Máster Universitario en Internet of Things
Master in Internet of Things

Master's Thesis		
Title		
Author		Signature
Tutor / Co-tutor		Signature
Director		Signature
Board of examiners		
Presidente/ President		
Secretario/ Secretary		
Vocal		
Date		
Grade		

Secretary

Abstract

Recipe applications are useful kitchen helpers when it comes to cooking. Nowadays, numerous applications that allow users to save and recreate recipes exist, some of which have the same or similar functionalities as well as different ones, to differentiate themselves in a highly competitive market. In this paper, a new recipe collection application for Android is presented, called *PanPal*, which provides the user with the opportunity to create its personalized recipe collection, as well as an interactive hands-free cooking mode. This cooking mode allows the user to follow easily and step-by-step the recipe without having to touch the phone. Instead, the navigation is done through hand gestures, which are captured by the front camera of the smartphone and translated into steering actions. The objective of this work is to present the work done to design, develop, and validate this application. For the implementation of the gesture control, the open-source library *MediaPipe* by Google was integrated into the application, which supplies a computer vision algorithm and a pre-trained model for gesture recognition. After designing and developing the application, it was verified technically and validated practically. For the practical validation, user tests were performed, and a survey was conducted, which included the system usability scale post-test questionnaire. This assessment measures the system's perceived usability and achieved an average score of 84.8 out of 100 amongst all test users. This grade comes very close to the equivalent of an adjective rating of *Excellent*, meaning that the application provides a pleasant and intuitive user experience to users. Finally, test users made requests for additional functionalities, which could enhance the application's usability and popularity.

Resumen

Las aplicaciones de recetas son útiles ayudantes de cocina a la hora de cocinar. En la actualidad, existen numerosas aplicaciones que permiten a los usuarios guardar y recrear recetas, algunas de las cuales tienen funcionalidades iguales o similares, así como otras diferentes, para diferenciarse en un mercado altamente competitivo. En este trabajo, se presenta una nueva aplicación de recopilación de recetas para Android, denominada *PanPal*, que ofrece al usuario la posibilidad de crear su colección de recetas personalizada, así como un modo de cocina interactivo manos libres. Este modo de cocina permite al usuario seguir fácilmente y paso a paso la receta sin tener que tocar el teléfono. En su lugar, la navegación se realiza mediante gestos de la mano, que son captados por la cámara frontal del smartphone y traducidos en acciones de dirección. El objetivo de este trabajo es presentar el trabajo realizado para diseñar, desarrollar y validar esta aplicación. Para la implementación del control gestual, se integró en la aplicación la librería de código abierto *MediaPipe* de Google, que proporciona un algoritmo de visión por computador y un modelo pre-entrenado para el reconocimiento de gestos. Tras diseñar y desarrollar la aplicación, se verificó técnicamente y se validó en la práctica. Para la validación práctica, se realizaron pruebas con usuarios y una encuesta, que incluía el cuestionario posterior a la prueba de la escala de usabilidad del sistema. Esta evaluación mide la usabilidad percibida del sistema y alcanzó una puntuación media de 84,8 sobre 100 entre todos los usuarios de las pruebas. Esta nota se acerca mucho al equivalente de una calificación adjetiva de *Excelente*, lo que significa que la aplicación proporciona una experiencia de uso agradable e intuitiva a los usuarios. Por último, los usuarios de las pruebas solicitaron funcionalidades adicionales, que podrían mejorar la usabilidad y popularidad de la aplicación.

Contents

List of Figures

List of Tables

List of Listings

Abbreviations

IoT	**I**nternet **o**f **T**hings
ICT	**I**nformation **C**ommunications **T**echnologies
RFID	**R**adio-**f**requency **Id**entification
AI	**A**rtificial **I**ntelligence
OpenCV	**Open C**omputer **V**ision
ML	**M**achine **L**earning
SUS	**S**ystem **U**sability **S**cale
MVVM	**M**odel - **V**iew - **V**iew**M**odel
MVC	**M**odel - **V**iew - **C**ontroller
UI	**U**ser **I**nterface
RGBA	**R**ed **G**reen **B**lue **A**lpha
IDE	**I**ntegrated **D**evelopemnent **E**nvironment
XML	E**x**tensible **M**arkup **L**anguage

Chapter 1

Introduction

Cooking is one the most important tasks in the everyday life of people. Considering that most people eat at least three times a day and most of those meals are prepared at home, humans spend a considerable amount of time during their lives cooking. It comes as no surprise that information and communications technologies (ICTs) have already found their way into the kitchen to help humans during the cooking process [1].

Even more specifically, in recent years, Internet of Things (IoT) has been increasingly integrated for developing smart kitchens [2]. The idea of integrating IoT in the kitchen is to "enable things or objects with connectivity through e.g. RFID tags, sensors, actuators, big data analysis, cloud computing, mobile phones, etc." [3]. In this sense, many kitchen appliances have been equipped with small computers and built-in sensors, which opens up many opportunities to provide new services and applications.

Similar to what an algorithm is to IoT, a recipe is what many cooks refer to when preparing a meal. Formerly, recipes were written in books, magazines, notepads, or even single sheets of paper. Nowadays, many digital cooking applications or websites exist, which form huge databases of recipes ([4], [5], [6]). Therefore, it became very easy for individuals to access recipes from all around the world. Today, new recipe applications must be able to demonstrate decisive advantages and differences in order to assert themselves on the market. Some opt for innovative recommendation algorithms [7], others for recipe recommendation depending on the available ingredients [8], and others for an integrated system with a smart kitchen [2].

However, so far, none of them focused on solving one very important practical problem: when cooking, hands normally get dirty. Thus, using a smartphone while cooking is usually neither practical nor hygienic. Hands are ordinarily handling food ingredients and a phone's screen is known to carry many bacteria, which should not end up in one's

food. Existing applications haven't considered this problem yet, especially when offering a so-called "interactive cooking mode", which guides the user (usually the cook) step by step through the recipe and makes it easier for them to follow the recipe.

Developing an application that solves this problem presents a useful solution for many people. This includes people of all ages who have an interest in cooking and can be even particularly beneficial for individuals with upper body mobility limitations, as it enhances accessibility. Having an application that is accessible to all users not only meets essential accessibility requirements but also provides significant advantages by making cooking more inclusive and convenient and thus appeals to a wider range of users.

Additionally to accessibility, usability is also a crucial aspect for this kind of mobile application, ensuring that it is easy to use and intuitive for users of all ages and technical backgrounds. Interfaces have to be designed to be clean and straightforward, minimizing the learning curve and enabling users to quickly adapt to the system. By employing familiar human-phone interactions and providing clear visual feedback, the solution should aim to enhance user satisfaction and efficiency. This focus on usability would not only improve the cooking experience but also encourage consistent use, making cooking more enjoyable and less cumbersome for everyone.

To solve the mentioned problem, PanPal, a hands-free interactive cooking application for Android, was planned, developed, tested, and validated. The cooking mode in this application enables control by gestures. This is what will be called "hands-free" user interaction in this paper. With the proposed application, users do not have to physically touch their smartphone, but rather the interaction should be done through hand gestures in front of the camera.

1.1 Motivation

Using a smartphone with dirty hands is something very few people want to do while cooking. This problem posed the main motivation for the development of the application presented in this thesis. Enabling the possibility of not having to touch the cell phone while cooking and following a recipe on it can be very useful and practical. Additionally, this feature can also significantly benefit physically or visually impaired individuals in several ways:

1. For people with motor disabilities, removing the necessity for precise touch inputs can facilitate navigation through the recipe.

2. For visually impaired people, relying on gestures means they can also follow instructions audibly while focusing on their cooking tasks without needing to constantly refer to the screen.

Furthermore, users want to have the possibility to follow all kinds of recipes hands-free and not only a limited number provided by a cooking application. Therefore, offering the possibility to add personal recipes into the application provides further value.

1.2 Objectives

The goal of this thesis is to design and develop a hands-free interactive cooking application for Android smartphones and test and validate it technically and in real-world scenarios. By incorporating computer vision for gesture recognition in real-time, the project aims to achieve the following objectives:

1. Designing an intuitive user interface that supports gesture commands.
2. Selecting, integrating, and validating a robust image recognition algorithm for accurate and fast human-phone interaction.
3. Covering all essential functionalities for following a recipe with intuitive gestures.
4. Ensuring the application meets usability and accessibility standards to cater to users with different needs.
5. Providing the user with the possibility to add their recipes and follow them hands-free.
6. Testing and validating the application with real and practical user testing, employing testers from different backgrounds and ages.

By achieving these objectives, the project aims to propose a new technological solution for cooks following recipes on their smartphones.

1.3 Thesis Structure

Following the introduction, the thesis will analyze and discuss the current state of the art, meaning the existing cooking applications and frameworks that treat similar topics and present the required technologies behind them.

This will include a review of various open-source frameworks that enable gesture recognition for mobile applications, highlighting their advantages, limitations, and applicability to the presented problem. The section on computer vision will analyze the current literature and cover the fundamental principles and recent advancements in the field, and prevailing gesture recognition systems will be examined.

The description section will provide a comprehensive presentation of the application, detailing the user needs and the methodology employed throughout the project. This section will also include a thorough requirement analysis, presenting both functional and non-functional requirements. The complete system architecture will be detailed, including the setup, the design, and the implementation.

The experiments part will lay out how the application tests were conducted, presenting the methodologies, test cases, and metrics used to evaluate its performance and finally their results. Concluding this thesis, the budget for the development of the application will be calculated, the findings will be summarized, limitations and future work challenges will be addressed, and theoretical and practical implications will be drawn.

Chapter 2

State of the Art

In preparation for this paper, a research of existing literature and presented projects in the domain of smart kitchens and cooking applications was conducted. The goal of this review was to gain an understanding of the current state of the art, what different approaches have already been tried, and which ones can be built upon to solve the problem discussed in this paper. The results are presented in the following section.

2.1 Existing Cooking Applications

Cooking applications have existed since the beginning of the Google Play Store and the App Store. It should therefore come as no surprise that hundreds, if not even thousands of cooking applications already exist. One of the most popular ones is NYT Cooking [4] from the New York Times. With over 100 million users [9], it combines a large database of easy-to-follow recipes with an intuitive design. Epicurious [10] offers more than 50 thousand recipes, alongside personalized recommendations, video recipes, and cooking tips. KptnCook [11] is an application that proposes easy and fast recipes, where users can indicate their eating preferences, and the recipe propositions are adapted to those. Incorporating artificial intelligence (AI) to suggest recipes to users and give them cooking advice, Tasty [6] is an application that offers many video recipes where users can save their preferred recipes. Yummly [5] is another recipe and meal planning application, that allows users to search for recipes according to the recipes they have on hand and the time they want to dedicate to cooking. This application also offers a cooking mode, where users are taken step by step through the recipe, offering a clearer overview and making it easier to follow.

Feature/Application	NYT Cooking	Epicurious	KptnCook	Tasty	Yummly
Large Collection of Recipes	Yes	Yes	Yes	Yes	Yes
Personalized Recommendations	Yes	Yes	Yes	Yes	Yes
Step-by-step Cooking Mode	No	No	No	Yes	Yes
Hands-free App Control	No	No	No	No	No
Option to add own Recipes	No	No	No	No	Yes
Weekly Meal Planner	No	No	No	Yes	Yes
Advanced Search Options	Yes	Yes	No	Yes	Yes
Add Ingredients to Grocery List	Yes	No	Yes	No	Yes
Free Usage	No	No	Yes	Yes	Yes

TABLE 2.1: Comparison of Popular Cooking Applications

Table 2.1 compares the different features of some of the most popular existing cooking applications. As it becomes apparent, every application offers different features to users and therefore specializes in different highlights. Nevertheless, some features are shared by all of them: having a large database, bringing together thousands of recipes, and offering personalized recommendations based on user preferences. These are usually determined by asking the user for their dietary tastes and restrictions, as well as analyzing the saved and already cooked recipes.

However, it should be noted that none of those applications offer hands-free app control as is the idea of the proposed application in this paper and only Yummly offers the possibility to add new recipes from the user. This opens the door for the suggested application to enter this uncovered niche and presents a new opportunity for users to have a completely personalized recipe collection. Since the market is already crowded with many different applications, new developers have to invent new features and it becomes crucial to offer innovative value offers to stand out from the competition.

2.2 Computer Vision

According to the company IBM, Computer Vision can be defined as "a field of artificial intelligence that uses machine learning and neural networks to teach computers and systems to derive meaningful information from digital images, videos, and other visual inputs" [12]. In a more detailed approach to defining this field, Richard Szeliski defines it in his book as the development of mathematical techniques to recover the three-dimensional shape and appearance of objects in imagery [13]. The goal thereby is to allow computers to explain images with a very high level of detail and causality.

Nowadays, it is already used in a wide variety of fields and systems, such as optical character recognition for reading handwritten postal codes on letters or automatic number plate recognition, self-driving vehicles, medical imaging, surveillance, motion capture, and many more. For gesture recognition, which is the area to which this paper refers, two main techniques are necessary.

First, *recognition*: it focuses on classifying people or objects in an image. Narrowed further down, it is instructed with the task of recognizing faces, hands, or other smaller parts of humans and animals. Second, *feature detection and matching*: in computer vision, features are structures in the image, such as points, objects, or edges. Recognizing these features and matching them together consists of a crucial task to understand correctly the images from the point of view of a computer. How gesture recognition is applied in real-world applications will be explained in the next chapter which will show and describe some existing systems.

Some systems which employ computer vision in the food and farming industry are presented by Kakani et al. [14]. In their paper, they describe how computer vision and AI are used to help the food industry to be more productive and profitable. Some employments of those techniques serve to find pests in the crops, develop food processing strategies, or in quality assessment and strategy. A mobile recipe application that uses object recognition for food ingredients is proposed by Maruyama et al. [8]. Their application can be used to detect ingredients such as vegetables and meats and the application proposes recipes that can be cooked with these ingredients. The idea of this application is to use it during the groceries shopping.

2.3 Gesture Recognition Systems in Practice

Numerous gesture recognition systems and frameworks exist and are applied for diverse use cases. One framework developed by Google is MediaPipe Solutions [15]. MediaPipe

Solutions provides several tools and libraries to apply AI and machine learning in in-house developed projects. Some of the available solutions include object detection, face detection, and, more importantly for this project, gesture recognition and hand landmark detection. The gesture recognition solution detects hand gestures in real-time and is useful for calling application features based on those gestures. The gesture recognizer can work with static images, video, or live streams. The outputs are hand landmarks, handedness (left/right hand), and the hand gesture categories of multiple hands. To detect hands and gestures, the task works with two pre-packaged model bundles: a hand landmark model bundle and a gesture classification model bundle. The hand landmark bundle detects the hand and the gesture classifier the gesture. 2.1 shows which hand landmarks can be detected to recognize a hand by the gesture recognition task from MediaPipe.

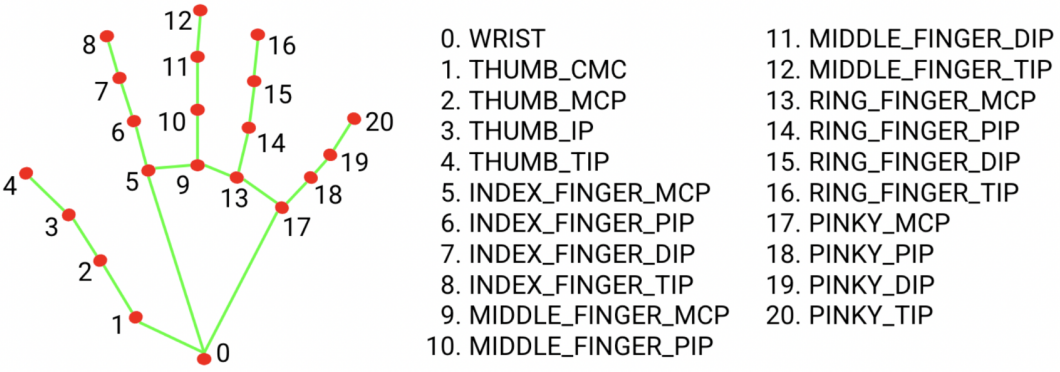


FIGURE 2.1: Hand Landmark Model Bundle from MediaPipe

Another popular framework for gesture recognition is OpenCV [16]. In a paper from 2016, Naveenkumar & Vadivel [17] explain what OpenCV is and how it can be used for computer vision applications. According to them, it can be used for image filtering, image transformation, object tracking, and feature detection. They present various real-life examples that can be applied to this, such as an intruder alarm system using motion detection, an authentication system using face detection, and video processing using an Android phone. Other alternatives include ML Kit [18], TensorFlow Lite [19], and ManoMotion [20]. Each of those proposes different features and advantages, as well as disadvantages, which will be compared roughly in the next chapter.

Some researchers have already looked into how gesture recognition can be built into mobile applications and some applications have been released with this function in the app stores. Already in 2014, Saxena et al. [21] developed an algorithm to detect hand gestures using an Android device. The camera provides the image, which is then pre-processed. Using a trained neural network, the gesture is recognized and the result is

provided. Gesture recognition algorithms as presented before still work comparable to this.

Similarly, Lahiani et al. [22] developed another system for gesture recognition in real-time. In their paper, the researchers describe the exact process of how this works using the OpenCV framework on an Android device. They already thought about the possibility of using this kind of system for deaf people to communicate with each other.

A slightly different approach was proposed with an air-swipe gesture recognition system for Android [23]. Instead of recognizing static hand gestures in front of the camera, this system recognizes hand movements such as swipes from left to right or up to down, without having to touch the screen. Although this application also uses the OpenCV framework, the algorithm behind the gesture detection is different, as it has to capture the movement and location of the hand.

Finally, in a more recent approach, a user guide application with gesture recognition was developed using the MediaPipe framework [24]. The goal of this application is to navigate through a user guide, which consists of a menu with eight different widgets, solely using hand gestures and without touching the phone. Although the developers used MediaPipe, they took profit from the possibility of training their model with custom hand gestures, allowing the user to use different hand gestures than included with the standard model.

Chapter 3

Description

In this chapter, an overview of the user needs will be given and the development methodology will be presented. Additionally, an explanation of the design choices and a detailed description of the implementation of the proposed application will be provided.

3.1 User Needs

The objective of this work is to develop and present a mobile application for Android smartphones, which provides an easy-to-follow, step-by-step cooking mode for recipes and allows the user to interact with the device hands-free through gesture recognition. It is assumed that the proposed system works on a smartphone that has a built-in front camera facing the user. The user should be able to get familiar before and use the system easily and intuitively during the cooking process. The setup is as follows: users place the phone close to the kitchen workspace with the screen and front camera facing the user. When activating the cooking mode for a recipe they want to cook, the recipe steps will be displayed one-by-one on the screen. To navigate through these steps, the user can make hand gestures that the system understands and will respond accordingly to them. Seven different hand gestures can be recognized by the system, each of them having a different function: thumbs up, thumbs down, open palm, pointing up, closed fist, victory, and I love you. The phone screen will provide in a small widget the camera perspective showing the user and display the name of the detected gesture in real-time, to get instant feedback. Additionally, the application provides the user with the feature to add new recipes to his collection and can adjust the settings for gesture recognition, if he desires a more robust or more sensitive interaction.

3.2 Methodology

This application was developed following the software development life cycle model called *waterfall model* [25]. According to this model, there are six main phases which are planning and requirements analysis, design, development, testing, deployment, and maintenance. Nonetheless, this model was not followed strictly and there was some jumping back-and-forth between some phases, especially between the design and development phases. For each phase, the sub-activities and outputs are identified.

The first phase of the waterfall model consists of planning and gathering requirements from stakeholders and analyzing them to understand the scope and objectives of the project. For this purpose, several non-formal interviews and discussions were conducted with four home cooks to get to know what their expectations of the application are. The gathered requirements were then analyzed and formalized in tables for functional and non-functional requirements. Additionally, a timeline with milestones was created to keep the project on time and track.

The second phase is the design phase. This is where the software architecture, user interface, and system components are outlined. The architecture design defines the overall structure of the system, including the relationships and interactions between different components. A recipe database is needed to save the recipes and a general cooking mode has to be provided for each recipe. Additionally, the gesture control module has to be present in the cooking mode. The user interface design focuses on the layout and navigation to ensure an intuitive and user-friendly experience. Sketches were drawn for a cooking theme and the idea of a bottom navigation bar for user navigation between the recipe collection and the profile settings was adopted. This phase ensures that all aspects of the system are planned and documented, reducing the risk of problems during development.

Next comes the development phase. This phase includes the implementation of the software based on the design specifications. This involves writing the program code for each software component, as well as for the layout files of the user interface. The development phase also includes unit testing, meaning that small tests are written for each component to ensure that each component of the software is working as expected. This phase consisted of the main part of this project and is thereby crucial for transforming the design into a functioning system while maintaining high code quality and functionality.

The boundary between the first 3 phases was not always strictly adhered to, as ideas for new functionalities were also added during the development phase. As a result, these 3 phases were iterated over and over again.

Fourth comes the testing. Here, the software is tested as a whole, meaning that the whole application is tested on a mobile phone by real users in order to ensure that it meets all the requirements and is free from any problems or bugs. For this application, the app was made available for test users and a questionnaire conducting the system usability scale (SUS) assessment [26] was sent to each test user. According to the results from this questionnaire, the app was then evaluated and refined. The results of these tests and the analysis are presented in chapter 4.

The fifth step encompasses the deployment. Once the software has been tested and approved, it should be deployed to the production environment, in this case, published to the Google Play Store [27]. At the time of writing this paper, this is the current phase of the project. To be published in the Play Store, different criteria have to be met such as configuring the application for release and building and signing a release version.

The sixth and last step is the maintenance. This phase is responsible for fixing any issues that arise after the application has been published, providing new updates with bug fixes and new features, and ensuring that it continues to meet the requirements over time. This phase will go on until the application is discontinued and no longer in use.

3.3 Requirement Analysis

As mentioned in the methodology section 3.2, the first step in the development of the proposed application was to gather the requirements, formalize them, and divide them into functional and non-functional requirements. Table 3.1 shows the functional requirements that were defined for this application. These requirements describe what the application should do, meaning that they refer to specific product functionalities.

ID	Name	Description
RCP_SAV_101	Saving Recipes	The system must save recipes with a recipe title, at least one ingredient, and at least one recipe step.
RCP_SAV_102	Optional Recipe Details	The system must allow saving optionally a recipe with an image, a description, and tags.
RCP_COL_101	View Recipe Collection	The system must allow the user to see a collection of all their saved recipes.
RCP_ADD_101	Add New Recipes	The system must allow the user to add new recipes to their collection.
RCP_COL_102	Search Recipes	The system must allow the user to search for a recipe.
RCP_COK_101	Cooking Mode	The system must have a step-by-step easy-to-follow cooking mode.
COK_GES_101	Gesture Control	The system must have a hands-free cooking mode controlled with hand gestures.
COK_GES_102	Adjust Gesture Detection	The system must provide the option to adjust hand gesture detection thresholds.
APP_TUT_101	App Tutorial	The system must provide a tutorial to the user to explain how to use the app and how the gesture control works.

Table 3.1: Functional Requirements for the Application

Especially requirements RCP_ADD_101, RCP_COK_101, and COK_GES_101 are essential to be implemented to achieve the formulated goals of this project. Table 3.2 showcases the non-functional requirements for the application. The idea of those non-functional requirements is to describe and place constraints on how the application should work. In summary, this requirements analysis intends to define all the functionalities that the application must have in order to function correctly and to offer the service that was set to be achieved.

ID	Name	Description
USA_101	Usability	The system shall have an intuitive and user-friendly interface, making it easy for users to navigate and interact with the application.
ACC_101	Accessibility	The system shall be accessible to users with disabilities, such as motor or visual, and comply with accessibility standards and guidelines.
PRV_101	Privacy	The system shall be secured to keep the privacy of the user.
REL_101	Reliability	The system shall be reliable and available for use at all times, minimizing system failures.

TABLE 3.2: Non-Functional Requirements for the Application

3.4 Setup and Design

The setup and design section describes the experimental setup, which gesture recognition framework was implemented and why, and finally the design decisions of the application.

3.4.1 Experimental Setup

The proposed application is built for the Android operating system, as it presents many opportunities as Android is present in many low-, medium-, and high-end mobile phones, representing a large market share for smartphones. The platform also provides multiple open-source libraries, including those for accessibility [28], which facilitate the development of the application. The main programming language needed for this project is Java and the code was written using the integrated development environment (IDE) Android Studio [29]. Android Studio has a built-in Android simulator and can emulate the application on a wide range of Google Pixel phones. As a gesture recognition library, MediaPipe by Google was chosen. The preceding reasons for this choice are explained in the next section.

3.4.2 Selection of Gesture Recognition Framework

As just mentioned, the gesture recognition framework from Google called MediaPipe was chosen to implement the gesture recognition features for this application. MediaPipe presents various advantages useful for the application compared to other similar

frameworks. We will start by describing briefly each of the frameworks presented in 2.3 and presenting the advantages and disadvantages of them.

MediaPipe was already presented in more detail, so this section will only talk about the pros and cons of this framework. The main advantage of it is that it provides real-time hand tracking and gesture recognition with high accuracy and precise results. Additionally, it is easy to integrate with other machine learning models and provides extensive tools for building custom pipelines for future development. The disadvantages are that more processing power may be required and it is not the easiest to set up and configure.

OpenCV is another very popular framework, which provides an open-source computer vision and machine learning software library. The main advantages are that it is highly flexible, therefore usable for both simple and complex computer vision tasks, which include image and video processing. It also provides extensive documentation thanks to its strong community. Disadvantages consist of realizing more manual coding to implement gesture recognition and the requirement of a deeper understanding of computer vision algorithms.

As the third library, *ManoMotion* was developed specifically for hand tracking and gesture recognition on mobile and augmented reality/virtual reality platforms. Since it was explicitly designed for hand tracking, its performance and accuracy are high. However, its functions are limited to these tasks and the commercial use may require a license. Also, due to a smaller community, less documentation is available.

Another framework developed by Google is *ML Kit*. It provides ready-to-use APIs for a variety of tasks, including text recognition, face detection, and gesture recognition. It is easy to integrate with Firebase [30], Google's mobile and web app development platform, and supports both on-device and cloud-based models. However, Firebase was not used in the development of this application, rendering this advantage worthless here. It also offers fewer customization possibilities compared to lower-level frameworks, such as MediaPipe and OpenCV.

Finally, *TensorFlow Lite* is a lightweight version of TensorFlow designed for mobile and embedded devices. It supports a wide range of pre-trained models and tools for optimizing custom models for mobile use. Its most notable advantages are the wide-ranging customization possibilities it offers and its optimization for on-device performance. However, this customization requires a good understanding of machine learning and model optimization, making it more complex to set up and configure.

Criteria	ML Kit by Google	TensorFlow Lite	MediaPipe by Google
Complexity	Low to Medium	Medium to High	Medium to High
Ease of Integration	Easy, especially with Firebase	Moderate, requires ML expertise	Moderate to complex
Customization	Limited	Highly customizable	Highly customizable
Performance	Good on-device performance, can use cloud	Excellent on-device performance	Excellent real-time performance
Documentation and Community	Strong documentation, good community support	Extensive documentation, strong community	Extensive tools, good community support
License	Free	Free	Free

Table 3.3: Comparison of Popular Gesture Recognition Frameworks 1

Criteria	OpenCV	ManoMotion
Complexity	Medium to High	Medium
Ease of Integration	Moderate to complex	Easy to Moderate
Customization	Highly customizable	Limited to hand tracking and gestures
Performance	Good, but may require manual optimization	Good real-time performance
Documentation and Community	Strong community, extensive documentation	Smaller community, less extensive documentation
License	Free	Requires a license for commercial use

Table 3.4: Comparison of Popular Gesture Recognition Frameworks 2

Tables 3.3 and 3.4 show a comparison of those above-mentioned frameworks. After careful consideration and weighing up the advantages and disadvantages of each of them, the choice of which framework to use for the proposed application fell on MediaPipe. For this application, rapid and high accuracy is critical, as otherwise users will get frustrated when trying to use the cooking mode with gesture control and will not see the added value. Having a slow response time or inaccurate readings of gestures would probably

make the users refrain from using the application. Additionally, MediaPipe offers high customization, which offers opportunities for further development, such as adding more hand gestures.

3.4.3 Design

After defining the setup and configurations needed for the project and selecting the best-fitting gesture recognition framework, the next step was to make the design decisions to define the overall system structure.

Regarding the user interface design, the main goal was to maintain a simple and clear scheme to enhance usability and accessibility. The interface was designed to be intuitive, with clear visual hierarchies and consistent layouts. Meanwhile, reducing the number of pages to a minimum was important to support the idea of finding every functionality easily. Similar to many famous and popular applications, the main navigation should be done through a bottom navigation bar.

The design of the database was kept as simple as possible to efficiently store the recipes. Recipes should be implemented as instances of a recipe class and these instances should then be stored together in a dataset. This could be done by using lists, such as an *ArrayList* in Java. The recipes in the dataset should easily be accessible and new ones added through according methods.

The gesture recognition module is a critical component of the application that enables hands-free control using predefined gestures. Each of the seven recognized gestures has a specific function, e.g. advancing to the next recipe step or starting a timer. The gesture recognition algorithm uses the MediaPipe framework, which enables robust real-time hand tracking and gesture classification. To enable gesture control, the module needs access to the front camera of the phone. Bringing together both parts is crucial as otherwise, the gesture control will not work.

Concerning the cooking mode, the choice of implementing a step-by-step approach was adopted, as this ensures an easy-to-follow and laid-out structure to follow a recipe. Every page for every step should look the same, such that the user always knows where to find which information.

A critical task of this application is to bring together the gesture control module with the cooking mode, which has to be available for every recipe. Therefore, each recipe and the cooking mode have to follow a standardized structure, identifying the most important attributes of the recipes that are needed for the cooking mode. This comprises the title,

the ingredients, and the steps, which in turn allows to enable the cooking mode also for newly added recipes.

One of the most important design decisions that was made was to use the Model - View - ViewModel (MVVM) architecture pattern for the code [31]. Using design patterns to structure and write clean code is highly valuable as it helps with the maintenance of the software. Usually, two different design patterns can be used for mobile applications: MVVM and Model View Controller (MVC). However, it is generally considered in the mobile development community that the MVVM pattern overcomes the drawbacks of MVC and is therefore more widely adopted [32]. The goal of MVVM is to separate the data presentation logic (views or user interface) from the core business logic part of the application. There are three different code layers in this pattern, also shown in Figure 3.1:

- Model: This layer is responsible for abstracting the data sources. Model and ViewModel work together to receive and store the data.
- View: The purpose of this layer is to inform the ViewModel about the user's action. This layer monitors the ViewModel and does not contain any application logic.
- ViewModel: This layer provides the data streams that are relevant for the View. It mainly serves as a link between the Model and the View.

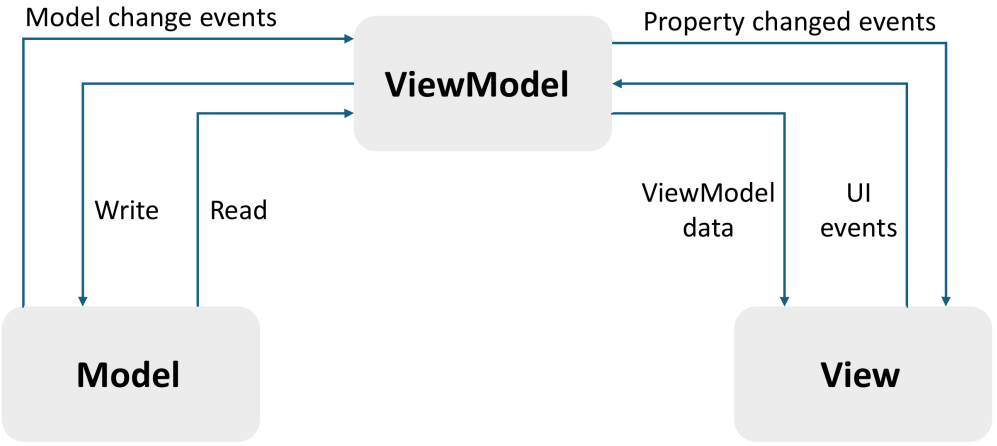


FIGURE 3.1: Model - View - ViewModel Design Pattern

To implement this pattern, a few important principles have to be followed: first, the ViewModel does not hold any kind of reference to the View; second, many-to-1 relationships exist between View and ViewModel, meaning that one ViewModel can be applied

to multiple Views; and third, there are no triggering methods to update the View. To implement MVVM in Android, Google released the DataBinding library [33] to support two-way data binding between the View and ViewModel. This allows any changes in the ViewModel to be automatically reflected in the View and vice versa.

3.5 Implementation

The implementation part will present the user interface of the application and explain how each of the requirements was built-in, how they work, and how they are to be used.

3.5.1 User Interface

The user interface (UI) is where human-computer interaction and communication happen. The application's UI is designed to provide users with an intuitive and natural experience when managing and accessing their recipes. It includes various components and views that enable easy navigation, recipe creation, and hands-free interaction through gesture control.

When starting the application for the first time, the user will be greeted with an application tutorial that guides the user through the application and its functionalities (3.2). The idea of this tutorial is to show the user what the application offers, to present the hands-free cooking mode (3.3), and to explain how the gesture control works and which gestures are available. The tutorial also asks for permission to access the camera when using the application and reminds them that without it, the gesture control will not work. If the user already knows how everything works, they also have the opportunity to skip the tutorial.

After going through the tutorial, the user is presented with the homepage interface. Two different pages are available on this homepage: the cookbook page (3.4) and the profile page (3.5). The navigation between these two pages is available through a bottom navigation view.

The cookbook page presents the user with their recipe collection. Each of the recipes is presented in a small widget, displaying the recipe image, the title, the start of the recipe description, and some tags associated with the recipe. At the top of this page, the user also has the option to search for recipes using the search bar. The profile page allows the user to adjust some settings, for example, to activate notifications, fine-tune the gesture recognition detection thresholds, and rate the app.

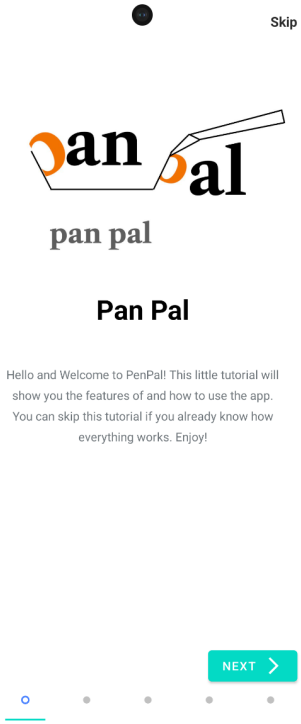


FIGURE 3.2: App Tutorial Welcome Screen

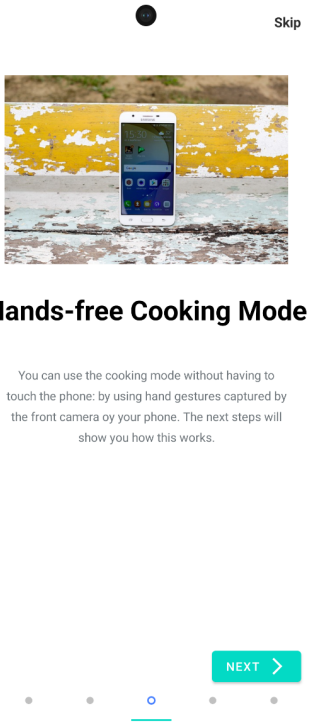


FIGURE 3.3: App Tutorial Hands-Free Cooking Mode

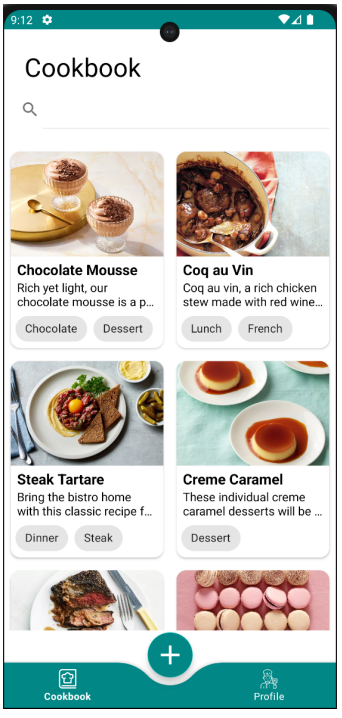


FIGURE 3.4: App Homepage Cookbook

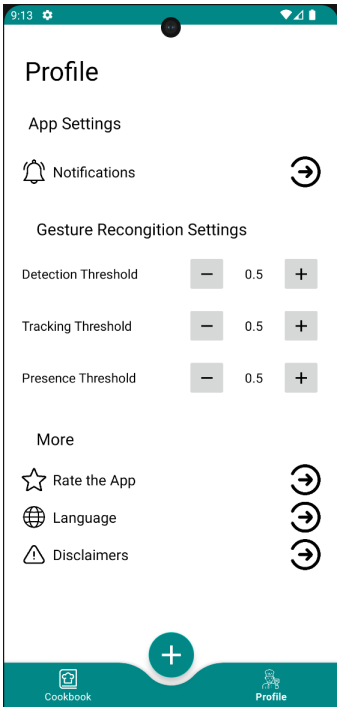
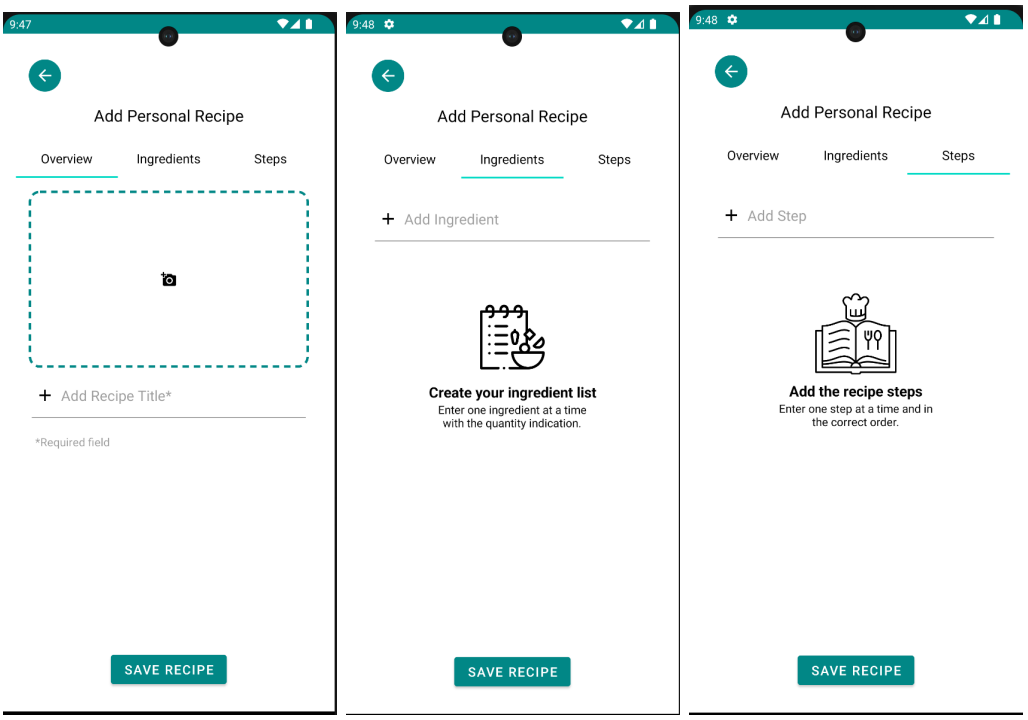


FIGURE 3.5: App Homepage Profile

Just above the bottom navigation bar is a floating action button, which allows the user to add new recipes to their collection. When clicking on the button, the user gets redirected to a page with three different tabs: overview, ingredients, and steps (3.6).



(a) Add Recipe - Overview Tab (b) Add Recipe - Ingredients Tab (c) Add Recipe - Steps Tab

FIGURE 3.6: Add Recipe Page

The overview page allows the user to upload an image of the recipe and provide it with a title. The ingredients tab lets the user add the necessary ingredients one after another and the steps tab is the same for the recipe steps. Providing a recipe title, as well as at least one ingredient and one step is mandatory to be able to save the recipe, which is done by clicking the "save recipe" button at the bottom of the page.

When clicking on a recipe widget in the cookbook page, the user gets redirected to the recipe overview page (3.7). This page shows the recipe image, title, description, and the list of ingredients needed. At the bottom of the page is the "start cooking" button, which will lead the user to the cooking mode.

When starting the cooking mode, the user is confronted with multiple elements. At the top left, they can find the buttons to exit the recipe as well as the help button, which will open a help dialog to guide the user with the possible gestures and the actions they trigger, as can be seen in Figure 3.9. Just below those buttons, the user can see the current step and how many steps the recipe has in total. On the right-hand side of this information, a small image window shows the live stream of the front camera. When

FIGURE 3.7: Recipe Overview Page

a hand is detected, the hand landmarks are detected and shown with blue lines in an overlay view. Below this, the user can find the description of the recipe step, see Figure 3.8.

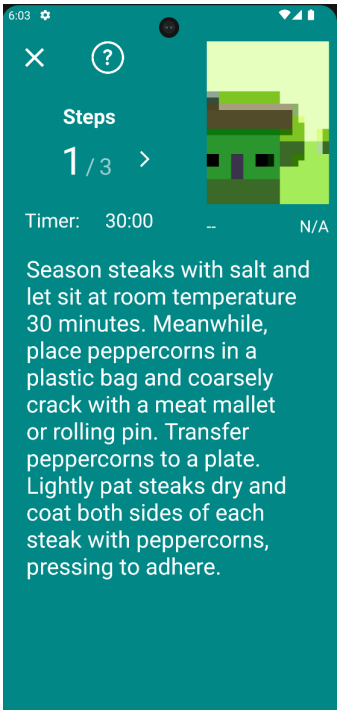


FIGURE 3.8: Cooking Mode Page

To apply the MVVM pattern, each View has to be linked to a ViewModel, which in turn

is linked to the Model. The user interface represents the View layer, being responsible for displaying the data and handling user interactions. The ViewModels hold the UI-related data that is lifecycle-aware, ensuring data persists across configuration changes, such as orientation changes of the phone.

These are all the available views that the proposed application has. The goal was to make the handling and navigation of the application as clear and intuitive as possible so that users could quickly find their way around and adopt it rapidly.

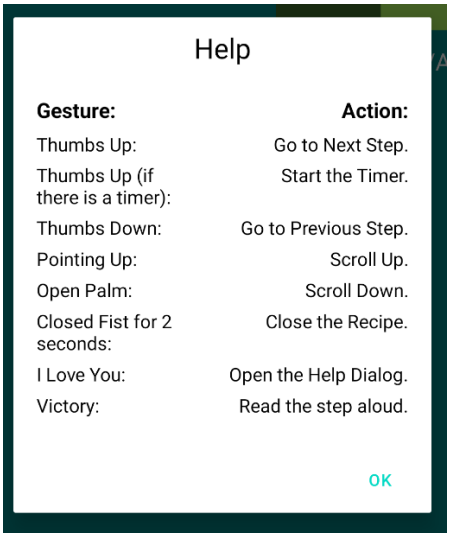


FIGURE 3.9: Cooking Mode Help Dialog

3.5.2 Digital Cookbook

The objective of the proposed application was to provide a digital cookbook to the user, meaning offering a collection of different recipes that the user likes and putting at their disposal the tools to follow these recipes easily. As was defined in the requirements analysis, four functional requirements are given for the digital cookbook: saving recipes RCP_SAV_101, saving optional recipe details RCP_SAV_102, viewing the recipe collection RCP_COL_101, and searching for recipes RCP_COL_102.

The cookbook page is displayed in an Android fragment and has a very basic setup. The scrollable page shows the recipes in a two-column format and at the top is located the search bar. Code-wise, each recipe is stored as an instance of the *Recipe* class, which has six different attributes: *title, description, image, ingredients, steps*, and *tags*. As mentioned by requirements RCP_SAV_101 and RCP_SAV_102, the title, at least one ingredient, and at least one step is necessary to store a recipe. This is to ensure that the recipe can be displayed in the cookbook and that the cooking mode can be activated. If no recipe picture is available, then a default picture will be set. The recipes are

stored in the class *RecipeDataset*, which holds a list of all saved recipes. For testing and demonstration purposes, the dataset was filled with eight example recipes. As there can only be one recipe dataset for the whole application, the singleton design pattern was used to ensure that the class has only one instance and to provide a global point of access to that instance.

To implement this pattern, three steps are needed. First, the constructor of the class has to be made private to prevent direct instantiation. Second, a static member to hold the single instance of the class must be created. Third, a static method to get the instance has to be provided, which creates it if it does not already exist. The code snippet in Listing 3.1 shows how this was done.

```
public class RecipeDataset {
    private List<Recipe> listOfRecipes;
    private static RecipeDataset instance;

    // Method to get the singleton instance
    public static synchronized RecipeDataset getInstance(Context
   context) {
        if (instance == null) {
            instance = new RecipeDataset(context.
   getApplicationContext());
        }
        return instance;
    }

    // Private constructor
    private RecipeDataset(Context context) {
        Resources res = context.getResources();
        listOfRecipes = new ArrayList<>();
    }
}
```

Listing 3.1: Singleton Pattern Implementation for the Recipe Dataset

To see the whole collection of recipes and fulfill requirement RCP_COL_101, the whole dataset is displayed in a RecyclerView in the CookbookFragment (3.4). For requirement RCP_COL_102, the option to search for specific recipes, two options were implemented. First, the search bar at the top of the view was initialized with an *OnQueryTextListener*. As the user enters the text in the search bar, the collection is instantly searched for matching recipes and only those are then displayed. The second option is to filter using the tags of the recipes. By overriding the *onTagClicked* method, a filter method is called, showing only the recipes that contain the same tag.

3.5.3 Gesture Recognition Method

Integrating a gesture recognition method is one of the hardest tasks that was conducted during the development of the application. As detailed previously, the framework that was to be used for this function is MediaPipe due to its advantages concerning performance and accuracy. To integrate MediaPipe into the Android project, two steps are required. First, including the necessary dependencies in the project to ensure that MediaPipe libraries were available for use. Second, the MediaPipe gesture recognizer task requires a trained model bundle that is compatible with this task. Google provides various pre-trained models and also the possibility to train customized models with different gestures. For this application, the pre-trained model *Hand Gesture Classifier* was used [34]. The solution architecture of this model says that it works as a two-step neural network pipeline with an embedding model followed by a classification model. The pipeline works with hand landmarks and runs for a single hand. This model has to be downloaded and stored within the project directory, for which a *download_tasks.gradle* file was added to the project. In this file is a task to download the gesture recognizer task file from Google and save it.

To set up the gesture recognizer task, a class called *GestureRecognizerHelper* was created which handles the setup as can be seen in Listing 3.2. According to the MediaPipe documentation, the function *createFromOptions()* is the one setting up the task and accepts customizable values for the configuration options.

```
// Initialize the gesture recognizer using current settings on
the
// thread that is using it. CPU can be used with recognizers
// that are created on the main thread and used on a
background thread, but
// the GPU delegate needs to be used on the thread that
initialized the recognizer
public void setupGestureRecognizer() {
    BaseOptions.Builder baseOptionBuilder = BaseOptions.
builder();

    switch (currentDelegate) {
        case DELEGATE_CPU:
            baseOptionBuilder.setDelegate(Delegate.CPU);
            break;
        case DELEGATE_GPU:
            baseOptionBuilder.setDelegate(Delegate.GPU);
            break;
    }
```

```
        baseOptionBuilder.setModelAssetPath(MP_RECOGNIZER_TASK);

        try {
            BaseOptions baseOptions = baseOptionBuilder.build();
            GestureRecognizer.GestureRecognizerOptions.Builder
    optionsBuilder =
                    GestureRecognizer.GestureRecognizerOptions.
    builder()
                            .setBaseOptions(baseOptions)
                            .setMinHandDetectionConfidence(
    minHandDetectionConfidence)
                            .setMinTrackingConfidence(
    minHandTrackingConfidence)
                            .setMinHandPresenceConfidence(
    minHandPresenceConfidence)
                            .setRunningMode(runningMode);

            if (runningMode == RunningMode.LIVE_STREAM) {
                optionsBuilder
                .setResultListener(this::returnLivestreamResult)
                .setErrorListener(this::returnLivestreamError);
            }
            GestureRecognizer.GestureRecognizerOptions options =
    optionsBuilder.build();
            gestureRecognizer = GestureRecognizer.
    createFromOptions(context, options);
        } catch (RuntimeException e) {
            gestureRecognizerListener.onError("Gesture recognizer
    failed to initialize. See error logs for details");
        }
     }
```

Listing 3.2: Setting up the Gesture Recognizer Task

Additionally, this task is set up only for camera live stream as is specified through the running mode. The task could potentially also accept images and videos, however this is not necessary for the application. The following options are set for the gesture recognizer:

- runningMode: As just mentioned, this attribute sets the running mode for the task, which is a live stream. When choosing live stream, a resultListener must also be configured to set up a listener to receive results asynchronously.

- minHandDetectionConfidence: This aspect dictates the minimum confidence score that the model must achieve for a detection to be considered successful.
- minHandPresenceConfidence: This setting determines the minimum confidence score of the hand presence score in the hand landmark detection model. In the live stream mode of the gesture recognizer, if the hand presence confidence score is below this threshold, it triggers the palm detection model. Otherwise, a lightweight hand-tracking algorithm is used to determine the location of the hand for subsequent landmark detection.
- minTrackingConfidence: This setting determines the minimum confidence score needed for the hand tracking to be considered successful. This threshold measures how much the hand's bounding box in the current frame overlaps with the hand's bounding box in the previous frame. In the live stream mode of the gesture recognizer, if the tracking fails, the gesture recognizer triggers hand detection. Otherwise, the hand detection is skipped.
- resultListener: This listener has to be implemented for the live stream mode. It is used to receive the classification results asynchronously
- errorListener: This feature sets an optional error listener to recover the potential error.

The values for minHandDetectionConfidence, minHandPresenceConfidence, and minTrackingConfidence can lie between 0.0 and 1.0 with a default value of 0.5. Seven different gestures can be recognized by the Hand Gesture Classifier model: *Closed_Fist, Open_Palm, Pointing_Up, Thumb_Down, Thumb_Up, Victory*, and *ILoveYou*. Additionally, a *None* gesture can be recognized, if the hand is detected, but no gesture is identified.

The next step for the gesture recognizer is to prepare the incoming live stream data. The task handles all of the data input pre-processing, including resizing, rotation, and value normalization. Each frame is converted into an *MPImage*, which is the wrapper class for MediaPipe image objects and how the framework handles images.

After the data has been prepared, the gesture recognizer is fed with the MPImage, and the task is run. This is done using the *recognizeAsync()* method, which triggers inferences. This means that for gesture recognition, detecting hands in the image, detecting hand landmarks, and recognizing hand gestures from the landmarks are performed.

The final step is to handle and display the results. For every recognition attempt, a gesture detection result object is generated. This result object consists of the detected hand

gestures, the handedness, meaning whether it is a left or right hand and the hand landmarks in image coordinates and world coordinates. These results are all tied together in a *ResultBundle*, which is implemented as a static inner class in the *GestureRecognizerHelper* class. In addition, the result bundle contains the inference time, the input image height, and the input image width.

In conclusion, five steps are needed for the integration of the gesture recognition module. First, the necessary dependencies have to be added and the trained model has to be downloaded and saved. After that, the gesture recognizer task has to be set up and configured, followed by a preparation of the incoming data. This data is then fed to the task, with which the task can be run and finally, the results are returned and presented. The next section will show how this procedure was integrated into the cooking mode to allow hands-free control of the application.

3.5.4 Hands-free Cooking Mode

The hands-free cooking mode is probably the main distinctive functionality of the presented application, as it proposes a new way of interacting between the human and the recipe present on the smartphone. The idea of this feature is to solve one simple problem most people have when cooking: handling the phone whilst having dirty hands. Controlling the phone without having to touch it can facilitate the cooking process and resolve this major inconvenience.

To do so, the presented application aims to integrate the gesture recognition module into the interactive cooking mode, where users follow the recipe step-by-step. Numerous functionalities are needed for this.

First, the camera has to be set up and bound to the use cases for hand gesture recognition using the *CameraX* module from Android. By binding the use cases, the camera is configured to use the front-facing lens and sets up a preview with a 4:3 aspect ratio and the appropriate rotation. In addition, an image analyzer is set up to process each frame using the "RGBA 8888" format, matching the requirements of the hand gesture recognition model. After this is done, the *recognizeHand()* method can be invoked to detect hand gestures using the GestureRecognizerHelper. All of this is done using a single background thread, which ensures that the tasks are executed sequentially on the same background thread, which is ideal for tasks that need to be executed individually and without simultaneous execution, such as processing the camera frames.

Also, the cooking mode has to implement the GestureRecognizerListener interface (3.3), which handles the outputs provided by the result bundle.

```
public interface GestureRecognizerListener {
    void onError(String error);
    void onResults(ResultBundle resultBundle);
}
```

LISTING 3.3: GestureRecognizerListener interface

The *onError()* method simply displays a Toast message to the user if an error occurs. The *onResults()* method has a more important function (3.4). It displays the name of the recognized gesture, as well as the recognition score. Furthermore, an overlay view shows the hand landmarks on the hand to visualize the recognition process. Most importantly, the gesture is processed to execute the corresponding action that is tied to it.

```
@Override
public void onError(@NonNull String error) {
    runOnUiThread(() -> Toast.makeText(getApplicationContext()
, error, Toast.LENGTH_SHORT).show());
}

@Override
public void onResults(@NonNull GestureRecognizerHelper.
ResultBundle resultBundle) {
    runOnUiThread(() -> {
        if (binding != null) {
            // Update TextViews with gesture name and score
            GestureRecognizerResult gestureRecognizerResult =
resultBundle.getResults().get(0);

            if (gestureRecognizerResult.gestures().isEmpty())
{
                cookingModeViewModel.updateGestureResults("--"
, "N/A");
                return;
            }

            List<Category> categories =
gestureRecognizerResult.gestures().get(0);

            if (categories != null && !categories.isEmpty()) {
                Category topCategory = categories.get(0);

                processGesture(topCategory);
```

```
                cookingModeViewModel.updateGestureResults(
topCategory.categoryName(), String.format(Locale.US, "%.2f",
topCategory.score()));
            } else {
                cookingModeViewModel.updateGestureResults("--"
, "N/A");
            }

            binding.overlay.setResults(
                    resultBundle.getResults().get(0),
                    resultBundle.getInputImageHeight(),
                    resultBundle.getInputImageWidth(),
                    RunningMode.LIVE_STREAM
            );

            binding.overlay.invalidate();
        }
    });
}
```

LISTING 3.4: onError() and onResults() Implementation for the Cooking Mode

Each hand gesture is bound to an action in the cooking mode, which allows the user to navigate through it without having to touch the phone.

The first action is closing the recipe with a closed fist gesture. Since a closed fist is a very common gesture that people do often, a timer mechanism is added to prevent misunderstandings of closing the recipe when this was not intended. The timer is used to measure if the closed fist gesture is recognized and held by the user for more than two seconds without interruption, only after which the recipe is closed.

The second gesture is the thumbs up. This gesture has two functions, depending on the context. Usually, this gesture is associated with continuing to the next recipe step. To prevent the cooking mode from advancing too fast to the next recipe steps, a debounce mechanism is implemented. This mechanism ensures that the user only advances to the next step every two seconds if the gesture is held constantly, thus allowing the user to make the gesture calmly and avoid skipping to the next step too rapidly. The second action of the thumbs up gesture is to start the timer, if the current recipe step has a timer. The timer is always indicated below the display of the current step and is dependent on the step description. When a step has a timer, the first thumbs up gesture will start the timer. The second thumbs up gesture will stop the timer and make the user advance to the next cooking step. After the timer has been started, the user can

exit the app or turn off the phone and the timer will continue running, until it finishes, upon which a ringtone alarm will ring. For this purpose, the *onStart()* method of the activity has to be overridden to save the timer start time in the *SharedPreferences* and retrieve it when reopening the app after exiting it. The idea of this feature is that the user can do other things while the timer is running, which usually signals a waiting time in the recipe, e.g. when a dish is cooking in the oven and therefore the user does not need to have the recipe open on their phone.

The contrary gesture to thumbs up is thumps down. The corresponding action is going back to the previous step. As with the thumbs up gesture, a debounce mechanism is used to prevent skipping steps too fast.

The corresponding gesture to scroll down if the step description is too long to fit on the whole page is the open palm. This gesture moves the text upwards by a certain scroll value, which is 200 density-independent pixels (dp) by default.

The opposite of scrolling down is scrolling back up. This is done by doing the pointing up gesture. As for the scrolling up action, the scroll value is 200 dp. Both scrolling gestures are also equipped with the previously mentioned debounce mechanism, which prevents the scrolling action from being executed several times in a row.

In the case that the user forgets which action corresponds to which gesture, the help dialog can be opened with the I love you gesture. Once again, this gesture is configured with the debounce mechanism to prevent the help dialog from being opened multiple times.

The final gesture is the victory, also known as peace, gesture. This gesture enables another accessibility and convenience feature, which is reading the text of the step description. To implement this feature, the Android text-to-speech library has to be used. Instead of reading the whole text, even if it is not currently visible because it is too long to fit on the whole screen, only the currently visible text is read aloud. This also works if the user scrolls down or up, and every time only the currently visible text is read. This restriction is applied through the *readStepDescription()* method, which calculates the visible text based on the scroll view height and the text height.

These are all the available gestures and their corresponding actions for the interactive hands-free cooking mode. These are all the necessary actions that are needed to navigate through this cooking mode and complete the recipes without having to touch the phone. This fulfills functional requirements RCP_COK_101 and COK_GES_101 and represents the main work of this project.

3.5.5 Gesture Recognition Settings

According to requirement COK_GES_102, the application must provide the user with the option to adjust the hand gesture detection thresholds. The idea of this requirement is to adjust these settings according to the current cooking environment, which can vary, especially concerning the lighting conditions. This could also benefit persons with restricted motor skills, who are not able to make the gestures so clearly, by making the gesture recognition looser and not requiring the gesture to be perfectly executed.

Three different thresholds can be adjusted according to the users' needs. They were already presented in section 3.5.2 and concern the minHandDetectionConfidence, minHandPresenceConfidence, and minTrackingConfidence. Each of these values could theoretically be adjusted between 0.0 and 1.0 (according to the MediaPipe documentation). However, in this application, the values are restricted to lie between 0.2 and 0.8, since having values below 0.2 would usually recognize gestures that are incorrect or too fast, and values above 0.8 would require the gestures to be perfectly executed with perfect lighting conditions, which is very hard to achieve and would make the application cumbersome to use.

Whenever a gesture is recognized, the gesture recognizer task also provides the score with which the gesture was recognized. This score lies between 0 and 1.0 and shows the accuracy of the detection. When adjusting the values of the thresholds in the settings page, the user can determine what precision is required for the detection. The higher the threshold values, the more precise the gesture has to be detected to be recognized as such. For poor lighting conditions or for people who find some gestures more complicated to execute, the threshold values should be reduced so that the recognition is somewhat easier.

To adjust the values, each threshold is provided with a "+" and a "-" button, which increments or decrements the values by 0.1. These values are tied to the same ViewModel as the cooking mode with the gesture recognizer, such that the recognizer task has direct access to the current value. Using these settings, the user can adjust the gesture recognition according to their needs and environment.

3.5.6 Adding new Recipes

One of the main ideas of the designed application was to propose a digital cookbook where users can add all of their favorite recipes, a feature that most popular cooking applications do not offer, as was shown in the comparison chapter 2.1 and formalized in requirement RCP_ADD_101. This can be done by using the floating action button

on the homepage of the application and filling in the required and optional information. The page to add recipes (3.6) uses a *ViewPager* and *TabLayout* to have three tabs and to be able to swipe between these three tabs.

The *Overview* tab is responsible for requesting the recipe title and recipe image. To get the recipe title, an *EditText* is initialized with an *OnEditorActionListener* and saves the entered text. To get the image, the user must click on the ImageView requesting the image, whereupon the user's photo gallery is opened. After selecting an image, the photo is uploaded to the ImageView and set as the recipe image. Additionally, the image is saved to the internal storage to retrieve it every time it is needed. The code which does that can be seen in Listing 3.5

```
// Method to save image to internal storage
    private void saveImageToInternalStorage(Bitmap image) {
        ContextWrapper cw = new ContextWrapper(getContext());
        File directory = cw.getDir("imageDir", Context.
   MODE_PRIVATE);
        String fileName = "image_" + System.currentTimeMillis() +
   ".png";
        File path = new File(directory, fileName);

        FileOutputStream fos = null;
        try {
            fos = new FileOutputStream(path);
            image.compress(Bitmap.CompressFormat.JPEG, 100, fos);
        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            try {
                if (fos != null) {
                    fos.close();
                }
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
        recipeViewModel.setImagePath(path.getAbsolutePath());
    }
```

Listing 3.5: Saving Recipe Image to Internal Storage

The ingredients and the steps tab work very similarly to each other. Both allow the user to add one item at a time, which will then in turn be displayed and saved as a list. The

user also has the option to edit or delete an element from the list, in case they made an error.

To exit the add recipe page, the user has two options. If they have filled in all the necessary information and want to save the recipe, the "Save Recipe" button is to be used. The newly created recipe will then be saved in the recipe dataset and be visible on the cookbook page. The user can then follow this recipe step-by-step with the hands-free cooking mode. The other option for the user is to close the page using the back button at the top left or using the back key of the phone. In both cases, the user will be asked if they want to leave the page and discard the changes or stay on the page and continue with the recipe. This ensures that no progress is unnecessarily lost.

3.5.7 App Tutorial

The app tutorial is responsible for presenting and explaining the application to the user, especially when accessed for the first time, as gesture control is not something many applications possess, and therefore users usually do not know immediately how it works. Also, the applied gestures and their corresponding actions are individually programmed for this application.

The tutorial page is programmed to be the page that is displayed when the user opens the app. However, every time the app is opened, the code checks if the tutorial has already been shown to the user before. The first time the user opens it, this check is negative, resulting in the tutorial being shown to the user. This information is then saved to the *SharedPreferences*, which is a simple key-value storage mechanism used to save small amounts of primitive data within the application, saying that the tutorial has already been shown to the user. Every time the app is opened, the SharedPreferences are accessed to perform the check, and the tutorial is not shown again. Listing 3.6 shows the necessary methods for checking and saving the data to SharedPreferences.

```
    // When this activity is about to be launched, check if it was
    opened before
    if (restorePrefData()) {
        Intent intent = new Intent(getApplicationContext(),
MainActivity.class);
        startActivity(intent);
        finish();
    }

    // Checks if the tutorial has been opened before by reading
from shared preferences
```

```
private boolean restorePrefData() {
    SharedPreferences pref = getApplicationContext().
getSharedPreferences("myPrefs", MODE_PRIVATE);
    return pref.getBoolean("isTutorialOpened", false);
}

// Saves value in shared preferences indicating that the
tutorial has been opened
private void savePrefsData() {
    SharedPreferences pref = getApplicationContext().
getSharedPreferences("myPrefs", MODE_PRIVATE);
    SharedPreferences.Editor editor = pref.edit();
    editor.putBoolean("isTutorialOpened", true);
    editor.apply();
}
```

LISTING 3.6: Check for Showing the App Tutorial

In the tutorial, the user is welcomed to the application, and it is explained to them that the application serves as a digital cookbook, where they can add their recipes. Also, the cooking mode and the hands-free control are presented, for which the application asks for the camera permission, meaning that it can access the camera while the user is using the application.

The user is then guided through an explanation of each gesture and its associated action. To continue in this part of the tutorial, the user has to do the indicated gesture for every step. This will allow the user to get familiar with the gestures and help him internalize the functionality. After completing this part, the tutorial finishes and the user is ready to use the app. If the user is already familiar with the application, he can use the skip button, which will automatically lead him to the last page of the tutorial. The app tutorial fulfills requirement APP_TUT_101, which is the last of the functional requirements, and thereby all requirements are satisfied.

3.6 Process Flow

As mentioned previously, the developed application aims to allow users to follow recipes easily during the cooking process without having to touch the phone. This section describes the whole flow of how to use the application during that process.

Step 1 The user chooses a recipe that they want to cook from the recipe collection by clicking on it.

Step 2 The user sees the recipe with its description and ingredients. Now is the time to prepare all the ingredients for the cooking process.

Step 3 After all the preparations, the cooking mode starts after the user clicks on the "Start Cooking" button.

Step 4 The user works his way through the recipe with the hands-free cooking mode.

Step 5 The gesture recognizer module tracks the gestures from the user using the live stream from the front camera. Gestures are recognized and the name and score are displayed below the camera view.

Step 6 Each gesture is associated with an action, e.g. the thumbs up gesture forwards the user to the next step.

Step 7 The recipe is closed with the closed fist gesture and the user returns to the recipe overview.

Chapter 4

Experiments

This chapter will describe how the presented application was verified and validated both technically as well as practically. This process is of high importance to ensure the application's functionality, usability, and reliability and involves a series of tests and evaluations designed to confirm that the application meets its specified requirements and performs satisfactorily in real-world scenarios. Technical verification focuses on evaluating the code, architecture, and individual components of the application to verify their correctness and efficiency. Practical validation, on the other hand, includes user testing and simulations under real-life conditions to ensure that the application provides a seamless and intuitive user experience. This extensive approach aims to identify and resolve any issues to ensure that the application is robust, user-friendly, and ready for use.

4.1 Technical Verification

The goal of the technical verification is to verify and ensure that each component of the system functions correctly and fulfills the specified requirements. It is therefore of crucial importance in the development life cycle of the application and falls into phase four of the waterfall model. This phase includes a systematic review of the code, architecture, and individual modules of the application to establish their correctness, performance, and compliance with coding standards.

4.1.1 Code Review

One of the first steps of the technical verification process is to conduct a thorough code review to ensure it is complete and consistent. This involves reading and analyzing

the complete source code to identify potential bugs or errors and adherence to common coding practices and standards. The used IDE for the development, Android Studio, offers various tools, that help in analyzing the code for errors or potential improvements. The code review also serves to verify that the code is well-documented and easy to read for external individuals.

4.1.2 Unit Testing

Unit testing is a fundamental aspect of all kinds of software testing and forms the foundation of technical verification, testing individual components or functions of the software application in isolation and ensuring that they work as expected. In Android development, testing tools like JUnit and Mockito are used to facilitate the creation and execution of unit tests. Listing 4.1 shows an example of such a unit test to create a new recipe. The constructor is tested to see if the recipe is correctly created with the right title, description, image, tags, ingredients, and steps. This test is executed independently from all the other code and has therefore no effect from or on it. Advantages of conducting unit testing include early detection of issues, improved code quality, and better documentation.

```
@Test
public void testRecipeConstructor_withImageResourceId() {
    Recipe recipe = new Recipe("Cheese Fondue", "The best
cheese recipe.",
    R.drawable.recipe_salade_nicoise, Arrays.asList("Cheese",
"Swiss"),
    Arrays.asList("Comte", "Gruyere"),
    Arrays.asList(new Step("Melt the cheese.", 10), new Step("
Eat with bread.", 20)));
    assertEquals("Cheese Fondue", recipe.getRecipeTitle());
    assertEquals("The best cheese recipe.", recipe.
getRecipeDescription());
    assertEquals((Integer) R.drawable.recipe_salade_nicoise,
recipe.getImageResourceId());
    assertNull(recipe.getImagePath());
    assertEquals(Arrays.asList("Cheese", "Swiss"), recipe.
getTags());
    assertEquals(Arrays.asList("Comte", "Gruyere"), recipe.
getRecipeIngredients());
    assertEquals(2, recipe.getSteps().size());
}
```

Listing 4.1: Unit Test for a Recipe Creation

4.1.3 Integration Testing

After verifying the individual components through unit testing, integration testing is performed to ensure that these components work together correctly. This involves testing the interactions and data exchange between different components and modules of the application, such as the integration between the gesture recognition module and the cooking mode. The main goal of integration testing is to identify any issues or bugs that may arise when different components are combined and interact with each other. Integration testing helps to identify and resolve integration issues early in the development cycle to avoid more severe problems later on, which could result in more costly to resolve.

There are different approaches to perform integration testing, such as sandwich or top-down testing, however, for this application, the bottom-up method was chosen. This means that lower-level modules were evaluated first, followed by higher-level modules gradually. In the case of the proposed application, for instance, the gesture module was tested first, checking if the gestures were recognized correctly, and afterward, the gestures module was tested with the cooking mode to verify that the correct actions were performed. This method has the advantages that several disjoint subsystems can be tested simultaneously, test conditions are easy to create, and the test results are easy to observe and evaluate.

4.1.4 Conclusion

Technical verification is an important and complete testing process that contains various testing and verification techniques. The goal is to ensure that the developed application is bug-free, robust, and works correctly. By systematically reviewing the code, performing rigorous testing, and following best coding practices, it can be ensured that the application meets technical specifications and is ready for real-world validation and deployment. This thorough approach to technical review helps to minimize the risk of errors, improve performance, and deliver a high-quality application to users.

4.2 Practical Validation

As mentioned in the introductory chapter of this paper, the validation of the proposed application in real-world scenarios and with real test users posed one of the main objectives of this project. Practical validation is one of the most crucial phases in the development cycle of any application, especially for mobile applications, which are interactive

and user-oriented. This is certainly the case with the presented application, which proposes a gesture-recognizing system to follow cooking recipes. This phase ensures that the application not only works technically correctly but also meets user expectations, fulfills all defined requirements, and proves itself in real-life scenarios.

User testing is the backbone of practical validation. Real users are testing and interacting with the application, checking every functionality, verifying if there are any issues or bugs, and providing feedback on the application and their experience. This process can be broken down into several stages:

4.2.1 Recruitment of Test Participants

The application's target audience is home cooking enthusiasts, that are moderately to highly technology-affine. This includes mothers and fathers who enjoy cooking different recipes or young people who like to try new food. For effective user testing, the group of participants should be as diverse as possible to reflect the entire target audience. This group should include individuals with varying levels of technical proficiency, familiarity with mobile apps, and interest in cooking.

4.2.2 Test Planning and Design

To conduct the user tests effectively and efficiently, a test plan should be developed, where the objectives, methodology, tasks, and success and failure criteria are determined. The objectives determine what should be achieved with these user tests, the methodology describes how the tests are conducted, and the tasks cover all the functionalities of the application, such as navigating through a recipe with gestures or adding a new one. Finally, the success and failure criteria illustrate whether the tests were successful or not and what is left to improve.

To have a complete and representative practical validation, two different user tests were conducted. The first one was conducted in a smaller circle of **four** test users, which followed closely the development process and tested each new functionality that was added during the development as soon as it was ready. For the second test, the group of test users was expanded to **15** test users, having the goal for this test to analyze the whole application as soon as it was complete and have a more representative result. Each test design will be explained in more detail in the following.

Test 1: The objective of the first user test is to evaluate one-by-one if all the functional requirements are implemented and work correctly, and to remark if some functionalities, that were not defined in the requirements but are important for the user, are missing.

The methodology for this test was an iterative approach, not in total accordance with the waterfall model that was the main development cycle, however, it was deemed practical for this kind of test. At regular intervals, the users were asked informally to test and evaluate the currently available functionalities. Each test was used to try out the newly added functionalities and give feedback about it. The test was successful if the users deemed that the functionality worked well and no major changes had to be made. On the contrary, if the functionality did not work or if any changes or additions had to be made, the result was negative. However, the negatively resulting tests helped in figuring out what was going wrong and which features had to be added as extras.

Test 2: This test was conducted to evaluate the overall design, usability, and experience with the application. The users were asked to download the application on their phones and test each functionality that the app has to offer. Everything was supposed to be tested and remarks recorded. After the test was conducted, the users were requested to fill out a post-test questionnaire. This questionnaire reflects how the users perceive the usability of the application as a whole and is subject to the *peak effect*, which is the most intense and last parts of the experience that impact participants' recollections and evaluations the most.

One of the most well-known post-test questionnaires used in user experience research is the system usability scale (SUS) [26]. This test already existed since the 1980s, is widely employed, and has been demonstrated to be valid and reliable. Therefore, it was chosen to evaluate the user experience for this application. The questionnaire consists of ten Likert-scale questions, meaning that the answers can range on a scale from one to five, one expressing total disagreement and five total agreement. The SUS produces a score between zero and 100, however the score is not equivalent to a percentage score. According to Bangor et al. [26], an average score of 50.9 would be interpreted as *OK* or *Fair*, an average of 71.4 as *Good*, and an average of 35.7 as *Poor*. Some examples of popular software or websites include: "Excel" with a score of 56.5, "Word" with 76.2, "Amazon" with 81.8, and "Google Search" which reached an average score of 92.7, which is considered *Best imaginable*. According to this scale and these average ratings, achieving an average score of 70 or higher would be considered a success for the presented application, a score below 55 a failure, and a score between 55 and 69 neutral. Table 4.1 shows the questions of the SUS questionnaire and Equation 4.1 how the final SUS score is calculated.

Number	Question
1	I think that I would like to use this system frequently.
2	I found the system unnecessarily complex.
3	I thought the system was easy to use.
4	I think that I would need the support of a technical person to be able to use this system.
5	I found the various functions in this system were well integrated.
6	I thought there was too much inconsistency in this system.
7	I would imagine that most people would learn to use this system very quickly.
8	I found the system very cumbersome to use.
9	I felt very confident using the system.
10	I needed to learn a lot of things before I could get going with this system.

TABLE 4.1: System Usability Scale Questions

$$\mathrm{SUS} = 2.5 \left(20 + \sum \mathrm{SUS}_{01,03,05,07,09} - \sum \mathrm{SUS}_{02,04,06,08,10}\right) \tag{4.1}$$

Table 4.2 shows an example of how a user could respond to the questions and the calculated final score. According to this example, the score would be 80, which represents an adjective rating of *Good*.

Question	1	2	3	4	5	6	7	8	9	10	**Final Score**
Answer	4	2	4	1	3	2	5	1	4	2	**80**

TABLE 4.2: Example Answers for SUS Questionnaire

4.2.3 Test Execution

During the test sessions, some participants are observed as they interact with the app. They were asked to provide instant feedback on how they perceive the application, the functionalities, the difficulties they encounter, and which kind of additional features they are missing. This was especially done during test 1. For test 2, users were left alone to test the application extensively, trying to figure out problems on their own and finally completing the questionnaire without external influence.

4.2.4 Data Analysis and Evaluation

The final step for user testing is the data collection, analysis, and evaluation. The data collected from the user testing session should include both quantitative metrics, such as the numerical responses from the SUS questionnaire, and qualitative feedback, which includes verbal and written comments. The data analysis helps in evaluating the application and identifying potential for improvement.

4.2.4.1 Results of the SUS questionnaire

In total, 15 test users conducted test 2 and completed the SUS questionnaire. In addition to the 10 usual questions mentioned above, the users were asked about their age, gender, ability to use a smartphone, and familiarity with the Android operating system. These characteristics could potentially affect the outcome of the results, which is why it is important to analyze those factors as well.

All test users were aged between 13 and 64 years, with the majority of them being 18-24 or 45-54 years old. The gender split was almost exactly 50/50 with eight women and seven men and all users consider themselves good or experts at handling smartphones. Regarding familiarity with the Android operating system, the group is not homogeneous. Three testers rated their familiarity with three points, five other users with four, and the remaining with five. This test group represents well the main target audience of the application, however, additional and more diverse test users could bring even more validity to the test results.

Question	1	2	3	4	5	6	7	8	9	10	**Final Score**
Average Answer	3.9	1.6	4.5	1.1	4.1	1.6	4.3	1.3	4.1	1.4	**84.8**

TABLE 4.3: Average Answers from Test Users for SUS Questionnaire

Table 4.3 shows the average response score for each question and the total average. In general, test users responded well to the system and rated the application with an average score of 84.8 out of 100. Considering a score of 70 or higher was the self-imposed goal for success, it can be said that the test was favorable and the target was clearly exceeded. According to the scale defined by Bangor and his colleagues [26], 85.5 is the exact score to achieve to obtain an adjective rating of *Excellent*. After evaluation of the results, it can be asserted that the conducted project fulfills its objectives of providing an application that meets usability standards and offers a pleasant user experience to the target audience.

Also, test users were asked to qualify their overall experience with the application, how likely they were to recommend the application to others, and to provide additional written feedback or comments. The results of these questions were as follows: The overall experience was qualified on average with a 4.0, and the likelihood of recommending the application was rated with 4.5 out of 5. According to the analysis of the answers to these questions, users were generally satisfied with the proposed application, however, much additional feedback was given and numerous requests for additional features were made.

The principal objective of the developed application is to propose a new kind of interactive cooking mode for recipe applications: allowing the user to navigate through the recipe hands-free, meaning without having to touch the phone; instead, the navigation is done through gesture control. The overall consensus regarding this feature was that it works well and is very easy to learn, especially after going through a tutorial that explains each gesture. Some comments were made about the closed fist gesture for two seconds for closing the recipe, saying that the holding time was not very clear, so some adjustments might have to be made.

Regarding additional features, test users wanted the possibility to share recipes with others, voice control in addition to gesture control, and more specific saving options, such as creating collections or marking a recipe as a favorite. Also, users saw the potential for more "social settings", e.g. having user accounts, following other users, and publishing recipes for everybody.

4.2.4.2 Conclusion

Practical validation is a critical step during the testing process, which helps to identify user problems and additional needs and to prepare the application for deployment. In general, the user feedback has been very positive and many users see the usefulness of the presented application. Most users are ready to adopt this new application, however, they also see great potential for further development and ask for more features.

Chapter 5

Budget

The budget chapter aims to estimate the approximate costs that the development of the presented project would entail. This estimation includes various aspects of the project, including hardware, software, development resources, and testing. By providing a detailed breakdown of these costs, the budget chapter will offer an overview of the financial requirements needed to complete this kind of project in a company setting.

First of all, the hardware resources are to be estimated. Two resources were needed: a computer and a mobile phone with an Android operating system. The computer that was used to program the application was a MacBook Air laptop from 2020 with an Apple M1 chip, eight GB of memory, and 512GB SSD storage. This kind of laptop nowadays has a value of around **675€** [35], however, these kinds of specifications are not fully necessary for the development, meaning that a lower-price computer could also be used. The second hardware device was a mobile phone on which the app can run. In this case, a BQ Aquaris V from 2017 was used, which runs with the Android operating system, has a 5.2-inch screen, a front camera with 8 megapixels resolution, and has a price of about **210€** [36].

The software resources required for this project include Android Studio, the programming language Java, and the gesture recognition framework MediaPipe. All of these resources are open-source, meaning that they are accessible to everybody and can be downloaded for free. Therefore, no software costs were incurred.

The next part comprises the development resources. This includes first and foremost an Android developer to develop the application. Considering that a junior Android developer in Spain should cost about **16€**/hour [37] and that 300 hours of working time were invested into the project, the developer would cost in total around **4800€**. In addition to this developer, a project manager could also be needed. Considering that he

would cost about **25€**/hour [38] and invest 100 hours into the project, this would add another **2500€** to the project costs. Adding these numbers up, the software resources cost would amount to **7300€**.

Testing is also an additional cost that has to be considered in the budget calculation. This encompasses unit testing, integration testing, system testing, and conducting the system usability scale test. Considering that this would take up another 50 hours for the developer, this would add another **800€** to the project budget.

In total, considering all the costs for hardware, software, development resources, and testing, the total for the budget amounts to **8985€**. It is important to note that this amount only represents an estimation of what the presented project would cost if developed in a company setting. Furthermore, costs for maintenance would also have to be considered in the future.

Chapter 6

Conclusions

In conclusion, this work presented a mobile recipe collection application with an interactive hands-free cooking mode for the Android operating system. For this purpose, a new recipe collection application was designed, developed, and validated to solve the problem of using a smartphone when having dirty hands whilst cooking. After giving a brief introduction on IoT in the kitchen and existing recipe applications, and highlighting the importance of usability and accessibility for existing and new solutions, the project motivation and objectives were outlined. The main goal of the project was to plan, develop, and validate an intuitive and easy-to-use application, that integrates a gesture recognition algorithm and enables navigating through the cooking mode only using gestures.

In this work, it was shown that this kind of application is an innovation in the cooking applications market. After presenting and comparing popular existing applications in this field, a brief introduction to computer vision, which is the base of gesture recognition, was given, and existing gesture recognition systems in the food area and for mobile phones were introduced. In that chapter, MediaPipe Solutions by Google was presented for the first time in this paper and explained in more detail, as it is the framework that was finally integrated into the developed application.

After outlining the state of the art, the development process was presented. This process was conducted following the waterfall software development life cycle model, which proposes a step-by-step approach to software development. Starting with the planning and requirements definition, where the main features were defined, and followed by the design phase, where the gesture recognition framework MediaPipe was chosen, as it presents the best trade-off between performance and complexity for this application. Following the design phase came the development phase, which consisted of programming the application and integrating the gesture recognition module into it. Finally,

the technical verification and practical validation were conducted, where 15 test users responded to the SUS post-test questionnaire. The average score of this questionnaire was 84.8, which comes very close to the equivalent of the grade *Excellent*, showing hereby that the system as a whole and the gesture control are practical to use. In addition, the comments and feedback showed that the users would like to adopt this application for their cooking. Considering these results, it can be affirmed that the application was designed, developed, and validated successfully. However, some limitations remain and further development is planned for the future.

6.1 Limitations and Future Work

Even though the presented application received good reviews and obtained an almost excellent score on the SUS test, the application has its limitations and potential for future improvement. Regarding the gestures, only seven different ones are available, even though eight different actions can be performed. Having different gestures for starting the timer and advancing to the next step would reduce ambiguities. For this purpose, another gesture model would have to be used or a customized one trained and integrated into the module. In addition, the application is currently only designed for mobile phones. During interviews with some of the testers, it became apparent that tablets might be even more popular to use as a digital cookbook. Therefore, the application should also be designed for tablets. Following on from this, the application is currently only available for the Android operating system. Although this operating system represents the largest market share in the smartphone market, developing the application also for iOS could help with reaching more users.

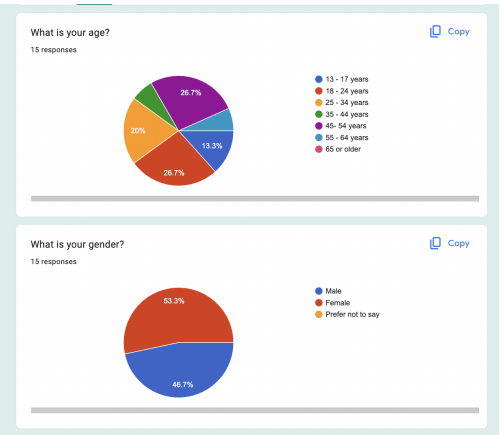
Finally, users have expressed their wishes for more functionalities. Especially requested by many users was to design the app as a social network, where users can create an account, follow other users, share recipes with others, and publish their recipes to everybody. Since the application is also supposed to be a personalized collection, users should be able to easily import recipes to their collection from other users or even different sources such as cooking websites. Recipe Markup Language (RecipeML) could be of great help for this as it serves as a format for representing recipes on computers and is written in the popular Extensible Markup Language (XML).

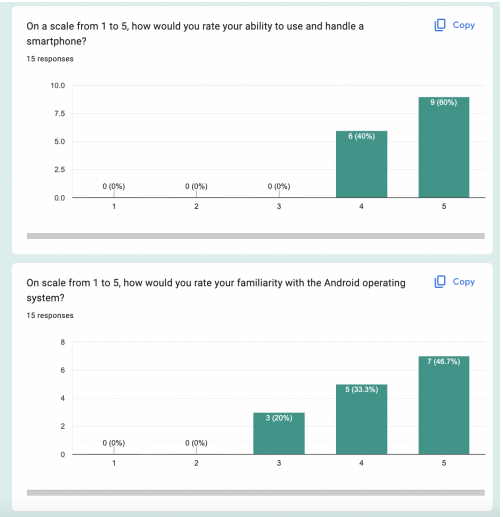
As can be seen, numerous additional features could be added to the presented application. And finally, the last step would be to publish the application to the Google Play Store to make it available to every Android user.

As a final note, the code used for this thesis can be found on Github [39].

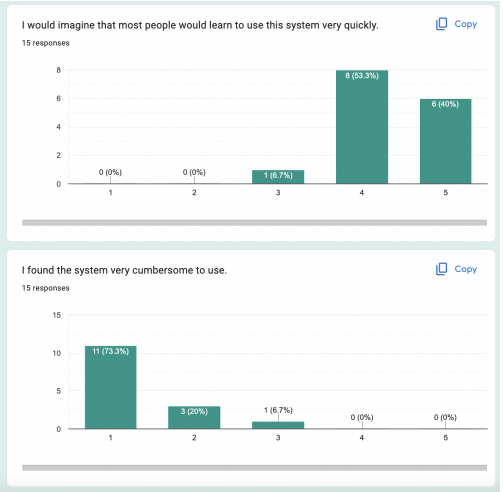
Appendix A

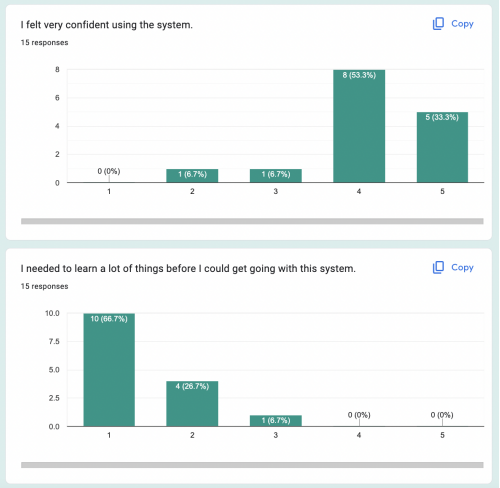
SUS Test Answers

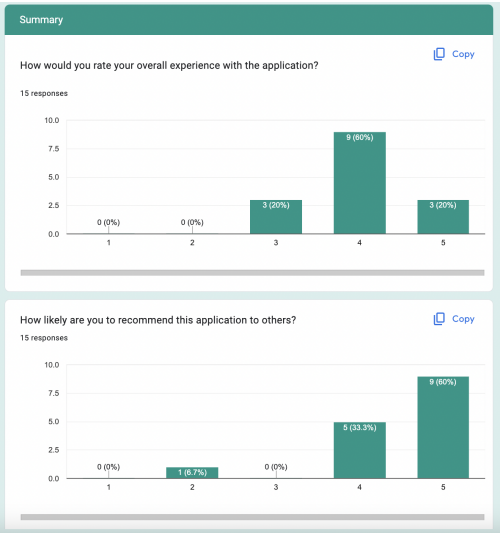





I found the various functions in this system were well integrated.
Copy
15 responses
6
4
2
0
0 (0%)
0 (0%)
5 (33.3%)
4 (26.7%)
6 (40%)
1
2
3
4
5
I thought there was too much inconsistency in this system.
Copy
15 responses
10.0
7.5
5.0
2.5
0.0
10 (66.7%)
2 (13.3%)
2 (13.3%)
1 (6.7%)
0 (0%)
1
2
3
4
5


I would imagine that most people would learn to use this system very quickly.
Copy
15 responses
8
6
4
2
0
0 (0%)
0 (0%)
1 (6.7%)
8 (53.3%)
6 (40%)
1
2
3
4
5
I found the system very cumbersome to use.
Copy
15 responses
15
10
5
0
11 (73.3%)
3 (20%)
1 (6.7%)
0 (0%)
0 (0%)
1
2
3
4
5


I felt very confident using the system.
Copy
15 responses
8
6
4
2
0
0 (0%)
1 (6.7%)
1 (6.7%)
8 (53.3%)
5 (33.3%)
1
2
3
4
5
I needed to learn a lot of things before I could get going with this system.
Copy
15 responses
10.0
7.5
5.0
2.5
0.0
10 (66.7%)
4 (26.7%)
1 (6.7%)
0 (0%)
0 (0%)
1
2
3
4
5


Summary
How would you rate your overall experience with the application?
Copy
15 responses
10.0
7.5
5.0
2.5
0.0
0 (0%)
0 (0%)
3 (20%)
9 (60%)
3 (20%)
1
2
3
4
5
How likely are you to recommend this application to others?
Copy
15 responses
10.0
7.5
5.0
2.5
0.0
0 (0%)
1 (6.7%)
0 (0%)
5 (33.3%)
9 (60%)
1
2
3
4
5

Bibliography

[1] Atsushi Hashimoto, Naoyuki Mori, Takuya Funatomi, Yoko Yamakata, Koh Kakusho, and Michihiko Minoh. Smart kitchen: A user centric cooking support system. In *Proceedings of IPMU*, volume 8, pages 848–854. Citeseer, 2008.

[2] Matteo Zallio, Paula Kelly, Barry Cryan, and Damon Berry. A co-design approach to develop a smart cooking appliance. applying a domain specific language for a community supported appliance. *arXiv preprint arXiv:2101.08886*, 2021.

[3] Jyotir Moy Chatterjee, Raghvendra Kumar, Manju Khari, Dao Thi Hung, and Dac-Nhuong Le. Internet of things based system for smart kitchen. *International Journal of Engineering and Manufacturing*, 8(4):29, 2018.

[4] New York Times. "NYT Cooking". *The New York Times Company*. [Google Play Store]. Available: https://play.google.com/store/search?q=nyt+cooking&c=apps. Accessed: Jul 11, 2024.

[5] Yummly. "Yummly Recipes & Cooking Tools". *Yummly*. [Google Play Store]. Available: https://play.google.com/store/apps/details?id=com.yummly.android. Accessed: Jul 11, 2024.

[6] BuzzFeed. "Tasty". *BuzzFeed*. [Google Play Store]. Available: https://play.google.com/store/apps/details?id=com.buzzfeed.tasty. Accessed: Jul 11, 2024.

[7] Syafiqah Elma Shaharuddin and Noraini Ibrahim. Self-learning cooking application. *Applied Information Technology And Computer Science*, 2(2):839–855, 2021.

[8] Takuma Maruyama, Yoshiyuki Kawano, and Keiji Yanai. Real-time mobile recipe recommendation system using food ingredient recognition. In *Proceedings of the 2nd ACM international workshop on Interactive multimedia on mobile and portable devices*, pages 27–34, 2012.

[9] C. Mauran. "10 apps you need to succeed in the kitchen". *Mashable*, Sep. 01, 2021. [Online]. Available: https://mashable.com/article/best-cooking-apps.

[10] Condé Nast Digital. "Epicurious". *Condé Nast Digital*. [Apple App Store]. Available: https://apps.apple.com/us/app/epicurious/id312101965. Accessed: Jul 11, 2024.

[11] KptnCook GmbH. "KptnCook Meal Plan & Recipes". *KptnCook GmbH*. [Google Play Store]. Available: https://play.google.com/store/apps/details?id=com.kptncook.app.kptncook. Accessed: Jul 11, 2024.

[12] IBM. "What is Computer Vision?". *IBM*, 2019. [Online]. Available: https://www.ibm.com/topics/computer-vision.

[13] Richard Szeliski. *Computer vision: algorithms and applications*. Springer Nature, 2022.

[14] Vijay Kakani, Van Huan Nguyen, Basivi Praveen Kumar, Hakil Kim, and Visweswara Rao Pasupuleti. A critical review on computer vision and artificial intelligence in food industry. *Journal of Agriculture and Food Research*, 2:100033, 2020.

[15] Google. "MediaPipe Solutions guide". *Google Developers*, . [Online]. Available: https://ai.google.dev/edge/mediapipe/solutions/guide. Accessed: Jul 02, 2024.

[16] OpenCV. "OpenCV - Open Computer Vision Library". *OpenCV*, Jul. 10, 2024. [Online]. Available: https://opencv.org/.

[17] Mahamkali Naveenkumar and Ayyasamy Vadivel. Opencv for computer vision applications. In *Proceedings of national conference on big data and cloud computing (NCBDC'15)*, pages 52–56, 2015.

[18] Google. "ML Kit | Google for Developers". *Google Developers*, . [Online]. Available: https://developers.google.com/ml-kit/. Accessed: Jul 02, 2024.

[19] TensorFlow. "TensorFlow Lite: ML for Mobile and Edge Devices". *TensorFlow*. [Online]. Available: https://www.tensorflow.org/lite. Accessed: Jul 02, 2024.

[20] L. Boscher. "Vision-Based Hand and Body Tracking Solutions". *ManoMotion*, Jun. 11, 2024. [Online]. Available: https://www.tensorflow.org/lite.

[21] Ankita Saxena, Deepak Kumar Jain, and Ananya Singhal. Hand gesture recognition using an android device. In *2014 fourth international conference on communication systems and network technologies*, pages 819–822. IEEE, 2014.

[22] Houssem Lahiani, Mohamed Elleuch, and Monji Kherallah. Real time hand gesture recognition system for android devices. In *2015 15th International Conference on Intelligent Systems Design and Applications (ISDA)*, pages 591–596. IEEE, 2015.

[23] Twinkle Sharma, Sachin Kumar, Naveen Yadav, Kritika Sharma, and Piyush Bhardwaj. Air-swipe gesture recognition using opencv in android devices. In *2017 international conference on algorithms, methodology, models and applications in emerging technologies (ICAMMAET)*, pages 1–6. IEEE, 2017.

[24] Moh Harris, Ali Suryaperdana Agoes, et al. Applying hand gesture recognition for user guide application using mediapipe. In *2nd International Seminar of Science and Applied Technology (ISSAT 2021)*, pages 101–108. Atlantis Press, 2021.

[25] Kai Petersen, Claes Wohlin, and Dejan Baca. The waterfall model in large-scale development. In *Product-Focused Software Process Improvement: 10th International Conference, PROFES 2009, Oulu, Finland, June 15-17, 2009. Proceedings 10*, pages 386–400. Springer, 2009.

[26] Aaron Bangor, Philip T Kortum, and James T Miller. An empirical evaluation of the system usability scale. *Intl. Journal of Human–Computer Interaction*, 24(6): 574–594, 2008.

[27] Google. "Android Apps on Google Play". *play.google.com*, . [Online]. Available: https://play.google.com/store/games. Accessed: Jul 04, 2024.

[28] Android Developers. "Build accessible apps : App quality : Android Developers". *Android Developers*, . [Online]. Available: https://play.google.com/store/games. Accessed: Jul 05, 2024.

[29] Android Developers. "Download Android Studio & App Tools". *Android Developers*, . [Online]. Available: https://developer.android.com/studio. Accessed: Jul 05, 2024.

[30] Android Developers. "Firebase". *Android Developers*, . [Online]. Available: https://firebase.google.com/. Accessed: Jul 05, 2024.

[31] Chris Anderson. The model-view-viewmodel (mvvm) design pattern. In *Pro Business Applications with Silverlight 5*, pages 461–499. Springer, 2012.

[32] Artem Syromiatnikov and Danny Weyns. A journey through the land of model-view-design patterns. In *2014 IEEE/IFIP Conference on Software Architecture*, pages 21–30. IEEE, 2014.

[33] Android Developers. "Data Binding Library : Android Developers". *Android Developers*, 2019. [Online]. Available: https://play.google.com/store/games.

[34] Google. "Gesture recognition task guide | Google AI Edge | Google for Developers". *Google for Developers*, . [Online]. Available: https://ai.google.dev/edge/

mediapipe/solutions/vision/gesture_recognizer#models. Accessed: Jul 06, 2024.

[35] Best Buy. "Apple MacBook Air 13.3" Pre-Owned M1 chip 8GB Memory 8 GPU 512GB SSD (2020) Space Gray MGN73LL/A". *Best Buy*. [Online]. Available: https://shorturl.at/YNuXm. Accessed: Jul 05, 2024.

[36] gsmarena. "BQ Aquaris V - Full phone specifications". *gsmarena*, Jun. 25, 2024. [Online]. Available: https://www.gsmarena.com/bq_aquaris_v-9034.php.

[37] Glassdoor. "Salary: Junior Android Developer in Barcelona, Spain 2024 | Glassdoor". *Glassdoor*, . [Online]. Available: https://www.glassdoor.com/Salaries/barcelona-junior-android-developer-salary-SRCH_IL.0,9_IM1015_KO10,34.htm. Accessed: Jul 06, 2024.

[38] Glassdoor. "Salary: Project Manager in Spain 2024 | Glassdoor". *Glassdoor*, . [Online]. Available: https://www.glassdoor.com/Salaries/spain-project-manager-salary-SRCH_IL.0,5_IN219_KO6,21.htm. Accessed: Jul 06, 2024.

[39] Quentin Mathieu. "GitHub - Radfahrer00/PanPalApp". *Github*. [Online]. Available: https://github.com/Radfahrer00/PanPalApp. Accessed: Jul 11, 2024.