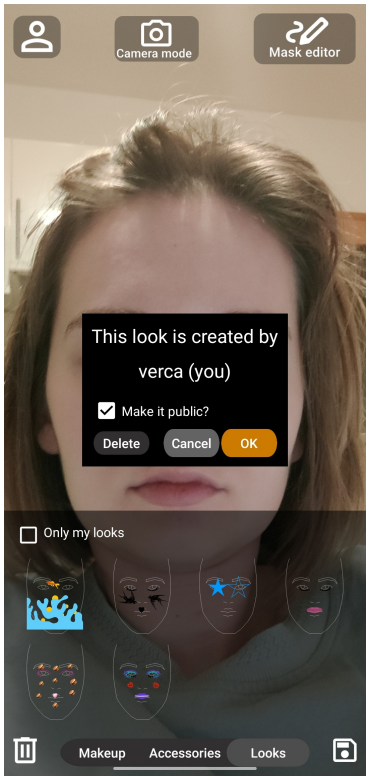
Figure 5.8: Overall satisfaction with the application

5.2.2 Improvements

During the second round of testing, it became obvious that all testers had problems with the same functionalities. Selecting an element from the brushes or stamps menu for editing and deleting a created look turned out to be the most problematic among the test group.



(a) New selection mode button



(b) Redesigned look info window

Figure 5.9: Improvements made after the second iteration

Most users tried to find a button to select the element or attempted to select it with a two-finger long press. In the original UI, users had to deselect the brush to stop drawing

and enter selection mode, which was not intuitive. Therefore, a newly added button allowed users to select elements even if the brush or stamp was selected. Figure 5.9a displays the updated design of this feature.

Additionally, 3 out of 5 testers, when asked to change the color of an existing element, tried to do so by selecting the element and then choosing the new color from the palette. Therefore, the new version of the application allowed changing the element color this way. In the original design, the button for changing the element color appeared after pressing the burger menu on the selected element.

When asked to delete a look, most of the testers tried to do so by long-pressing the look preview image, intuitively expecting a menu to appear. After redesigning the looks menu, the menu appears after the user holds their finger on the look icon for some time instead of pressing the 'info' button, as depicted in Figure 5.9b. Additionally, the buttons on the menu were rearranged, with the delete button being moved to the left and becoming less emphasized to reduce the likelihood of accidental presses.

Chapter 6

Conclusion

This thesis covers research on existing applications and editors that use AR to enhance user appearance. It provides an overview of technologies that allow the creation of such an application and describes its implementation for the Android platform, utilizing ARCore for face tracking and Sceneform for rendering.

The application design was conducted using a user-centered design methodology. Functional and non-functional requirements and use cases were derived from the focus group's feedback. Architecture and class diagrams defined the application architecture, sequence diagrams illustrated the flow of interactions within the application. High-fidelity wireframes were drafted for the UI design and also shown to the focus group, and their feedback was incorporated.

The application's technical aspects were then explained, covering the combination and presentation of makeup and accessories on the user's face, the implementation of the 2D editor, including finger drawing, layers, drawing history, serialization of the editor's state, and saving masks. It follows with a guide on uploading new content to the application.

The application underwent two rounds of usability testing. In the first testing iteration, two participants identified several significant flaws in the application, which were fixed. Five participants tested the improved version in the second iteration, and most of them identified the same usability issues and expressed general satisfaction with the application. After that, its final version resolved some of those usability problems.

6.1 Future work

User feedback from the usability testing in Chapter 5 identified several areas for future development. Those suggestions aim to enhance the user experience and extend the

application's functionality.

UI revision During testing, several issues with the UI were found, such as the layer menu hiding the size slider and the images menu cropping part of the layer menu. Some users found the makeup and accessories menu to be too cramped. For better usability, the UI of the application should be adjusted.

Social Media Integration The user experience could be further improved by adding social media features to the application: functionalities such as the ability to rate looks, sort looks by what is currently trending, and view the profiles of look creators. Additionally, the possibility of exporting created looks into compatible formats for use as lenses on Instagram and Snapchat should be analysed.

Advanced Editing Features Expanding the editor's functionality to include layer filters, such as multiply, overlay, or hard light, would give users more creative control. Adding the ability to customize the background behind the user would also improve the creative possibilities.

Multiple Faces Future versions could include support for multiple faces, allowing users to try looks with friends. However, as of today, ARCore on Android devices does not support multiple face tracking.

Preview of 3D models in the look Future versions could create previews of 3D models used in the look and display them in the preview image.

3D Models uploading and Giphy integration Allowing users to upload 3D models, change their size and position on the face, and integrate Giphy for adding GIFs directly into the 2D editor would enrich the application's content creation possibilities.

Appendix A

Used software

In accordance with the Methodological guideline No. 5/2023 [59], the following software was used during the development of this thesis:

- ChatGPT [60]. Models: GPT-4, GPT-4o. For text style feedback and and rephrasing suggestions.
- Grammarly [61]. For grammar, spelling checking and style suggestions.

Bibliography

[1] E. Klopfer, *Augmented learning: Research and design of mobile educational games*. MIT press, 2008.

[2] P. Daponte, L De Vito, F Picariello, and M Riccio, "State of the art and future developments of measurement applications on smartphones", *Measurement*, vol. 46, no. 9, pp. 3291–3307, 2013.

[3] Snapchat, *Gucci ar try on*, `https://forbusiness.snapchat.com/inspiration/gucci-ar-tryon`, Accessed: January 21, 2024, 2020.

[4] ModiFace Inc., *Modiface - official website*, `https://modiface.com/index.html`, Accessed: January 13, 2024, 2024.

[5] ModiFace Inc., *Modiface patent - system and method for light field correction of colored surfaces in an image*, `https://patents.justia.com/patent/10430977`, Accessed: 2024-01-21, 2017.

[6] ModiFace, Inc., *Modiface patent - end-to-end merge for video object segmentation (vos))*, `https://patents.justia.com/patent/20210150728`, Accessed: 2024-01-21, 2020.

[7] Warby Parker, Inc., *Warby parker - official website*, `https://www.warbyparker.com/`, Accessed: 2024-01-13, 2024.

[8] Apple, *Apple face id*, `https://support.apple.com/en-us/102381`, Accessed: 2024-01-21, 2024.

[9] Lily Oberstein, *Warby parker - app review*, `https://www.businessinsider.com/guides/style/warby-parker-virtual-try-on-app-review`, Accessed: 2024-01-13, 2021.

[10] Inkhunter, Inc., *Inkhunter application*, `https://apps.apple.com/us/app/inkhunter-ai-tattoo-designs/id991558368`, Accessed: 2024-01-13, 2024.

[11] QRea, *Tryo - oficcial web page)*, `https://www.qreal.io/tryo-virtual-try-on-app-for-shopping`, Accessed: 2024-01-21, 2023.

[12] Meta, *Meta spark studio - official webside*, `https://spark.meta.com/`, Accessed: 2024-05-23, 2024.

[13] Snap, Inc., *Lens studio - official webside*, `https://ar.snap.com/lens-studio`, Accessed: 2024-05-23, 2024.

[14] Snap, Inc., *Camera kit - official webside*, `https://ar.snap.com/camera-kit`, Accessed: 2024-05-23, 2024.

[15] Apple, *Arkit - official website*, https://developer.apple.com/augmented-reality/arkit/, Accessed: 2024-01-13.

[16] Apple, *Arkit documentation - face anchor*, https://developer.apple.com/documentation/arkit/arfaceanchor, Accessed: 2024-05-23, 2024.

[17] Apple, *Arkit documentation - blend shapes*, https://developer.apple.com/documentation/arkit/arfaceanchor/2928251-blendshapes, Accessed: 2024-01-13, 2024.

[18] Apple, *Arkit documentation - eye blink*, https://developer.apple.com/documentation/arkit/arfaceanchor/blendshapelocation/2928262-eyeblinkright/, Accessed: 2024-05-23, 2024.

[19] Google, *Arcore overview*, https://developers.google.com/ar/discover, Accessed: 2024-01-13, 2024.

[20] Google, *Arcore documentation - augmented face*, https://developers.google.com/ar/reference/java/com/google/ar/core/AugmentedFace, Accessed: 2024-05-23, 2023.

[21] Google, *Arcore documentation - augmented faces*, https://developers.google.com/ar/develop/augmented-faces, Accessed: 2024-05-23, 2024.

[22] Unity Software, Inc., *Unity ar foundation - features*, https://unity.com/unity/features/arfoundation, Accessed: 2024-01-13, 2023.

[23] J. G. Wang, *Unity 2018 Augmented Reality Projects: Build four immersive and fun AR applications using ARKit, ARCore, and Vuforia*. Packt Publishing, 2018.

[24] Google, *Mediapipe - official website*, https://google.github.io/mediapipe/, Accessed: 2024-01-13, 2024.

[25] Google, *Mediapipe documentation - solutions*, https://ai.google.dev/edge/mediapipe/solutions/guide, Accessed: 2024-05-23, 2024.

[26] Google, *Mediapipe documentation - face landmarker*, https://ai.google.dev/edge/mediapipe/solutions/vision/face_landmarker/android, Accessed: 2024-05-23, 2024.

[27] Brothaler, K., *OpenGL ES 2 for Android: A Quick-start Guide* (Pragmatic programmers). Pragmatic Bookshelf, 2013, ISBN: 9781937785345.

[28] Google, *Arcore android sdk - samples*, https://github.com/google-ar/arcore-android-sdk/tree/master/samples, Accessed: 2024-01-13, 2024.

[29] G. Sellers and J. Kessenich, *Vulkan Programming Guide: The Official Guide to Learning Vulkan* (OpenGL). Pearson Education, 2016, ISBN: 9780134464688. [Online]. Available: https://books.google.cz/books?id=XVBxDQAAQBAJ.

[30] Google, *Arcore documentation - vulkan*, https://developers.google.com/ar/develop/vulkan, Accessed: 2024-01-13, 2023.

[31] Google, *Arcore android sdk - vulkan sample*, https://github.com/google-ar/arcore-android-sdk/tree/master/samples/hello_ar_vulkan_c, Accessed: 2024-01-13, 2023.

[32] M. Romain Guy (@romainguy), *Filament documentation*, https://google.github.io/filament/Filament.html, Accessed: 2024-01-13, 2024.

[33] Robert Chrzanowski (@zirman), *Arcore filament sample app*, https://github.com/zirman/arcore-filament-example-app, Accessed: 2024-01-13, 2021.

[34] Google, *Sceneform documentation*, https://developers.google.com/sceneform/develop, Accessed: 2024-01-13, 2020.

[35] Google, *Sceneform google framework*, https://github.com/google-ar/sceneform-android-sdk, Accessed: 2024-01-13, 2020.

[36] Google and Open Source Contributors, *Sceneform maintained framework*, https://github.com/SceneView/sceneform-android, Accessed: 2024-01-13, 2023.

[37] Google and Open Source Contributors, *Sceneview framework*, https://github.com/SceneView/sceneview-android, Accessed: 2024-01-13, 2024.

[38] *Sceneview - augmented faces issues*, https://github.com/SceneView/sceneview-android/issues/299, Accessed: 2024-05-23, 2023.

[39] T. Lowdermilk, *User-Centered Design: A Developer's Guide to Building User-Friendly Applications*. O'Reilly Media, Incorporated, 2013, ISBN: 9781449359805. [Online]. Available: https://books.google.cz/books?id=XiX5bNJjW0kC.

[40] VMware, *Spring boot - official website*, https://spring.io/projects/spring-boot, Accessed: 2024-05-15, 2024.

[41] PostgreSQL Global Development Group, *Postgresql - official website*, https://www.postgresql.org/, Accessed: 2024-05-15, 2024.

[42] Amazon, *Amazon s3 - official website*, https://aws.amazon.com/s3/, Accessed: 2024-05-15, 2024.

[43] D. Richard Hipp, *Sqlite - official website*, https://www.sqlite.org/, Accessed: 2024-05-15, 2024.

[44] Google, *Firebase - official website*, https://firebase.google.com/, Accessed: 2024-05-15, 2024.

[45] Google, *Firebaseui for storage - documentation*, https://firebaseopensource.com/projects/firebase/firebaseui-android/storage/readme/, Accessed: 2024-05-15, 2024.

[46] Sam Judd, *Glide documentation*, https://bumptech.github.io/glide/, Accessed: 2024-05-15, 2024.

[47] Google, *Android documentation - activity*, https://developer.android.com/reference/android/app/Activity, Accessed: 2024-05-15, 2024.

[48] Google, *Android documentation - fragments*, https://developer.android.com/guide/fragments, Accessed: 2024-05-15, 2024.

[49] Google, *Sceneform documentation - augmented face node*, https://developers.google.com/sceneform/reference/com/google/ar/sceneform/ux/AugmentedFaceNode, Accessed: 2024-05-21, 2020.

[50] Google, *Arcore documentation - augmented face*, https://developers.google.com/ar/reference/java/com/google/ar/core/AugmentedFace, Accessed: 2024-05-21, 2023.

[51] Google, *Android documentation - canvas*, https://developer.android.com/reference/android/graphics/Canvas, Accessed: 2024-05-21, 2024.

[52] Google, *Android documentation - path*, https://developer.android.com/reference/kotlin/androidx/compose/ui/graphics/Path, Accessed: 2024-05-21, 2024.

[53] Google, *Android documentation - motion event*, https://developer.android.com/reference/android/view/MotionEvent, Accessed: 2024-05-21, 2024.

[54] Google, *Android documentation - paint*, https://developer.android.com/reference/kotlin/androidx/compose/ui/graphics/Paint, Accessed: 2024-05-21, 2024.

[55] Google, *Android documentation - bitmapshader*, https://developer.android.com/reference/android/graphics/BitmapShader, Accessed: 2024-05-21, 2024.

[56] Google, *Arcore android sdk - assets - canonical face mesh*, https://github.com/google-ar/arcore-android-sdk/tree/master/assets/canonical_face_mesh.fbx, Accessed: 2024-01-13, 2019.

[57] Mullen, T., *Mastering Blender* (EBL-Schweitzer). Wiley, 2012, ISBN: 9781118330562. [Online]. Available: https://books.google.cz/books?id=NBEGht_s3UkC.

[58] S. Krug, *Don't Make Me Think, Revisited: A Common Sense Approach to Web Usability* (Voices That Matter). Pearson Education, 2013, ISBN: 9780133597264. [Online]. Available: https://books.google.cz/books?id=QlduAgAAQBAJ.

[59] ČVUT, *Rámcová pravidla používání umělé inteligence na čvut pro studijní a pedagogické účely v bc a nm studiu*, https://intranet.fel.cvut.cz/cz/rozvoj/MP-pouzivani-ui.pdf, Accessed: 2024-05-21, 2024.

[60] OpenAI, *Chatgpt - official website*, https://chatgpt.com/, Accessed: 2024-05-21, 2024.

[61] I. Grammarly, *Grammarly - official website*, https://www.grammarly.com/, Accessed: 2024-05-21, 2024.