Czech University of Life Sciences Prague

Faculty of Economics and Management

Department of Information Engineering

Diploma Thesis

Development of Android mobile application using Java programming language

Gleb Dmitrievsky

CZECH UNIVERSITY OF LIFE SCIENCES PRAGUE

Faculty of Economics and Management

DIPLOMA THESIS ASSIGNMENT

Gleb Dmitrievsky

Systems Engineering and Informatics
Informatics

Thesis title

Development of Android mobile application using Java programming language

Objectives of thesis

The main objective of the thesis is to explore the process of the mobile application development using recent Android platform and to compare different approaches and alternatives in all stages of the development process.

Methodology

In the theoretical part of the thesis, investigation of the steps of creating a mobile application using professional literature will be done. Then the different technical aspects of programming will be analyzed with a further choice of the right structure, defining principles, and a suitable workflow. Moreover, the employment of theoretical knowledge will be done in realisation of application. An application will be created using the selected best practices of development and also maintenance. The final application will be able to track expiry dates of the products that a user has bought, notify a user about the oncoming expiration date, and even help a user to utilize those products.

The proposed extent of the thesis

80 – 120 pages

Keywords

Java programming; OOP; Android; mobile application development; database

Recommended information sources

GRIFFITHS, Dawn and David GRIFFITHS. Head First Android Development. 2nd Edition. Newton: O'Reilly Media, 2017. ISBN 9781491974056.

SCHILDT, Herbert. Java: The Complete Reference. 11th Edition. New York: McGraw-Hill, 2018. ISBN 9781260440249.

WEISFELD, Matt. The object-oriented thought process. 5th Edition. Boston: Addison-Wesley, 2019. ISBN 9780135182130.

Expected date of thesis defence

2020/21 SS – FEM

The Diploma Thesis Supervisor

doc. Ing. Vojtěch Merunka, Ph.D.

Supervising department

Department of Information Engineering

Electronic approval: 19. 11. 2020	Electronic approval: 19. 11. 2020
Ing. Martin Pelikán, Ph.D.	**Ing. Martin Pelikán, Ph.D.**
Head of department	Dean

Prague on 31. 03. 2021

Declaration

I declare that I have worked on my diploma thesis titled "Development of Android mobile application using Java programming language" by myself and I have used only the sources mentioned at the end of the thesis. As the author of the diploma thesis, I declare that the thesis does not break copyrights.

In Prague on 31.03.2021 ______________________________

Acknowledgement

I would like to thank my supervisor doc. Ing. Vojtěch Merunka, Ph.D. for his guidance and support during my work on this thesis. Additionally, I would like to thank my family for all their support and belief in me.

Development of Android mobile application using Java programming language

Abstract

The food wasting problem is unlikely to be one of the most discussed problems. But the fact is that about one third of the food produced globally for human consumption is wasted or lost. It has a huge negative economic impact on the income of food producers and customers. The population is rapidly growing and the demand of natural resources does not decrease. People can consume nature's resources in different ways, for example, by producing electronics. As to now, there are about 3 billion smartphones today. The Android operating system is installed on almost 2.5 billion mobile devices. Why not try to use smartphones to help us reduce the food wasting problem? This thesis is focused on the development of the FoodNoWaster Android application which should help to solve the food wasting problem. The application was written using the Java programming language for core logic. The core of the application was written with the OOP approach. The User Interface of the FoodNoWaster application was built with XML layout files. The FoodNoWaster application stores the user’s data in the SQLite database. When the food item is approaching its expiration date, the user will be notified about that. The result of this thesis is ready to use the FoodNoWaster application, which is being used by the author and his relatives.

Keywords: Java programming, OOP, Android, mobile application development, SQLite database, food waste problem.

Vývoj Android aplikací v programovacím jazyku Java

Abstrakt

Pravděpodobně problém plýtvání potravinami není jedním z nejdiskutovanějších problémů. Faktem však je, že třetina potravin vyrobených globálně pro lidskou spotřebu je zbytečná nebo ztracená. Tento problém má obrovský negativní ekonomický dopad na příjmy výrobců potravin a na zákazníky. Populace rychle roste a spotřeba přírodních zdrojů vůbec neklesá. Lidé využívají přírodní zdroje pro různé účely, jako je například výroba elektroniky. Lidé dnes používají přibližně 3 miliardy miliardy chytrých telefonů. Operační systém Android je nainstalován na téměř 2,5 miliardy mobilních zařízení. Proč nezkusit použít chytré telefony k tomu, abychom pomohli snížit plýtvání potravinami? Tato práce je zaměřena na vývoj Android aplikace FoodNoWaster, která by mohla pomoci s problémem plýtvání potravinami. Základní logika aplikace byla vyvinutá v programovacím jazyku Java. Při vyvinutí jádra aplikace byl použit OOP přístup. Uživatelské rozhraní aplikace FoodNoWaster bylo vyvinuto pomocí XML souborů. Aplikace FoodNoWaster ukládá uživatelská data do SQLite databáze. Pokud se bude blížit datum expirace, uživatel o této skutečnosti bude informován pomocí notifikací. Výsledkem této práce je připravena k použití aplikace FoodNoWaster, kterou používá autor a jeho příbuzní.

Klíčová slova: Programování v Javě, OOP, Android, vývoj mobilních aplikací, SQLite databáze, plýtvání potravinami.

Table of content

List of pictures

List of tables

1 Introduction

Since my early childhood, I was in love with technology. The very first smartphone was an incredible gift for me. Back in the days, it was hard to find a proper Wi-Fi connection to download an application. In this thesis, I will try to apply knowledge gained at this university to develop an Android application using the Java programming language. The application code will be written with object-oriented programming approach and under Google guidelines.

Almost 50 years have passed since the day when Motorola company demonstrated the first handheld mobile phone. This phone weighed about 2 kilograms. Nowadays 12 new iPhones weigh less. Comparing the weight and price, one can imagine how popular these devices are now.

There are about 3 billion smartphone users today. Moreover, a smartphone user spends around 3 hours per day using the device. Without a doubt, smartphones have become an important part of our lives. They did not change us but also helped us to adapt to new life conditions where people need to store and process much more information as ever before.

Almost 30 years ago the World Wide Web became publicly available. However, the first internet connection was available at the speed of 56000 bits per second. Today one can download a 1 GB file in about 32 seconds, while 30 years ago it would take about 3.5 days. Nowadays it is almost impossible to find a person who has never heard or used the internet. With its popularization, people started to use it not only for entertainment and work but also to gain attention to certain problems such as food wasting.

The modern food waste problem began during the late 1800s and early 1900's when industrialization and manufactured products changed the way people consume food. Food became commercialized and went from farms and factories to tables. Families started to waste food as it became less expensive and more accessible. Today, about 1/3 of all the food produced for human consumption is wasted or lost.

The main goal of this thesis was to analyse and then develop an Android application that could help people to prevent their food wasting. Another goal of this thesis was to develop an application using the latest technologies and best practices available on the market. In this thesis, all stages of application development were completed. Starting with the environment set up and creating a project, the code was written, debugged, and built.

The theoretical background of the food wasting problem and the main aspects of Android application development were examined in the theoretical part. The analysis of the target audience, as well as the determination of the key functionalities of the application, were also done. The potential of the FoodNoWaster application was also explored from the market perspective. In the practical part, the theoretical background was applied during the application development process.

The main logic of the application was written in the Java programming language. The application uses the SQLite database for data storing. Data is stored only in the device' memory. The user interface is defined using the XML layout files. The application development was processed in the Android Studio integrated development environment.

2 Objectives and Methodology

2.1 Objectives

The main objective of the thesis is to explore the process of Android mobile application development, including the analysis, debugging, and testing stages. Different approaches, techniques, and alternatives will be examined during the creation of the Android mobile application.

2.2 Methodology

In the theoretical part of the thesis, an investigation of the steps of creating a mobile application using professional literature will be done. The theoretical background and analysis will be covered in the theoretical part. Then the different technical aspects of programming will be analysed with a further choice of the right structure, defining principles, and a suitable workflow. Moreover, the employment of theoretical knowledge will be done in the realisation of application. The application will be created using the selected best practices of development and also maintenance. The final application will allow the user to input the food items' data, store it in the database, display the sorted data in the lists, notify the user about the food items approaching their expiration date, and even help a user to utilize those products.

3 Literature Review

3.1 Food Waste Problem

Food waste is an issue directly linked with environmental (energy, water, climate change, resources availability), economic (price volatility, resource efficiency, consumption, increasing costs, commodity markets, waste management), and social (equality, health) impacts [1].

One third of the food produced globally for human consumption is wasted or lost, according to the Food and Agriculture Organization of the United Nations (FAO). This represents a threat to food security and shows how many resources are lost along the food supply chain (FSC). It also has a huge negative economic impact on the income of food producers and customers. In 2007, the cost of food waste at global level was estimated to be around 750 billion $. The demand for food, feed, and energy is expected to pose an unprecedented pressure on natural resources, due to a rapidly growing population. For closing the gap between the supply and demand of food, the reduction of food waste can be a potential strategy [2].

Several initiatives have been launched globally to address this issue. A specific target, aiming at halving per capita global food waste at retail and consumer levels by 2030 was adopted by the United Nations in the Sustainable Development Goals (SDGs). The Circular Economy Action Plan by the European Commission, which defines food waste as a priority area, is committed to achieving such a target [2].

Food waste prevention will bring some economic benefits related to not wasting food that has an economic value and not having to cover disposal costs. Such reduction will also bring an environmental benefit by avoiding the impacts embedded in the food items saved. With the lower number of waste management operations, some energy and material resources will be also reduced [2].

The household is the sector that contributes the most to food waste. In 2007, it produced around 47 million tonnes of food waste. This is equivalent to 92 kilograms per person per year. Households are followed by processing with its 17 million tonnes, this corresponds to 33 kilograms per person per year. These two sectors combined are 72 percent of European Union food waste. The costs associated with food waste for 28 European Union member countries are estimated at around 143 billion euros [1].

3.2 Market Development

Once upon a time, the only thing someone could do with a telephone was a phone call. That was an incredibly big technological step back in the days. In 1973 Motorola became the first company in the World who presented a wireless call from the handheld phone. Since then, we were able to get rid of the wires and call from wherever we wanted to. 19 years later, in 1992, the first Short Message Service was sent from a computer to a mobile phone. Now it is a very uncomfortable situation for a huge amount of people when they do not have a mobile phone in their hands or pocket.

The market of mobile applications has its roots both in technologies and services. People do not need to look for a postman to get your important mail anymore. In most cases, a person will get it electronically on a smartphone and will be notified about that. The technologies bring services right into your hands.

Nowadays it is hard to imagine our day without using a mobile application. Need to wake up? Use the clock application to enter the time so the application wakes you up. Need to go to work? Check your city's transport application to know the best route and schedule. Need to pay for something? Open your Bank application and do the payment in terms of seconds. Mobile applications made our lives so much easier that people do not even want to hear about paper checks, lines in banks and rude personnel. They want to open the application by one touch and be able to manage everything by themselves in the shortest time possible.

Companies like Uber or Revolut built their entire business model based on the simplicity and effectiveness of using mobile applications. Just one click and your taxi driver is on his way to you. He knows exactly where you will wait for him, you know where he is and what exact price you will pay for your ride. Revolut allowed their client to get rid of the most traditional bank charges: you can save your money when you withdraw your money in a foreign bank ATM, convert money into a different currency or when you need to issue a new plastic bank card.

In 2008 the iOS App Store opened its virtual doors with an initial 500 applications available. In the 3rd quarter of 2020, it had 1.96 million applications creating over $72 billion in consumer spending. Google's Google Play Store had about 2.87 million applications in the same period with $38.6 billion in consumer spending [3].

The first applications are dated back to the end of the 20th century, such as simple arcade games, calculators, and ringtone editors. Mobile users always demanded some sort of customization of their devices, so applications were used for such a purpose. The first applications for mobile phones were developed by handset manufacturers to improve user experience. The legendary video game called “Snake” was firstly published by Nokia for the company’s monochrome phones in 1997. The rapid evolution of mobile applications started with the expansion of operating systems for smartphones. Windows Mobile, Symbian, RIM, and iOS brings an opportunity for third-party developers to implement their own programs.

According to Moore’s law, during the last decade of the 20th century, MOS transistors went down to sub-micron level. The mobile phones industry was able to get smaller processors with greater computational power. As computer progress was developed, elements such as memory and processors became cheaper and cheaper. With more powerful and cheaper components industry needed an operating system to fulfil its requirements. In the late 1990s, personal digital assistant devices were very popular and used among mobile phones.

In January 2007, the first iPhone was introduced by Apple Computer. The smartphone was running on Apple’s own operating system – iPhone OS 1. This operating system made a revolution in the smartphones market, it allowed users to use “desktop-class applications” on their smartphones. One year later, Android OS was presented by Google. The Android operating system was built as an open-source free OS. Every producer was able to get the core for free and it led to the extreme popularity of the OS. In 2019, there were about 2.5 billion active Android devices users [4].

There are so many applications, that even the biggest technology companies are trying to integrate their functionalities right into the operating system, so you can do most of your routine tasks from your notification centre.

3.2.1 **Android vs iOS**

3.2.1.1 Market positions

The main two mobile operating systems nowadays are Google’s Android and Apple’s iOS. Those two systems share over 99% of the whole market, Apple has approximately 15%, while Google about 85%, being the best-selling mobile operating system since 2011. Each company’ mobile operating system has its own benefits as well as disadvantages. Apple’s iOS from the beginning was developed as a proprietary mobile operating system, which can be used only on Apple devices. Modified versions of iOS powers many Apple devices, for example, iPadOS running iPad, tvOS runs Apple TV and watchOS runs Apple Watch. The history of the Android operating system started in 2003 when Andy Rubin, Rich Miner, Nick Sears, and Chris White founded Android Inc. in Palo Alto, California, USA. They were creating a rival operating system to existing Symbian and Microsoft Windows Mobile. Their early intention was to create an operating system for digital cameras, this idea was presented to the investors in 2004. One year later, Android Inc. was acquired by Google. The price of the deal has never been released. In the next two years, a consortium of technology companies called Open Handset Alliance was established by device manufactures, wireless carriers, and chipset makers. Later in 2007, Google released to developers the first Android SDK Beta. The Android is an open-source mobile platform based on the Linux kernel, which allows the creation of a flexible and upgradeable operating system [5].

Since Android is an open-source platform, many companies have allocated significant engineering resources to improve the Android system. The Open Handset Alliance has also developed open standards for mobile devices. Together that means that carriers, Original Equipment Manufacturers, and developers can implement their innovative ideas in real products.

However this brings a certain possibility that when an end-user sees the name of the leading company such as Google, he expects some level of user experience. But in most cases, Google is just a platform provider, the final implementation and user satisfaction is on the software developers. Apple, on the contrary, does not provide their operating system to third companies, so a higher level of integrity can be reached inside of one company. They also have much more control over their ecosystem, providing developers with stricter guidelines and rules than Google does. That can lead to both improvement of the user experience and the limitation of the user on the other hand. Google’s Android is much more customizable, but that also brings possible threats.

3.2.1.2 Key differences

Android and iOS platforms have many core differences. For example, the development process of the Android application can be far more time consuming and more complex than for iOS. Android has a larger variety of devices that uses that operating system, so the development process will be longer due to the need to adjust the application to different screen sizes, aspect ratios, and hardware features. Android also has a bigger number of currently supported Android versions than Apple’s iOS does. Apple provides developers with incredibly specific guidelines. When Android came into existence, Java was the most used language for development. It is still an official language of Android development and is supported by Android Studio. Kotlin is another official language for Android developments but is not as widely used outside of Android Studio. The first official language for iOS development was Objective-C and it still can be used. In 2014, the new language, called Swift, was announced as a replacement for Objective-C language [6,7].

Apple has its own integrated development environment – Xcode, which is available only for the Mac operating system. An iOS application can be developed in this program. There are a variety of IDEs that can be used to develop an Android application. Google's Android Studio is the official IDE for Android, which is available for Windows, Mac, Linux, and Chrome operating systems. When it comes to application testing, Apple's iOS simulator is often called a faster one, but Android's provides a more realistic application simulation. Both platforms require application approval before publishing the application in the stores. However, Google's Play Store has a faster approval process than Apple's App Store. On the other hand, Apple's App Store review team provides the developers with feedback, which can be very useful. Apple has few promotion channels, such as the Popular App category, App of the Week, and more. Once the application successfully passes the approval process, it can be promoted through these channels, which increases the visibility of the application. There is also a different fee for enrolling as a developer. Apple offers a one-year subscription plan for 99$, while Google requires 25$ one-time fee [6,8].

Table 1 Comparison of application development for Android and iOS operating systems

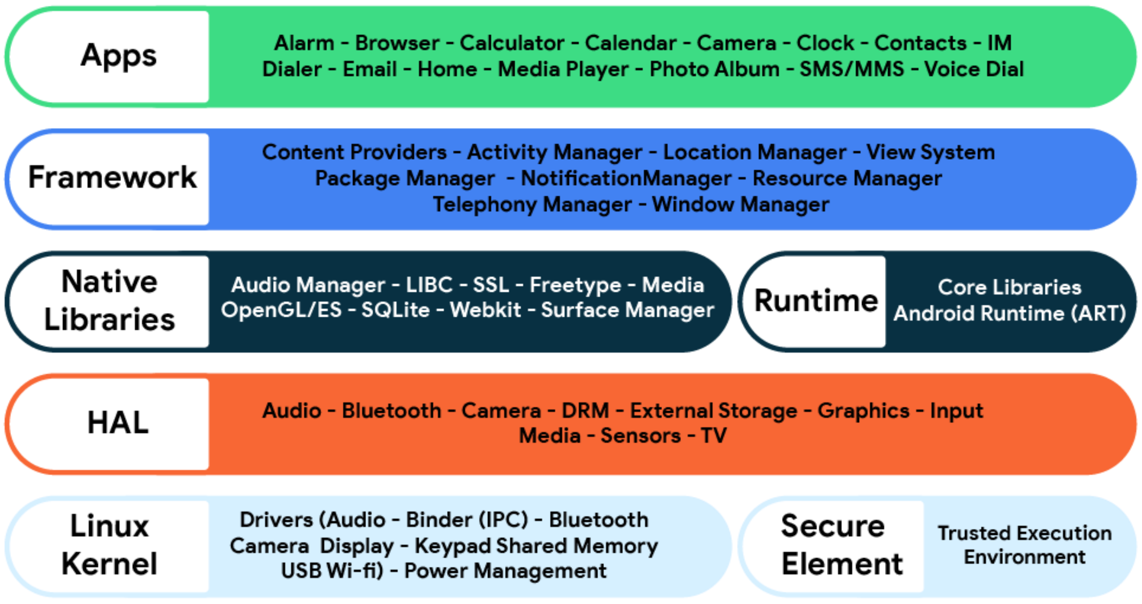
Operating System	Official development language	Official Development environment	Complexity	Approval process	Promotions	Fee
Android	Java, Kotlin	Android	Higher defragmentation	Faster	Lower chance	One-time 25$
iOS	Objective C, Swift	Xcode (Mac OS only)	Lower defragmentation	Slower, with feedback	Higher chance	One year 99$

Source: Table created according to [6,8]

3.3 Android Architecture

Various numbers of components are contained in the Android operating system architecture. These components serve to provide all needs of the Android. Android system is composed of 5 layers: Linux Kernel, HAL, Native Libraries, Framework, and Applications [9].

Figure 1 Android Stack



Apps
Alarm - Browser - Calculator - Calendar - Camera - Clock - Contacts - IM
Dialer - Email - Home - Media Player - Photo Album - SMS/MMS - Voice Dial
Framework
Content Providers - Activity Manager - Location Manager - View System
Package Manager - NotificationManager - Resource Manager
Telephony Manager - Window Manager
Native Libraries
Audio Manager - LIBC - SSL - Freetype - Media
OpenGL/ES - SQLite - Webkit - Surface Manager
Runtime
Core Libraries
Android Runtime (ART)
HAL
Audio - Bluetooth - Camera - DRM - External Storage - Graphics - Input
Media - Sensors - TV
Linux Kernel
Drivers (Audio - Binder (IPC) - Bluetooth
Camera Display - Keypad Shared Memory
USB Wi-fi) - Power Management
Secure Element
Trusted Execution Environment

Source: [9]

3.3.1 Linux Kernel

Android was created on the top of Linux Kernel - an extremely important part of the most Android devices. The Linux kernel is the largest ever collaborative software project. In 2006, over 4000 developers from over 450 companies contributed to the project. There were 6 releases with about 12 to 16 thousand changes. Android integrates the Linux kernel at the bottom of its software stack. The kernel is the first layer of software interacting with the device hardware, it is responsible for providing a basic architectural model for power management, drivers, process scheduling, and memory management. The Linux kernel contains code for all different chip architectures and hardware drivers it supports, but an individual system runs only on a part of the codebase [10].

Android uses a version of the Linux kernel with a few special additions for system functionality like Low Memory Killer and wake locks. There are numbers of reasons why Linux Kernel was selected. The main three are portability, features, and security [10].

3.3.1.1 Portability

Portability is one of the biggest advantages – Linux is an extremely portable platform. It is relatively easy to compile Linux on various hardware. Linux supports a wide range of computer architectures. Unix-like operation system is the most used operating system for public servers on the Internet. Most of the todays' supercomputers are also using Linux as an operating system. That makes it possible to run Android on almost all types of devices: from smartwatches to media players, from tablets to coffee machines, from smartphones to refrigerators.

3.3.1.2 Features

Linux Kernel includes features like process management – it is responsible for the allocation of resources to different processes that need them. For example, it can execute or stop the program.

Driver Model – here Linux ensures the ability of the application to run on Android, the manufactures can develop drivers in a familiar Linux environment. It gives the vendors plenty of space for operating system optimization.

File System Management – Linux Kernel is also responsible for file system managing. Android uses a Linux-esque file system structure with 6 partitions: boot, system, recovery, data, cache, and misc.

Network Stack – Linux Kernel is responsible for network communication, routing devices, and network adapters. The network stack works in two ways. The first is when the user makes a network request, for example for a file from a network server. The second is when the request is fulfilled when the file is returned for the user.

3.3.1.3 Security

Android completely relies on the Linux Kernel for security. All Android applications run as Linux processes under permissions set by the Linux system. Linux takes control of the authentications of the users and is responsible for user management. Users in Linux also have lower automatic access rights, which make it harder for them to spread the malware by accessing different files of the system.

Table 2 Permission Types in Linux

Number	Permission Type	Symbol
0	No Permission	`---`
1	Execute	`--x`
2	Write	`-w-`
3	Execute + Write	`-wx`
4	Read	`r--`
5	Read + Execute	`r-x`
6	Read +Write	`rw-`
7	Read + Write +Execute	`rwx`

Source: Table created according to [11]

3.3.2 **Hardware Abstraction Level (HAL)**

HAL defines a standard interface for hardware manufacturers to implement. It allows for implementing functionality without modifying or affecting the higher-level system. The modules with packed HAL implementations are loaded by the Android system at the appropriate time. Starting with Android 8.0, the new Android OS framework architecture allows its users to replace the framework without HAL rebuilding. It allows the manufacturers to update devices to a new Android version in an easier, faster, and cheaper way. The vendor simply provides access to the hardware-specific parts of Android, so manufacturers can provide new Android releases by updating the Android OS framework without additional input from the silicon manufacturers [10].

3.3.3 **Native Libraries**

Many of the Android system core components and *services*, for example, HAL, are built from native code that requires native C and C++ libraries. The Java framework APIs are provided by Android to applications to expose some functionality of native libraries. Since Android 5.0, each application runs with its own instance of the Android Runtime (ART) and runs in its own process. ART runs multiple virtual machines on low-memory devices by executing bytecode files in the DEX format designed specifically for Android to minimize memory footprint [10].

3.3.4 **Framework**

The application framework is mostly used by application developers. All Android OS features are available to developers through Application Programming Interfaces (APIs) written in the Java programming language [12]. These APIs are building blocks which allow the developers to reuse the core, modular system components, and *services* to create Android application. Developers have the same full access to framework APIs as Android system applications [10].

3.3.5 **System Applications**

Android comes with core applications for SMS messaging, internet browsing, contacts, calendars, email, and more. Those included applications have no special status through the system, so a third-party application can be chosen as the user's default for example for web browsing. The system applications also provide key capabilities that developers can access from their own application without the need to implement that functionality on their own. Developers can just invoke the needed functionality from the system application [10].

3.4 Android Application Fundamentals

Android applications can be written using three programming languages: Kotlin, Java, and C++. The Android SDK tools compile code with resource and data files into an Android package (APK), which is an archive file with `.apk` extension. All the contents of an Android application are contained in one APK file. Android-powered devices use the APK file to install the application [13, 14].

All Android applications live in their own sandboxes, which are protected by Android security features. Since the Android operating system is a multi-user Linux-based system, every application acts as a different user. By default, a unique Linux user ID is assigned to each application by the system. This ID is unknown to the application and is used only by the Android operating system. With the ID, the system also sets the permissions for all the files, so only the user with the assigned ID to the application can access them. Each application code runs isolated from other applications as a separate process in its own virtual machine (VM). Every process is started by the Android system when any of the application's components need it to be executed. The process shuts down by the Android when the system must recover free memory for other applications or when it is no longer needed in that process [13, 14].

The principle of least privilege is implemented in the Android system. That means that by default every application has access only to the components that it requires to run its processes and no more. The application cannot access parts of the system for which permissions are not given, which creates a very secure environment. However, the application can still share data with other applications and access system *services*. The same Linux user ID can be shared by two applications to make them possible to access each other files. The application can run in the same Linux process. If the applications are signed with the same certificate, they can also share the same virtual machine. To access device data like user's contacts, SMS messages, Bluetooth connection or the device's location, these permissions must be explicitly granted to the application by the user [13, 14].

3.4.1 **Application Components**

Every Android application is built from essential blocks - application components. Each component serves as an entry point through which the user or the system can enter the application. Some components are dependent on others. A unique aspect of the Android system design allows any application to start another application's component. For example, if there is a need to take a picture in a bank application, the bank application developers do not need to develop the functionality of making pictures by themselves. They can simply start the *activity* in the camera application that captures a photo. For the user, it seems like the camera application is a part of the bank application. Android applications do not have a single entry point, so when the system starts a component, it starts the process for that application. Since the system runs each application as a separate process with file permissions that restrict access to other applications, the applications cannot directly activate a component of other applications. However, the Android operating system can activate a component if the application will deliver a message with an *intent* to start a particular component. There are four types of application components: *Activities*, *Services*, *Broadcast receivers*, and *Content providers*. Each type has its own purpose and lifecycle that define how the component is created and destroyed [14].

3.4.1.1 Activities

The entry point for interacting with the user is an *activity*. It represents a user interface on a single screen. For example, in the phone application, there is one *activity* that enables you to make a phone call, another *activity* is to check recent calls history, and another *activity* is to watch your favourites contacts. Although *activities* are working together from the user point of view, each one is independent of the other. As a result, a different application can start any *activity* that a phone application allows. For example, your taxi application can use the phone call *activity* of a phone application to make a call. The Android operating system keeps track of what is on the user's screen to keep running the process that is hosting the *activity*. The Android also knows what the previously used processes were and keeps those processes around, so the user can return to them faster. Also, the system helps applications with process kill, which allows the user to restore the previous state. Finally, Android provides a way for applications to implement the user flows between them and coordinates those flows [15].

While the user navigates through, exits, or goes back to the application, the `Activity` class instances in the application transition through different states in their lifecycle. States changes, such as that the system is creating, resuming, or stopping an *activity*, or destroying the process where the *activity* resides, are caught by *callbacks* provided by the `Activity` class. The developer can declare behavior of the *activity* when the user leaves and goes back to the *activity* thanks to the lifecycle *callback* methods. For example, when the user watches the video online and decides to switch to another app, video playback and network connection can be terminated. When the user returns to the video player, the network connection and video playback can be resumed. Each *callback* allows the developer to do a specific work that is suitable for a given state change. The right implementation of the lifecycle *callbacks* ensures that the application avoids crashing while the user switches to another application, unnecessary consummation of valuable system resources while the user is not actively using the application, losing the user's progress while closing and returning to the application [15].

The `Activity` class provides a core set of six *callbacks* to navigate transitions between stages of the *activity* lifecycle: `onCreate()`, `onStart()`, `onResume()`, `onPause()`, `onStop()`, and `onDestroy()`. Each of these *callbacks* is invoked by the system as an *activity* enters a new state [15].

- `onCreate()` – the *callback* is called when the *activity* is first created. The *activity* enters the Created state on *activity* creation. A basic application startup that happens only once during the entire life of the *activity* is performed in this method. After finishing the execution of the `onCreate()` method, the Started state is entered by the *activity*. Then the system calls the `onStart()` and `onResume()` methods [15].

- `onStart()` – this *callback* is invoked when the *activity* enters the Started state. It makes the *activity* visible to the user. For example, the `onStart()` method is called where the application initializes the code that maintains the user interface. The method is followed by `onResume()` if the *activity* comes to the foreground, or `onStop()` if it comes to background [15].

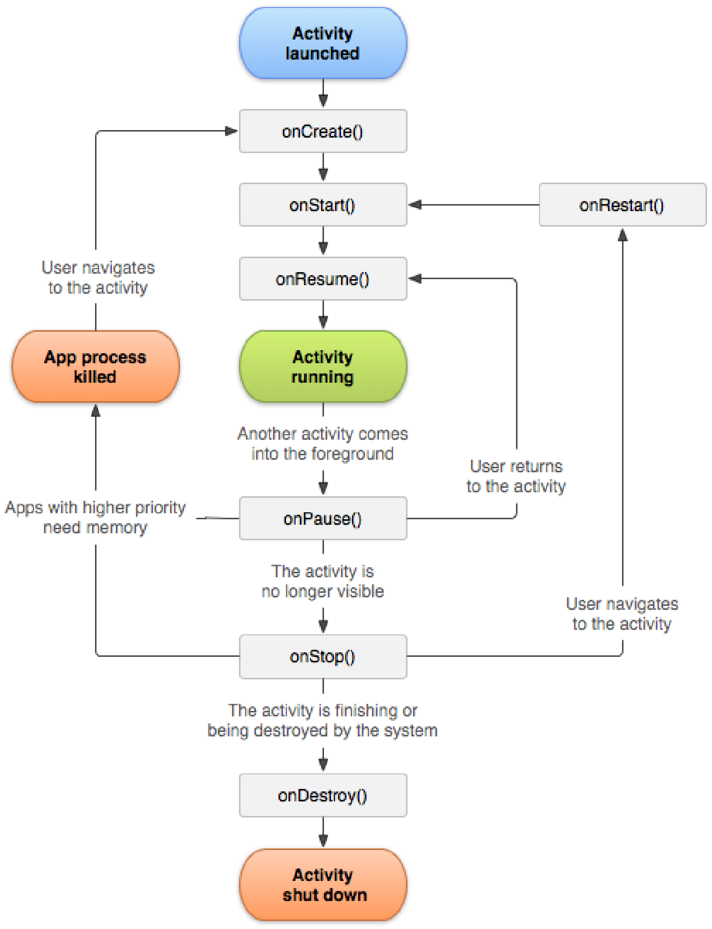
- `onResume()` – the system invokes this *callback* when the *activity* enters the Resume state and it comes to the foreground. The application interacts with the user in this state. The resume state is not left till something happens away from the application. For example, the music stops playing in the application when users receive a phone call. The Paused state is entered when an interruptive event occurs, and `onPause()` *callback* is invoked by the system. If the *activity* moves from the Paused state to the Resumed state, the system calls the `onResume()` method once again. This method is always followed by an `onPause()` *callback* [15].

- `onPause()` – the method is called the first indication that the user is leaving the *activity*, it indicates that the *activity* is no more in the foreground. This *callback* is used to pause the operations that should not continue during the Paused state, but that are expected to resume shortly. For example, when some events interrupt the application execution or when the system pauses the application due to multiple applications running in multi-window mode. System resources can also be released by invoking the `onPause()` *callback*. The *activity* is visible to the user, so it should be visually active, and the UI should also be updated. The `onPause()` *callback* is followed by `onResume()` method if the *activity* returns back to foreground, or `onStop()` method if it becomes invisible to the user [15].

- `onStop()` – when this *callback* is invoked, the Stopped state is entered, and *activity* is no longer visible to the user. For example, this occurs when a newly started *activity* covers the entire screen. The `onStop()` method can also be called by the system when the *activity* finishes running and is going to be terminated. In this method, the application releases the resources that are not needed while the application is not visible to the user. The *callback* should be used for performing CPU-intensive shut down operations. From the Stopped state, the *activity* can come back by invoking the `onRestart()` method or it can finish the run by invoking the `onDestroy()` method [15].

- `onDestroy()` – this *callback* is called before the *activity* is destroyed. This method is called by the system when the *activity* is finishing, or when the system is temporarily destroying the *activity* due to a configuration change. *Callback* `onDestroy()` is the final lifecycle *callback* that the *activity* receives when the *activity* is finishing. If a configuration change occurred and the `onDestroy()` method was called as a result, a new *activity* instance is immediately created by the system. After that, the system calls `onCreate()` *callback* on that new instance in the new configuration. All resources that have not been released by earlier *callbacks*, should be released in the `onDestroy()` method [15].

Figure 2 Android activity lifecycle


Activity launched
onCreate()
onStart()
onRestart()
User navigates to the activity
onResume()
App process killed
Activity running
Another activity comes into the foreground
User returns to the activity
Apps with higher priority need memory
onPause()
The activity is no longer visible
User navigates to the activity
onStop()
The activity is finishing or being destroyed by the system
onDestroy()
Activity shut down

Source: [15]

3.4.1.2 Services

A *service* is a component that helps to run the application in the background. It runs in the background to perform long-running operations or to perform work for remote processes. A user interface is not provided by a *service*. For example, a *service* can run the music application in the background, while the user surfs the internet in a web browser. A *service* can be started by another component, such as an *activity*, and let it run or bind to it to interact with it. These components run at the backend, updating the data sources and *activities*, *broadcast intents*, and triggering notifications. There are three different types of *services*: foreground, background, and bound [16].

- Foreground *services* perform operations that are visible for users. For example, a web browser application would use a foreground *service* to display web pages. Foreground *services* must display notifications. Even when the user is not interacting with the application, the foreground *services* continue to run. When a foreground *service* is used, the application must display a notification for the user, notifying him that the *service* is running. The notification is active till the *service* is stopped or removed from the foreground [16].

- Background *services* perform operations that are not directly noticeable for users. For example, a background *service* can be used in an application for compacting its storage [16].

- Bound *services* offer client-server interfaces that allow components to interact with the *services*, send requests and receive results. These *services* run only till another application component is bound to them [16].

A *service* is created by creating a subclass of `Service` class or using one of its existing subclasses. Some *callback* methods that handle key aspects of the *service* lifecycle and provide a mechanism that allows the components to bind the *service* must be overridden during the implementation:

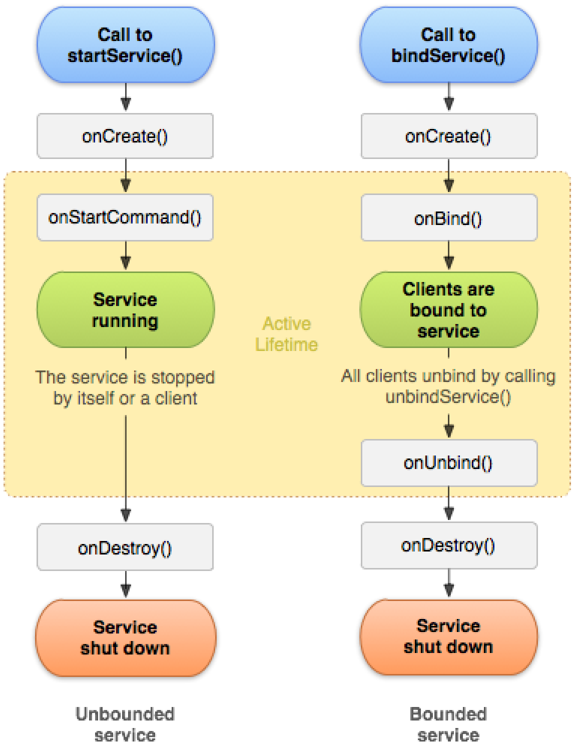
- `onStartCommand()` – is invoked by the system by calling the `startService()` method when another component requests that the *service* be started. Within the execution of this method, the *service* is starting and can run indefinitely in the background. When the work is complete, the *service* should be stopped by calling the `stopSelf()` or the `stopService()` methods. There is no need for implementing this method if only binding providing is needed [16].

- `onBind()` – the system invokes this method by calling the `bindService()` method when another component wants to bind with the *service*. In the implementation of this method, an interface that clients use to communicate with the *service* by returning an IBinder object must be provided. This method must be always implemented, however, if no binding is needed, null can be returned [16].

- `onCreate()` – this method is invoked by the system to carry out one-time setup procedures when the *service* is initially created before it calls the `onStartCommand()` or the `onBind()` *services*. This method is not called if the *service* is already running [16].

- `onDestroy()` – when the *service* is no longer used and is being destroyed, the Android system invokes this method. This is the last call the *service* receives, so this method should clean up threads, registered listeners, and receivers [16].

Figure 3 Android service lifecycle



Source: [16]

Service is stopped by the Android system only if memory is low and it must recover system resources for the *activity* that the user is focused on. If the *service* is declared to run in the foreground, it is rarely killed. If the *service* is bound to an *activity* that the user is focused on, it is likely to be killed. If the *service* is in a long run, its position in the list of background tasks lowers by a system over time, and the chance to be killed increases. Due to that, the *services* should be designed to handle the restarts by the system. If the system kills the *service*, it restarts it as resources become available [16].

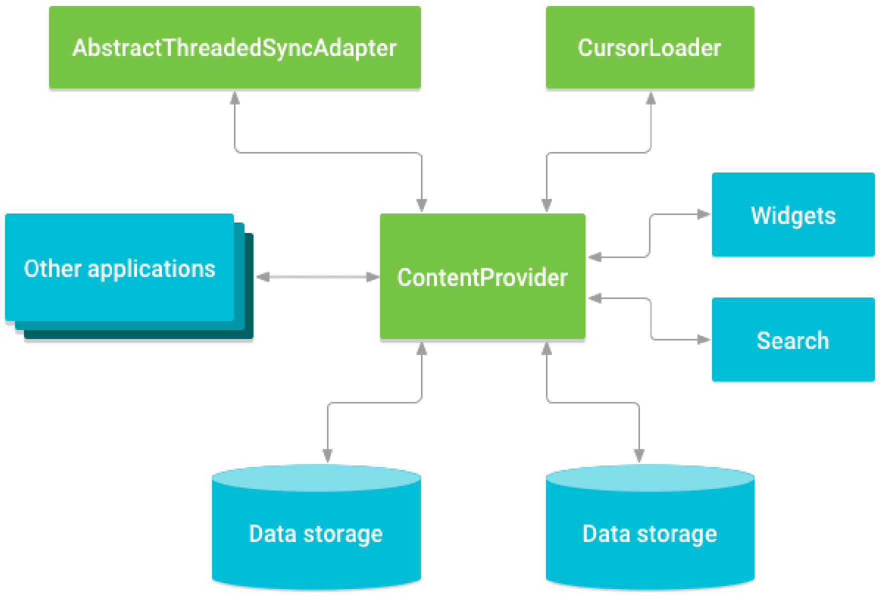
3.4.1.3 Broadcast receivers

A *broadcast receiver* is a component that allows the application to respond to system-wide *broadcasted messages* from other applications or the system. The system can deliver *broadcast events* even to applications that are not currently running. For example, the application can send a *broadcast message* that some file was downloaded to let other applications know that they can use it. Many of the *broadcasts* are generated by the system, for example, the *broadcast announcement* about the low battery or the turned off screen. After receiving the message, the *broadcast receiver* intercepts the communication and initiates predefined action. Even though *broadcast receivers* cannot display a user interface, they can create a status bar notification to notify the user about a *broadcast event*. In most cases, they are used as a gateway to other components [17].

3.4.1.4 Content providers

A *content provider* manages a shared set of application data that you can store on any storage location that the application can access. For example, the application can store data in the file system, on the web server, or in an SQLite database. Other applications if they are allowed can query or modify the data through the *content provider*. For example, any application with proper permissions can query the user's contact information provided by the Android system, and then read and write information about a particular contact. By using *content providers*, it is also possible to read and write data that is private to the particular application and not shared [18].

Figure 4 Relationship between content provider and other components



Source: [19]

3.4.1.5 Intents

The *intent* in the Android operating system is an abstract description of an operation to be performed. It is a framework that passes the messages between the applications. Three component types can be activated by an *intent*: *activities*, *services*, and *broadcast receivers*. An *intent* can activate a specific component or a specific type of component. For *activity* and *services*, the *intent* can define the action to perform, like to view or send something. For example, an *intent* can transfer a request for a web page opening. For *broadcast receivers*, the *intent* simply defines the *broadcasted message*. For example, for low battery indication, the action string that indicates the battery is low will be broadcasted. Unlike *activities*, *services*, and *broadcast receivers*, *content providers* cannot be activated by *intents*. They are activated by a targeted request from a ContentResolver. All direct transactions with the *content provider* are handled by the content resolver [20].

3.4.2 **The manifest file**

To start an application component the Android operating system must know that the component exists. The `AndroidManifest.xml` file is the application *manifest file*, where the application must declare all its components. *Activities*, *services*, and *content providers* that are not declared in the *manifest file* are not visible to Android, so they can't be run. This file must be placed at the root directory of the application project hierarchy. The *manifest file* also identifies all user permissions the application requires, declares the minimum API level required by the application, declares software and hardware features required or used by the application, and declares API libraries the application needs to be linked against. This file includes nodes that make the application, for *activities*, *services*, *content providers*, and *broadcast receivers*. It also uses *intent* filters and permission for determining how they coordinate with each other and other applications [14].

3.4.3 **Application Resources**

An Android application is not composed just of the code, but also it requires resources that are separate from the code. For example, images, audio files, and everything related to the visual appearance of the application. Animations, colours, styles, menus, and layouts of user interfaces can be defined in an XML file. Application resources make it easy to update specific characteristics of the application without the need of modifying a source code. Application resources also give a possibility to optimize the application for different devices' configurations. For example, screen sizes and different languages. The SDK build tools define a unique integer ID for every resource included in the Android project. This ID can be used for referencing the resource from the application code or the other resources defined in an XML file. For example, by defining UI strings in an XML file, the developer can translate the strings into different languages and store the translations in separate files. The appropriate language strings will be automatically applied by Android, based on the language qualifier in the resource directory's name and the user's language setting [14].

A lot of qualifiers for alternative resources are supported by Android. To define the resources used for specific device configuration, a short string, called qualifier, is included in the name of resource directories. For example, the developer can create and define two different UI layouts for both portrait and landscape screen orientations. Those layouts can be changed based on the orientation by applying the appropriate qualifier to each layout's directory name [14].

3.4.4 **Application features**

Mobile phones are not expected to be only phones today, but also cameras, personal assistants, instant-messenger clients and to do almost everything else a computer could do. With all these applications running on smartphones, applications need a way to notify users about actions and to get users’ attention. To do that, Android provides the developers’ with several options on how to notify the users: *notifications*, *alarms*, and *toasts*.

3.4.4.1 Notifications

A *notification* is a message that Android displays outside the application’s UI to provide the user with reminders, communication from other people, or other timely information from the application. Users can take an action directly from the *notification* or tap the *notification* to open the application. *Notifications* appear to users in different locations and formats, such as an icon in the status bar, a detailed entry in the notification drawer, and a badge on the application’s icon. Beginning with Android 5.0, *notifications* can also appear on the lock screen. If the user has a paired Wear OS device, all *notifications* will also be displayed there automatically [21].

Starting in Android 8.0 (Oreo, API level 26), all *notifications* must be assigned to a channel. Users can disable specific *notification channels* for the application by categorizing *notifications* into channels. On devices running Android 7.1 (Nougat, API level 25) and lower, users can manage *notifications* only on a per-application basis. Android uses the importance of *notification* to determine how much the *notification* can interrupt the user. On Android 7.1. and below, the importance of each *notification* is determined by the *notification’s* priority parameter. On Android 8.0 and above, importance is determined by the importance parameter of the channel the *notification* was posted to [21].

3.4.4.2 Repeated alarms

Alarms in Android are based on `AlarmManager` class. They provide a developer with a way to perform time-based operations outside the lifetime of the application. For example, the *alarm* can be used to initiate a long-running operation, such as starting a *service* once a day to download a weather forecast. *Alarms* let the developer to fire *Intents* at set times or intervals. They can be used in conjunction with *broadcast receivers* to start *services* and perform other operations. Alarms can trigger events or actions when the application is not running, and even the device itself is asleep. *Alarms* can also help the developer to minimize the application’s resource requirements by not relying on timers or continuously running background *services* [22].

3.4.4.3 Toasts

A *toast* provides simple feedback about an operation in a small popup. It only fills the amount of space required for the message. The current *activity* remains visible and interactive. *Toasts* automatically disappear after a timeout. *Toasts* are not clickable. To create a *toast*, an instance of a `Toast` object with one of the `makeText()` methods should be created. A *toast* can be positioned differently within the screen layout. A customized *toast* `View` layout can be created if a simple text message is not enough [23].

3.4.5 **Debugging, testing, and publishing**

Debugging is the process of detecting and correcting errors in a program. There are several types of errors the developer could face. Some of them, like syntax errors, are easy to track and fix, another, like subtle logic errors, might be a real challenge to sort. A debugger is a powerful tool that helps the developer to find bugs a lot faster by providing insight into the internal operations of a program.

3.4.5.1 Debugging

Android Studio provides a debugger that allows the developer to select a device to debug the application on, set breakpoints in code written in several programming languages, examine variables and evaluate expressions at runtime. To start debugging, debugging should be enabled on the device. In Android Studio Emulator, it is enabled by default. For a connected device, debugging should be enabled in the device developer options [24].

The Android Studio supports several types of breakpoints that trigger different debugging actions. The most common type is a line breakpoint that pauses the execution of the application at a specified line of code. While the application is paused, the developer can examine variables, evaluate expressions, then continue execution line by line to determine the causes of runtime errors. When code execution reaches the breakpoint, the execution of the program is paused by the Android Studio. Then the tools in the Debugger tab can be used to identify the state of the application. When Android Studio deploys the application to the target device, the Debug window opens with a tab or debug session view for each debugger process. The Variable pane in the Debugger view allows the developer to inspect variables when the system stops the application on a breakpoint and a frame from the Frames pane is selected. The Watches pane provides similar functionality except that expressions added to the Watches pane persist between debugging sessions [24].

The system log shows system messages while the developer debugs the application. These messages include information from applications running on the device. To use the system log to debug the application, the code should write log messages and print the stack trace for exceptions while the application is in the development phase. The `Log` class should be used to write log messages in the code [24].

3.4.5.2 Android Debug Bridge

While building an Android application, it is important to test the application before releasing it to users in an emulator and even more important, on a real device. The Android Studio development environment and Android device can be set for testing and debugging over an Android Debug Bridge (ADB) connection. The Android device can be connected using a USB cable or over a Wi-Fi network. First, an appropriate OEM driver should be installed on a computer to detect a device. After a driver is set and the device is connected to the computer, the developer can communicate with a device. The ADB command facilitates a variety of device actions, such as installing and debugging applications and provides access to a Unix shell that can be used to run a variety of commands on a device [25].

3.4.5.3 Publishing

Publishing is a general process that makes the Android application available to users. There are two steps in the publishing process: preparation and release. During the preparation step, the developer builds a release version of the application, which users will be able to download and install on their Android-powered devices. During the release step, the developer publicizes and distributes the release version of the application to the users. When the application' preparation for release is finished, the developer will have a signed `.apk` file that can be distributed to users [26].

The application can be released in several ways. Usually, the application is released through an application marketplace, such as Google Play or Huawei App Store, the developer can also release the application on own website or by sending the application directly to a user [26].

3.5 Analysis

3.5.1 **UML**

The Unified Modeling Language (UML) is an industry standard for object-oriented design notation. It is a language for visualizing, specifying, constructing, and documenting the artefacts of a software system. UML provides a pictorial and graphical notation for documenting classes, objects, and packages that object-oriented systems are built of. Interactions can be modelled at different abstraction levels [27, 028].

3.5.1.1 Use Case

Use Case diagrams are often used in the early stage of the development process. It is a top-level view of interactions. Use Case modelling is developed for specifying the context of the system, capturing the system requirements, validating a system architecture, driving implementation and generating test cases. A standard form of a Use Case is defined in the UML. An actor is someone who interacts with the Use Case, it is an external system-user who directly communicates with a system, but is not a part of the system. Use Case is a process, it represents a system function. It is a functionality which the system provides during the communication with the actor. Use Cases deal not only with users, but they can also deal with the system itself. The participation of an actor in a Use Case is presented by connecting the actor with a Use Case by a communication link. Use cases divide system functionalities in a whole model and they should have a comparable level of abstraction [28].

3.5.2 **SWOT**

SWOT is an analysis technique frequently used in strategic planning. SWOT is an acronym for Strengths, Weaknesses, Opportunities, and Threats. This is a structured planning method that evaluates four elements of an organisation or a project. SWOT analysis is a process where the internal and external factors that will affect the future company’s or project performance are identified. This analysis can be performed for each product or service separately, as well as a part of a project [29].

3.5.3 **Gantt chart**

The Gantt chart is one of the most used planning and controlling tools in projects today. It has been developed nearly a hundred years ago and is being used even today despite numerous innovations in the area. Gantt chart remains an important tool for planning and controlling the project schedule.

The Gantt chart is a simple, intuitive, practical, and useful visual representation of the project activities and their durations. It is useful to communicate the project schedule and create a shared understanding of the progress of tasks. In the projects, the Gantt chart can be used by project managers to point the potential schedule delays and focus the project team's attention on the critical tasks to the delivery of project time. Within small projects, it can be useful for defining the schedule for critical tasks and a project as a whole [30].

3.6 Development Environment

3.6.1 Integrated Development Environment

An Integrated Development Environment (IDE) is software that combines common developer tools into a single graphical interface. An IDE usually consists of a source code editor, local build automation tools, and a debugger. A source code editor is a text editor that can assist the developer in writing software code with features like syntax highlighting with visual cues, autocomplete, and bugs check while code is being written. This helps the developer to save time and makes the code easier to read. The compilation process is necessary for every program, and IDEs provide the developers with automated build processes for languages. They can compile the source code into binary code, pack binary code, and run automated tests. A debugger is a program for testing other programs that can visually display the location of a bug in the original code. It also allows the developer to examine different variables and deliberately inspect the code. Debuggers can also provide hints to prevent errors even before compiling. Some IDEs can contain compiler or interpreter, or both, while some do not contain any .

An IDE allows developers to start programming new applications quickly without the need for manual configuration and integration of multiple utilities. In IDEs, utilities are represented by a single Graphic User Interface (GUI), so that helps the developers to organize their workflow and solve problems, as they do not need to switch between applications. Some IDEs additionally provide the developers with class and object browsers, and even class hierarchy diagrams for certain languages.

There are a lot of options on the market for developers to choose from. Almost every Integrated Development Environment can offer some unique features that can be very useful for the developers.

3.6.1.1 Android Studio

The Android Studio is Google’s official IDE for Android mobile application development. It is based on IntelliJ IDEA with features custom-designed for Android. On top of IntelliJ's powerful code editor and developer tools, Android Studio offers even more features that enhance productivity when building Android applications, such as a flexible Gradle-based build system, a fast and feature-rich emulator, a unified environment, code templates and GitHub integration, extensive testing tools and frameworks, C++ and Native Development Kit (NDK) support, built-in support for Google Cloud Platform. It also allows the developer to push code and resource changes to the running application without the need to restart the application. Lint tools to catch performance, usability, version compatibility, and other problems are also offered by Android Studio. The Android Studio supports Java and Kotlin programming languages. C++ and C support is provided with the NDK toolset. This IDE also provides real-time statistics for memory usage, network activity, and application performance. Android Studio IDE is available for Windows, Mac, Linux, and Chrome OS platforms [31].

3.6.1.2 Eclipse

Eclipse Android Development Tools (ADT) was the preferred IDE for native Android application development before Android Studio was released. It is free and open-source software released under the Eclipse Public License. The Eclipse began as a Java IDE but since grown to support many different programming languages, like Java, C, C++, C#, JavaScript, PHP, Python, Ruby, and many others. It offers features such as extensive customization abilities, large plug-ins marketplace, integration with Git, Maven and other popular development tools, large community, and user base. It is available for Windows, Linux, and Mac operating systems [32].

3.6.1.3 Visual Studio

Microsoft’s flagship IDE, Visual Studio, can be used for Android development with the inclusion of Xamarin. C# programming language is mostly used in Xamarin. Using Xamarin, the developers can create native applications for both main mobile platforms – Android and iOS, and even for Windows. Visual Studio provides tools for application building, managing teams, version control managing, and services building. It also supports multiple languages and platforms, provides the developers features like CodeLens, Peek Definition, refactoring, integrated testing, dependency graphs and code maps. Visual Studio IDE supports dozens of programming languages, such as C#, C++, Visual Basic, Python, HTML, JavaScript, PHP, and others. It comes in multiple versions: Enterprise for large teams, Professional for small teams, Visual Studio for Mac, and free Visual Studio Community edition. It also has a separate open-source code editor – Visual Studio Code. Visual Studio IDE is available for Window, Mac, and Linux operating systems [33].

3.6.1.4 IntelliJ IDEA

The IntelliJ IDEA is an IDE written in Java programming language developed by JetBrains. It was created for importing, developing, modelling, and deploying computer software. It is designed for ultimate programmer productivity. The IntelliJ IDEA provides the developers with features like Smart code completion (gives a developer list of the most relevant symbols applicable in the current content), Chain completion (even deeper list accessible via methods or getters), Static members completion, Data flow analysis, Language injection, cross-language refactoring, detecting duplicates, inspections, and quick fixes. It offers its users an editor-centric environment, dedicated keyboard shortcuts for nearly everything, an ergonomic interface, and an inline debugger. Developer tools like version control (it supports Git, SVN, Mercurial and others), build tools (Maven, Gradle, Ant and others are supported), decompiler, terminal, database tools, application servers, and docker are built-in and offered to the users of this IDE. While IntelliJ IDEA is an IDE for Java, it also understands many other languages, including Kotlin, Groovy, Scala, JavaScript, SQL, and others The IntelliJ IDEA comes as a free open-source Community version, and a paid Ultimate edition version. It can be run on Windows, Mac, and Linux platforms [34].

3.6.2 **IDEs Comparison**

The IDE selection might be a relatively hard process. Even though Android Studio IDE is Google’s recommended option for Android development, due to different reasons it might not always be the best choice. For example, if the developer plans to write a cross-platform application that can run on Android and iOS, the cross-platform IDEs such as Visual Studio or Eclipse should be considered as the right choice. If the developer is used to some IDE, it could be easier and faster to stay with the known tool instead of learning a new one. The determination of which IDE should the developer use must be done based on individual needs and project requirements.

Table 3 Android Studio, Eclipse, Visual Studio, and IntelliJ IDEA IDEs comparison

IDE	Languages	Target OS	Runs On	Unique features	License Price
Android Studio	Java, Kotlin, C, C++	Android	Windows, Mac, Linux, Chrome OS	Code and resource changes push without restart	free
Eclipse	Java, C, C++, C#, JavaScript, and more	Android, iOS, Linux, Mac, Windows, and more	Windows, Mac, Linux	Marketplace with lots of plugins	free
Visual Studio	C++, C, C#, Visual Basic, PHP, JavaScript, and more	Windows, Android, iOS, and more	Windows, Mac, Linux	Cross-platform	Community – free, Professional – 45$ per month, Enterprise – 250$ per month
IntelliJ IDEA	Java, Kotlin, Groovy, Scala, JavaScript, SQL, TypeScript	Any Java supporting OS	Windows, Mac, Linux	Deep code insight	Community – free, Ultimate – from 149$ per year

Source: Table created according to [31,032,033,034]

3.6.3 **Android Studio System Requirements**

The system requirements for Android Studio, Eclipse, Visual Studio, and IntelliJ IDEA are almost identical for supported platforms. For example, system requirements for the Android Studio among the supported operating systems are listed [31].

Table 4 System requirements for Android Studio IDE

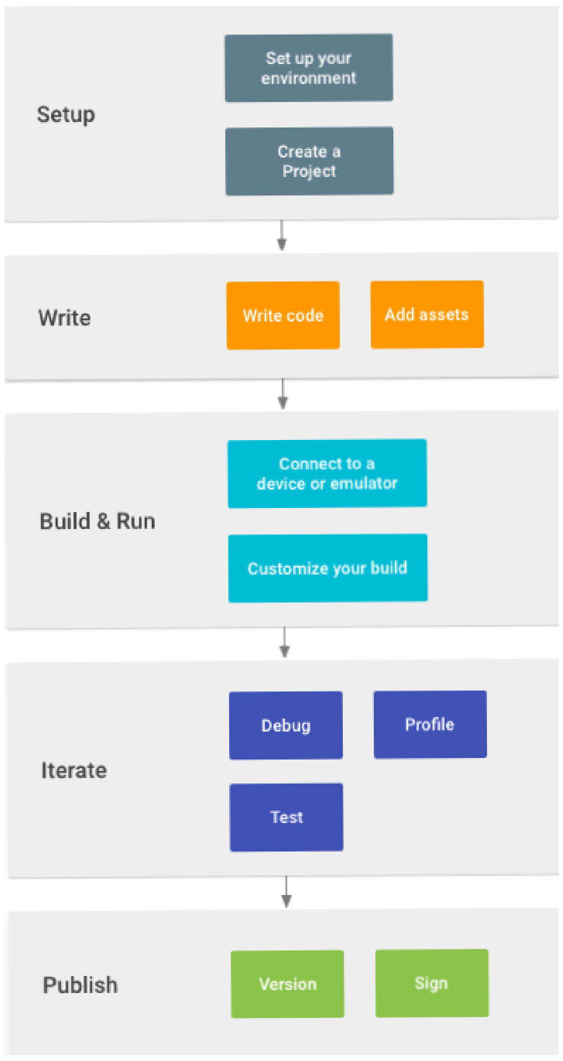
Operating System	OS Version	RAM	Disk space	Screen resolution
Windows	Microsoft® Windows® 7/8/10 (64-bit)	4 GB RAM minimum, 8 GB RAM recommended	2 GB of available disk space minimum, 4 GB Recommended (500 MB for IDE + 1.5 GB for Android SDK and emulator system image)	1280 x 800 minimum screen resolution
Mac	Mac® OS X® 10.10 (Yosemite) or higher, up to 10.14 (macOS Mojave)			
Linux	GNOME or KDE desktop. 64-bit distribution capable of running 32-bit applications. GNU C Library (glibc) 2.19 or later			
Chrome	Chrome OS 77 or later	8 GB RAM or more recommended	4 GB of available disk space minimum	

Source: Table created according to [31]

3.6.4 **Android application development workflow**

The workflow to develop an application for the Android system is conceptually the same as other application platforms. Some special tools are needed to efficiently build a well-designed application for Android. The first phase of the workflow is setting up the environment. In this phase, the Integrated Development Environment should be installed, and a new project must be created. In the second phase, the application code should be written. The variety of tools and intelligence provided by the IDE can be used in this phase to write high quality code, design a UI, and create resources for different device types. During the third phase, the project is built into a debuggable APK package that can be installed on the emulator or an Android-powered device. The customization of the build can begin in this phase. The fifth phase is iterative. In this phase the developers continue to write the application code, but with a focus on bugs elimination and application performance optimization. Various performance metrics such as network traffic, memory usage, CPU performance, and others should be reviewed and analysed during this phase. In the fifth phase, the application should also be tested. The sixth phase is publishing. Publishing is the process that makes Android applications available to users. The versioning of the application and signing it with a key should also be considered in this phase [35].

Figure 5 Android development workflow



Source: [35]

4 Practical Part

4.1 Application idea

The main idea of the FoodNoWaste application is to provide a service to the users that will track the preferred consumption date for a product. The user will be notified by the application about approaching the expiration date of the product. For example, the user will buy a lump of chicken meat with a one-week expiration date. The user will put the product name and the expiration date into the application. A few days later, if the product is still in the database and marked as active, the application will notify the user about the approaching expiration date of that product to help the user to consume it, and to prevent food wasting.

4.1.1 **Target audience analysis**

Firstly, there is a need to specify the target audience of the application. A target audience is a group of people that the product or the service is intended for. A well-defined target audience can benefit both the technical and business factors. For example, the application cannot be meant for everyone, it is simply impossible to please everyone. Different groups of people have different needs and interests. Moreover, they have different expectations and requirements. It also does not make sense to build an application just to satisfy one user and his or her needs.

The target audience of the FoodNoWaste application is a group of people that cares about nature and natural resources saving, people who are concerned about the impact of farming, food-production processes, and how much food is being thrown away without consumption. The second group could be students and lower to middle-class households and individuals. That group might want to use the opportunity to reduce their expenses by the elimination of food wasting.

4.1.2 **Alternative solutions**

There are already a lot of applications on the market that have a goal of reducing the food waste problem. The food production and supply chain consist of a lot of steps: production, processing, distribution, and finally consumption. The different solutions are oriented to the different parts of that chain. The Karma app allows users to reserve and pick up surplus food from restaurants, grocery stores, farms, and wholesalers. In the beginning, the Karma start-up was an application that worked as a platform with deals for all kinds of products. Later, when co-founders realized that some restaurants were uploading deals with surplus food, they decided to focus on that segment. The Karma application allows retailers to sell their surplus food to customers at a lower price instead of food wasting [36]. There is also another type of applications, for example OLIO. That application allows you to list and post a picture of unwanted food items that the user wants to be shared with the neighbourhood. The OLIO application helps to share the food in the neighbourhood, that otherwise might be thrown away. The application is used by local individual users as well as by the organisation such as retailers or corporate events providers. The OLIO helps them to become zero food waste organisations [37]. The FoodKeeper is the official United States Department of Agriculture (USDA) application developed with Cornell University and the Food Marketing Institute. This application is intended to help the user with maximizing the freshness and quality of items. The FoodKeeper application guides the safe preparation, handling, and storage of food items. The user can access cooking tips and methods through this application. It can also send a notification to the user about the food item nearing the end of their recommended storage date [38].

4.1.3 **Requirements analysis**

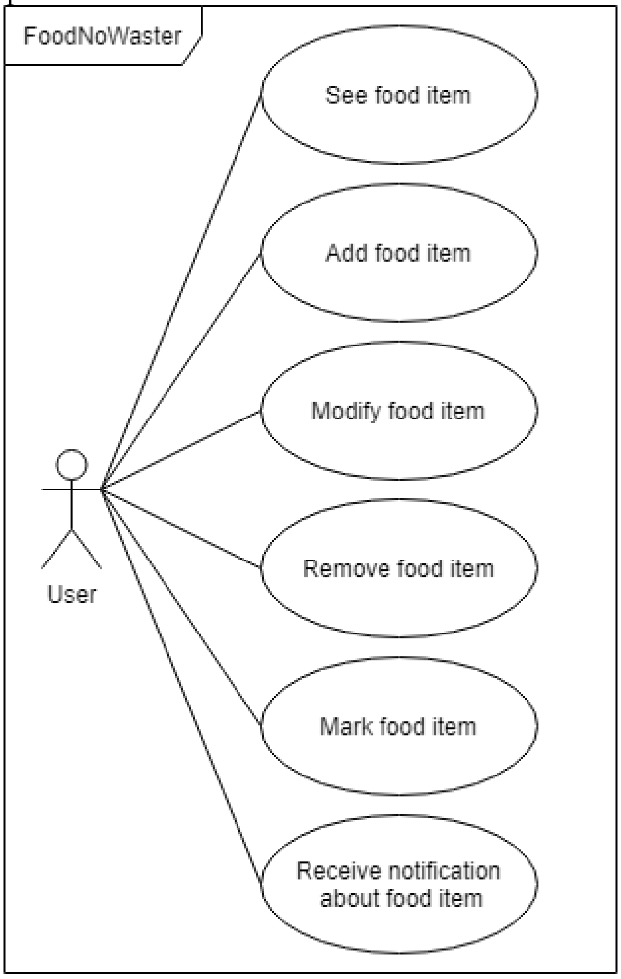
4.1.3.1 SWOT

FoodNoWaste application will have the following strengths: user-oriented service with a good idea behind it, free of charge for the users. Weaknesses are the following: the FoodNoWaste application does not have a strong brand and no investment capital is available. The growing interest in ecological inventions, saving resources, and reducing household expenses can be taken as Opportunities for FoodNoWaste application. Threats are the following: increasing competition from the biggest companies in the market, built-in functionality in alternative solution, Internet of Things (IoT).

4.1.3.2 Use Case

The Use Case diagram for the FoodNoWaster application will have 7 Use Cases. After launching the FoodNoWaster application, the user will be able to see a list of food items, if there are no food items in the database, the user will be asked to add a new food item. The user will always be able to add a new food item. It will be possible to modify an existing food item by changing the name or expiration date of a food item. The user can remove a selected food item, delete a food item, mark a food item, and receive a notification about food items in the database.

Figure 6 FoodNoWaster application Use Case model



4.2 Implementation

4.2.1 Android Studio installation

For the development process of the FoodNoWaste application, I decided to use Android Studio integrated development environment. To start the installation process, I downloaded the Android Studio package from the official Android website for developers [31]. The installation package for Windows OS can be downloaded in two options: as a `.exe` installer or as a `.zip` archive. The `.exe` version downloads SDK during the installation process, while the `.zip` version does not include it. The `.exe` version is preferable for fresh install when SDK is also needed to be downloaded and installed. After the download is completed, and

`.exe` executed, the Android Studio Setup will be opened. It guides the developer through the installation process. In the next step of the setup, the developer can choose if he wants to install an Android Virtual Device – it is recommended configuration that defines the Android device characteristics to make the application testing on the emulator easier. After choosing the standard Windows path and name for the file, the installation process is completed, and the developer can open the Android Studio IDE. With the IDE opening, the Android Studio Setup Wizard will be started. This helps the developer with setting up the development environment. Additionally, it is also possible to port existing Android applications or create a new Android application project. In the next steps of the Wizard, the developer can choose a type of setup (Standard or Custom), developer interface theme (dark or light), and what SDK components to update or install. Then the Wizard will download, execute, and install needed components, such as the Emulator, SDK, SDK tools, and sources.

4.2.2 **Android Studio configuration**

Wizards and templates, that verify the system requirements such as Java Development Kit (JDK) and available RAM and configure default settings such as an optimized Android Virtual Device (AVD) emulation and updated system images are provided by Android Studio. Java Virtual Machine customization options such as heap size and cache size are accessible through the Help menu in the `studio.vmoptions` configuration file. The plugin folder path or maximum supported file size customization is accessible in the `idea.properties` configuration file.

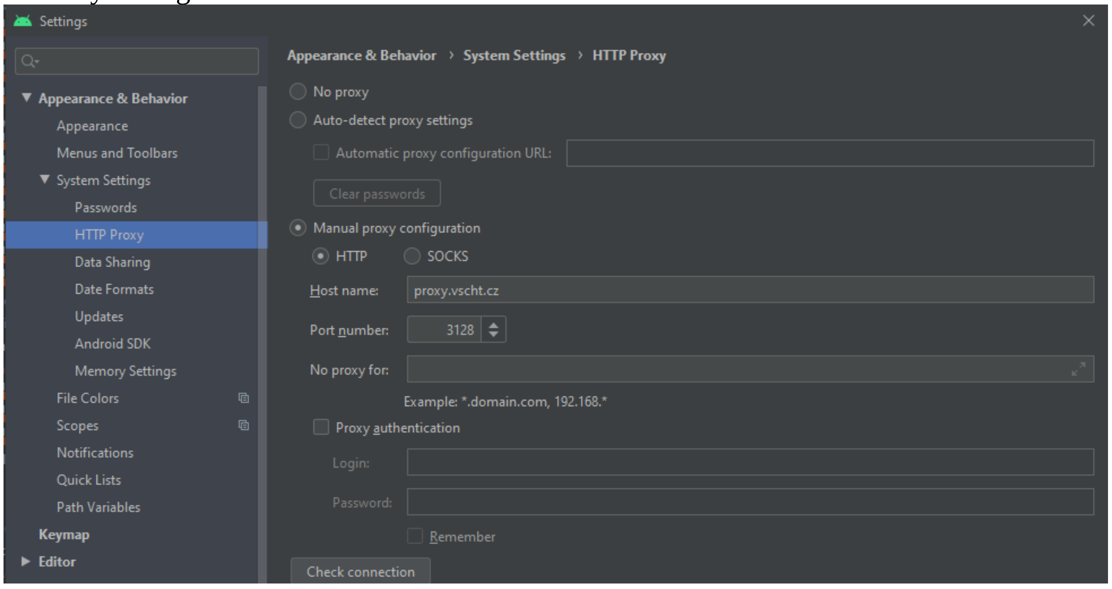
Adjusting the maximum heap size is the most common option to improve the Android Studio performance. However, other default settings such as initial heap size, cache size, and Java garbage collection switches can be overridden using the `studio.vmoptions` configuration file. The file is accessible through the `Help > Edit Custom VM Options` menu. If the VM options for Android Studio were never edited before, the IDE prompts the developer to create a new configuration file. The new file will be added to the bin/ directory inside the Android Studio installation folder. The `studio.vmoptions` should never be directly edited inside the Android Studio program folder to make sure the important Android Studio default settings are not overwritten.

1280MB is a default maximum heap size the Android Studio has. If the developer works on a large project and has a system with a lot of RAM, the performance can be improved by increasing the maximum heap size for Android Studio processes, such as the core IDE, Gradle daemon, and Kotlin daemon. Possible heap size optimizations are automatically checked by Android Studio. If the performance can be approved, the user will receive a notification.

The Android Studio can also be configured for low-memory machines. There are several recommendations on how to improve performance on such machines. The maximum heap size available to Android Studio should be reduced to 512Mb. Gradle and the Android plugin for Gradle should be updated to the latest available version. Enabling Power Save Mode turns off several battery-intensive background applications. Debugging on a physical device will decrease memory consumption, as it uses less memory than debugging on an emulator. Only necessary Google Play dependencies should be included in the project. Parallel compilation should also be disabled on low-memory machines.

Android Studio supports HTTP proxy settings, so the developer can run the IDE even behind a firewall or secure network. Proxy settings can be configured in two ways. First one is from the menu bar: `File > Settings > Appearance & Behaviour > System Settings > HTTP Proxy`. The HTTP proxy page will appear, where the proxy can be set.

Figure 7 Proxy settings in Android Studio



Manual proxy configuration: ● HTTP ○ SOCKS

Host name: proxy.vscht.cz

Port number: 3128

No proxy for:

Example: *.domain.com, 192.168.*

The second option is setting the proxy directly in the `gradle.properties` file, which is located in `app>Gradle Scripts>gradle.properties`.

Figure 8 Proxy settings in gradle.properties file

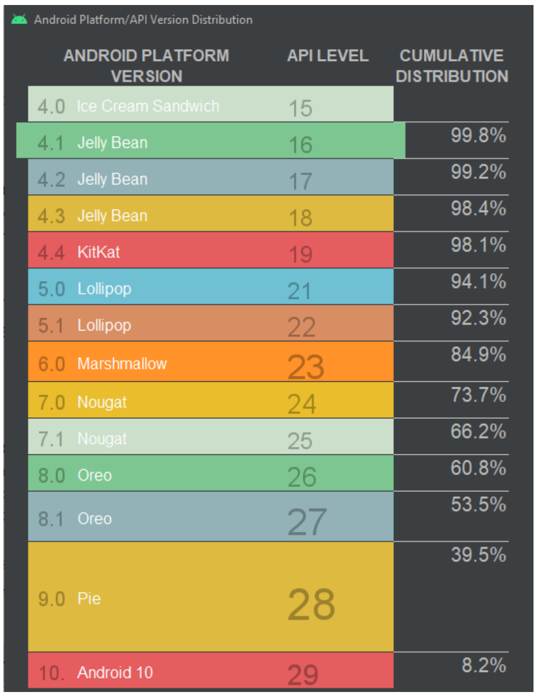
```
systemProp.http.proxyHost=proxy.vscht.cz
systemProp.http.proxyPort=3128
systemProp.https.proxyHost=proxy.vscht.cz
systemProp.https.proxyPort=3128
```

4.2.3 Creating a new Android Studio project

To create a new project in Android Studio, the developer should choose the type of device (Phone and Tablet, Wear OS, Android TV, Automotive, or Android Things) and an *activity* (No Activity, Basic, Bottom Navigation, Empty, Login and others). After that, the developer will be required to configure the project, choose an application, and package names, save location, language (Java or Kotlin), and minimum SDK version. The Android Studio helps the developer to choose a minimum SDK version by providing platform version statistics (cumulative distribution), and information features such as App Components, Multimedia, Camera, Connectivity, Input Framework, and others.

The decision about the minimum SDK version should be chosen based on features that needed to be supported and the percentage of cumulative distribution. By choosing the minimum Android platform version, the developer defines what is the minimum supported version, which reflects how many people will be able to install the application on their devices. Currently, version 4.1. is recommended due to its features and because it has the highest cumulative distribution – 99.8%.

Figure 9 Android platform versions: API level and distribution

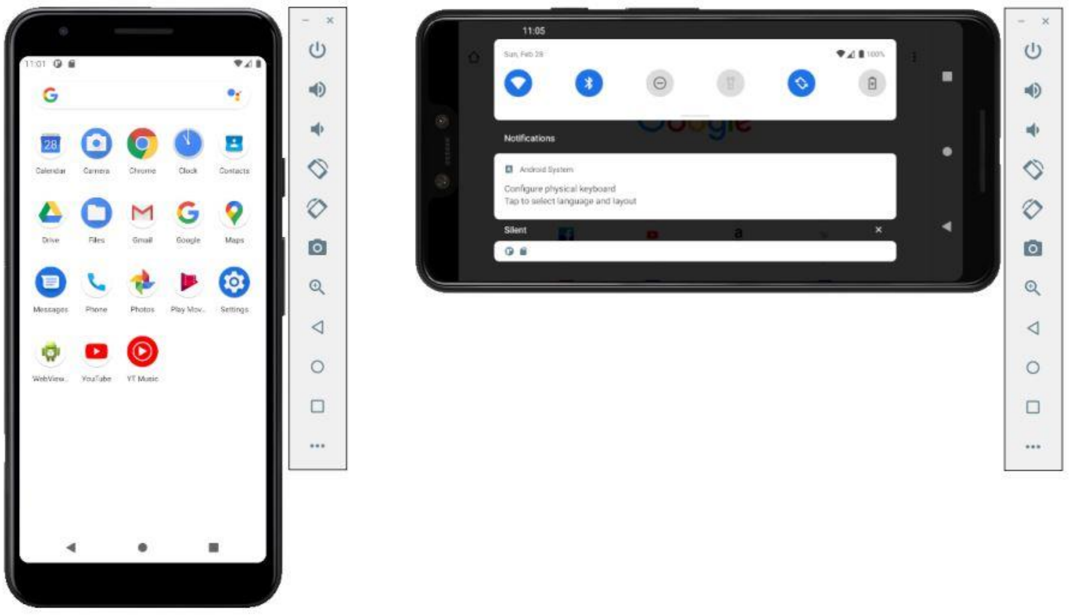


Android Platform/API Version Distribution

ANDROID PLATFORM VERSION		API LEVEL	CUMULATIVE DISTRIBUTION
4.0	Ice Cream Sandwich	15	
4.1	Jelly Bean	16	99.8%
4.2	Jelly Bean	17	99.2%
4.3	Jelly Bean	18	98.4%
4.4	KitKat	19	98.1%
5.0	Lollipop	21	94.1%
5.1	Lollipop	22	92.3%
6.0	Marshmallow	23	84.9%
7.0	Nougat	24	73.7%
7.1	Nougat	25	66.2%
8.0	Oreo	26	60.8%
8.1	Oreo	27	53.5%
9.0	Pie	28	39.5%
10.	Android 10	29	8.2%

When all downloads are completed, and the necessary steps have proceeded, the Android Studio opens the main window. To start the development process, the Project window should be opened, and the Android view should be selected in the drop-down list at the top of that window. The following files are available to the developer in the Project window: `app>java>com.example.myapplication>MainActivity` – this is the entry point of the application, when the developer builds and runs the application, the system launches an instance of the `Activity` class and loads its layout; `app>res>layout>activity_main.xml` is the XML file, which defines the layout for *activity's* user interface (UI); `app>manifests>AndroidManifest.xml` is the *manifest file*, that describes the fundamental characteristics of the application and defines each of its components; folder `Gradle Scriprs>build.gradle` consists the files that control how the Gradle plugin builds the application [39].

Figure 10 Google Pixel 3a and 3 XL emulation in Android Studio

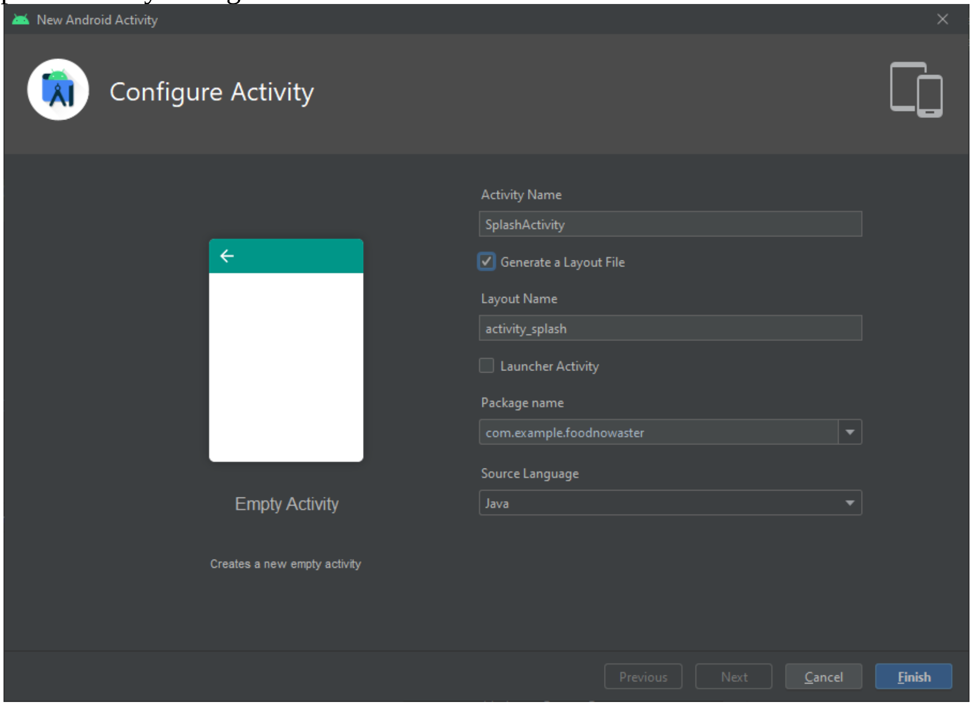


The application can be run on a real device or an emulator. To run on a real device, the device should be connected to the computer with a USB cable. The appropriate drivers for the device are also needed. Firstly, there is a need to enable USB debugging in the Developer options window (Developer options settings). Secondly, the connection should be allowed by the device. The connected device will appear in the drop-down menu in the toolbar (run/debug configurations). After the device is selected and the Run button is clicked, Android Studio installs the application on the connected device and starts it. After the application installation is done, it will be launched on the device. To run the application on an emulator, Android Virtual Device must be created in the AVD Manager. After creation, it should be selected from the drop-down menu in the toolbar (run/debug configurations). By clicking on the Run button, the Android studio installs the application on the AVD and starts the emulator.

4.2.4 Splash screen

First, the application' splash screen will be created, which will be shown to the user during application loading. The splash screen will consist of the application name and food bank icon from Clip Art set provided by Android Studio. To create a splash screen, the `SplashActivity` should be created. `SplashActivity` can be created by right-clicking on the java subfolder of the project, then choose `New>Activity>Empty Activity`. The New Android Activity configuration window will be opened. In that window, the developer can define the Activity Name, generate a Layout File for that *activity*, choose a Package name and a Source Language. A `SplashActivity` class with generated XML layout file should be created.

Figure 11 SplashActivity configuration



Activity Name: SplashActivity
[✓] Generate a Layout File
Layout Name: activity_splash
[ ] Launcher Activity
Package name: com.example.foodnowaster
Source Language: Java

After *Activity* is configured, the Android Studio will automatically create a `SplashActivity` class and activity_splash.xml layout. In the `SplashActivity` class, an *activity* that will show the loading screen for three seconds to the user before entering the Main application screen should be created.

Figure 12 Splash activity loading screen

```
public class SplashActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_splash);

        Handler handler = new Handler();
        handler.postDelayed(new Runnable() {
            @Override
            public void run() {
                startActivity(new Intent(SplashActivity.this,
MainActivity.class));
                finish();
            }
        }, 2500);
    }
}
```

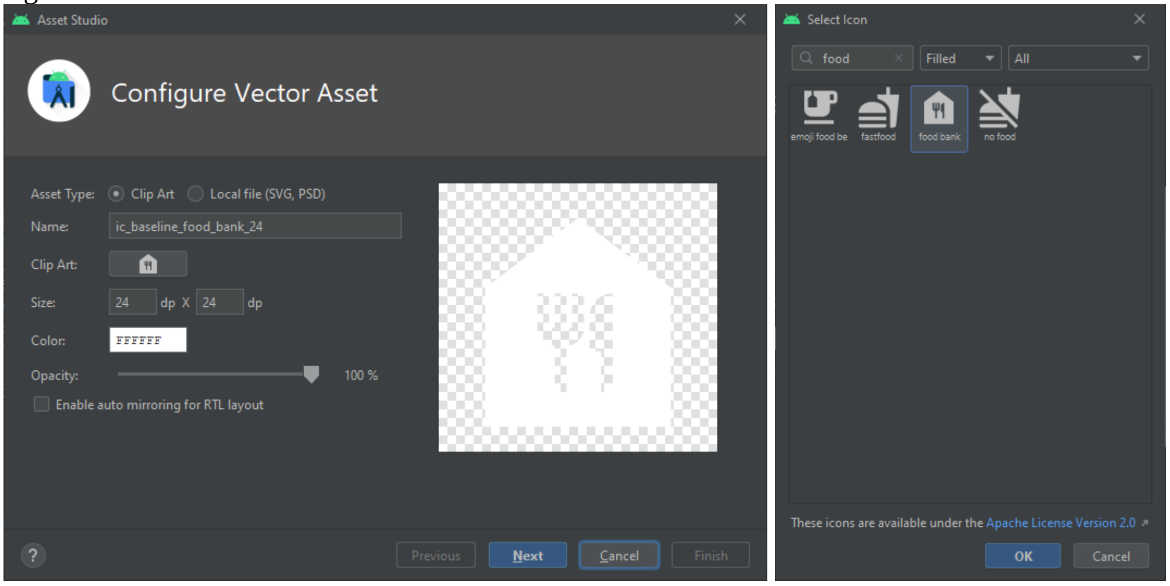
After the `SplashActivity` is created, the `activity_splash.xml` layout file should be configured. The structure for a user interface is defined in that file. The layout will consist of the application name FoodNoWaster and an icon on a green background. To specify colours used through the application, they should be defined in the `colors.xml` file located under the values subfolder in the res folder of the project.

Figure 13 Colours definitions

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <color name="colorPrimary">#FFFFFF</color>
    <color name="colorPrimaryDark">#000000</color>
    <color name="colorAccent">#4CAF50</color>
</resources>
```

To be able to use icons, they should be added to the drawable subfolder of the project. To add an icon, perform a right-click on the drawable folder, then check `New>Vector Asset`. The Asset Studio window will be opened. In that window, the user can choose the Asset Type (preloaded Clip Art or a Local source file), define the Name, choose a Clip Art, define a Size of the icon, select a Colour, and define the Opacity. The Select Icon window will be opened by clicking on Clip Art, where the developer will be able to choose an icon using the keyword and theme. The filling and shape types can be also chosen in that window.

Figure 14 Icon selection in Android Studio



Asset Studio

Configure Vector Asset

Asset Type: [✓ Clip Art] Local file (SVG, PSD)

Name: ic_baseline_food_bank_24

Clip Art:

Size: 24 dp X 24 dp

Color: FFFFFF

Opacity: 100 %

Enable auto mirroring for RTL layout

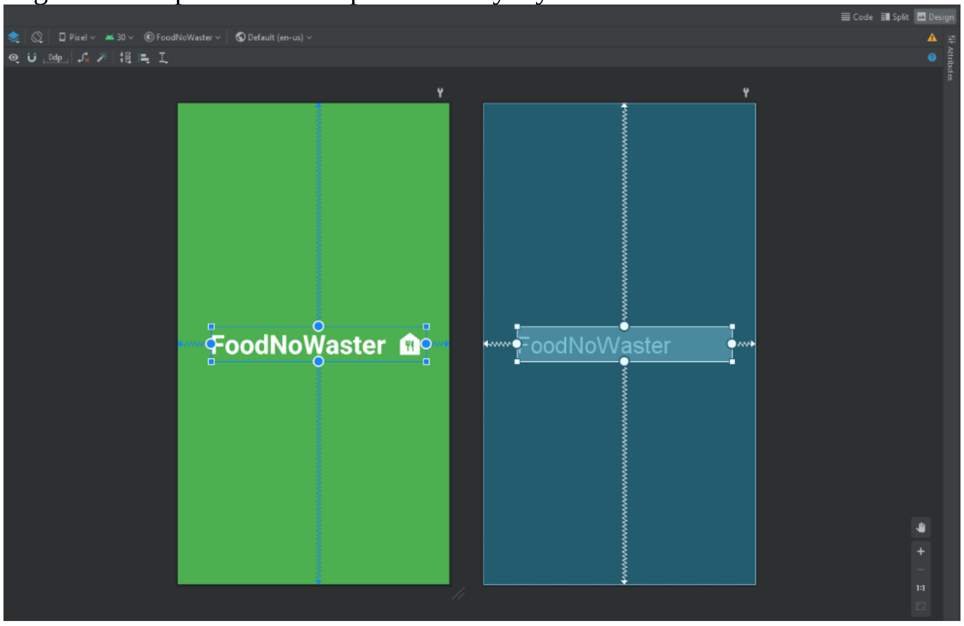
Select Icon

food | Filled | All

These icons are available under the Apache License Version 2.0

After the icons are chosen and corresponding XML files are created in the `drawable` folder, the layout of the `SplashActivity` can be finished by using the XML attributes. The design and blueprint views are showing the developer how elements and views interact with each other. The blueprint can be very useful when `View` is invisible, it will be seen in the blueprint view, but it will not be visible in the preview.

Figure 15 Design and Blueprint view of Splash activity layout

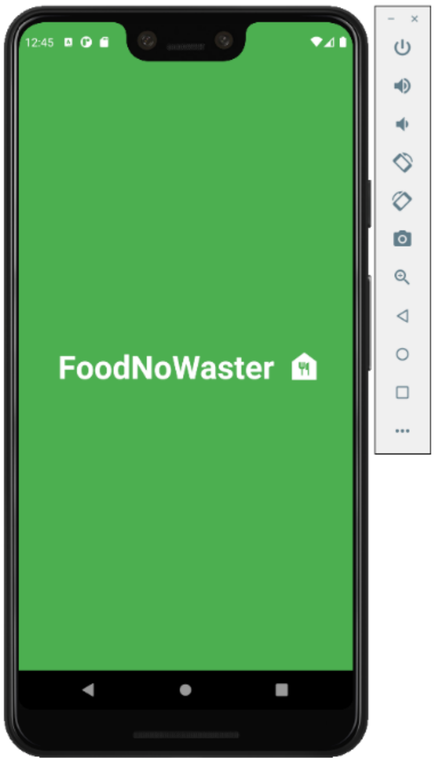


To run the Splash screen while the application is loading, the `AnidroidManifest.xml` file should be changed. To do that, the `<intent-filter>` attribute with its content should be moved to the `SplashActivity` *activity*. After that change, the Splash screen will be executed and displayed to the user every time he opens the application.

Figure 16 Splash activity definition in manifest file

```
<activity android:name=".SplashActivity">
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
```

Figure 17 FoodNoWaster splash screen in Android Studio Emulator



Sometimes the scope of supported layout XML `attributes` can be changed over API versions. In this case, the Android Studio will display a hint about the steps that are needed to support the specified version. For example, the XML attribute `marginEnd` which specifies extra space on the end side of the view is not supported in the older API versions than 17.

Figure 18 XML attribute example

```
android:layout_marginEnd="10dp";
```

The Android Studio will alert the developer about this fact and will offer an appropriate solution.

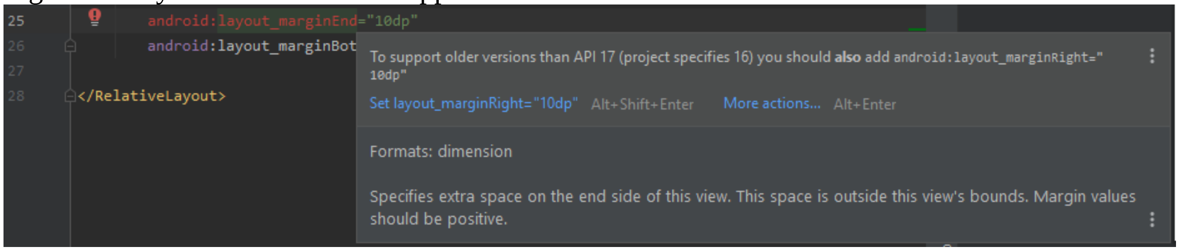


Figure 19 Layout XML attribute support alert

```
        android:layout_marginEnd="10dp"
        android:layout_marginBot
</RelativeLayout>
```

To support older versions than API 17 (project specifies 16) you should **also** add `android:layout_marginRight="10dp"`

Set layout_marginRight="10dp" Alt+Shift+Enter More actions... Alt+Enter

Formats: dimension

Specifies extra space on the end side of this view. This space is outside this view's bounds. Margin values should be positive.

In this case, the Android Studio will offer to add a new XML attribute called layout_marginRight, which specifies extra space on the right side of the view.

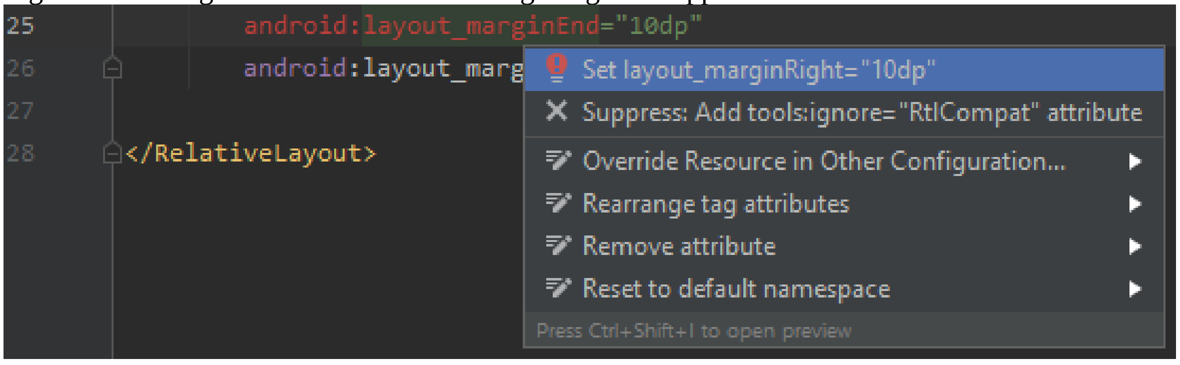


Figure 20 Adding a new XML attribute marginRight to support older API versions

```
        android:layout_marginEnd="10dp"
        android:layout_marg
</RelativeLayout>
```

Set layout_marginRight="10dp"
Suppress: Add tools:ignore="RtlCompat" attribute
Override Resource in Other Configuration...
Rearrange tag attributes
Remove attribute
Reset to default namespace
Press Ctrl+Shift+I to open preview

As a result, a new XML attribute called layout_`marginRight` will be added to the existing layout_`marginEnd` attribute to support API versions older than version API 17.

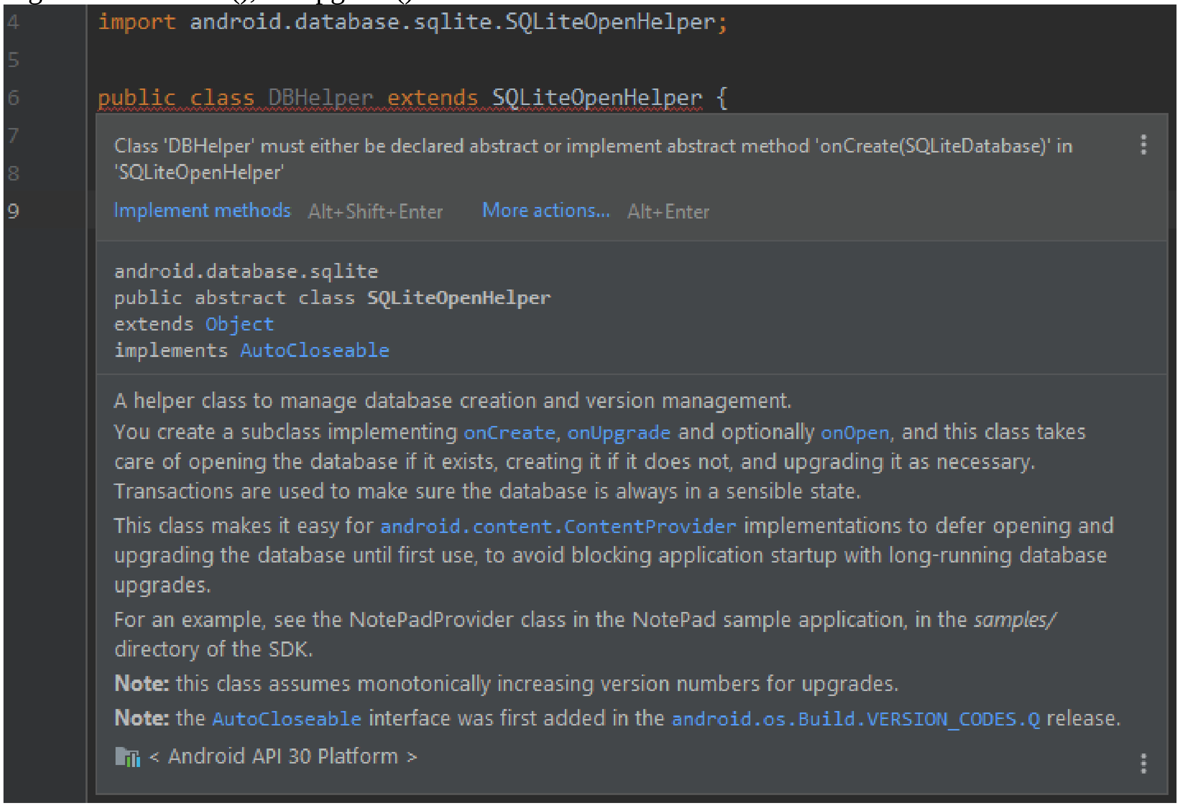
Figure 21 New XML attribute marginBottom added

```
android:layout_marginEnd="10dp"
android:layout_marginBottom="10dp"
```

4.2.5 SQLite database

The food items will be stored in the device memory. A `DBHelper` class is created for items storing. This class is inherited from the `SQLiteOpenHelper` class, which is a helper class to manage database creation and version management. This class takes care of opening the database if it exists, creating it if it does not, and updating it, as necessary. This method always returns very quickly. The database is not actually created or opened until one of `getWritableDatabse()` or `getReadableDatabase()` methods is called. Transactions are used to make sure the database is always in a sensible state. Android Studio offers an automatic construction for `onCreate()` and `onUpgrade()` methods.

Figure 22 onCreate(), onUpgrade() methods automatic construction in Android Studio



```
import android.database.sqlite.SQLiteOpenHelper;

public class DBHelper extends SQLiteOpenHelper {
```

Class 'DBHelper' must either be declared abstract or implement abstract method 'onCreate(SQLiteDatabase)' in 'SQLiteOpenHelper'

```
android.database.sqlite
public abstract class SQLiteOpenHelper
extends Object
implements AutoCloseable
```

A helper class to manage database creation and version management.
You create a subclass implementing `onCreate`, `onUpgrade` and optionally `onOpen`, and this class takes care of opening the database if it exists, creating it if it does not, and upgrading it as necessary. Transactions are used to make sure the database is always in a sensible state.
This class makes it easy for `android.content.ContentProvider` implementations to defer opening and upgrading the database until first use, to avoid blocking application startup with long-running database upgrades.
For an example, see the NotePadProvider class in the NotePad sample application, in the *samples/* directory of the SDK.
Note: this class assumes monotonically increasing version numbers for upgrades.
Note: the `AutoCloseable` interface was first added in the `android.os.Build.VERSION_CODES.Q` release.
< Android API 30 Platform >

Before implementation of the methods, the `DATABASE_NAME` and the `TABLE_NAME` variables should be specified to store the names of the database and table.

Figure 23 Database name and table name specification

```
public static final String DATABASE_NAME = "FoodNoWasterDBHelper.db";
public static final String TABLE_NAME = "foodItems";
```

Also, a public constructor `SQLiteOpenHelper(Context context, String name, SQLiteDatabase.CursorFactory factory, int version)` should be used to create a helper object to create, open, and manage a database. Context parameter is used for paths locating to the database. The parameter name is the name of the database file. Factory parameter is used for creating `Cursor` objects. A parameter version is an integer number of the database. It is used for database upgrade or downgrade. Context, name, and factory parameters may have a null value.

Figure 24 DBHelper class initialization

```
public DBHelper(Context context) {
    super(context, DATABASE_NAME, null, 1);
}
```

After the public constructor is defined and variables are set, the `onCreate()` and `onUpgrade()` methods can be implemented. Method `onCreate()` is called when the database is created for the first time. The creation of tables and the initial population of the tables should happen in this method. The database table will have 4 columns: `id`, `foodItem`, `expirationDate`, and `status`. `Id` column will store integer values and will increment automatically. `FoodItem` column will contain food item details. The expiration date of the food item will be stored in the `expirationDate` column. The `Status` column will be used for storing the information on whether the food item is consumed or not. For unconsumed items, the value will be equal to 0, for consumed to 1.

Figure 25 onCreate() method creation

```
@Override
public void onCreate(SQLiteDatabase db) {
    db.execSQL(
            "CREATE TABLE " + TABLE_NAME + "(id INTEGER PRIMARY KEY,
foodItem TEXT, expirationDate DATETIME, status INTEGER)"
    );
}
```

The `onUpgrade()` method is called when the database needs to be upgraded. The implementation should use this method to drop tables, add tables, or do everything that it needs to upgrade the database to the new schema version. This method is executed within a transaction. All changes will be automatically rolled back if an exception is thrown.

Figure 26 onUgrade() method creation

```
@Override
public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion)
{
    db.execSQL("DROP TABLE IF EXISTS " + TABLE_NAME);
    onCreate(db);
}
```

The `insertFoodItem()` method will be used to insert a food item into the database. It inserts food item name, expiration date and status into the `foodItems` table.

Figure 27 insertFoodItem() method creation

```
public boolean insertFoodItem(String foodItem, String expirationDate) {
    Date date;
    SQLiteDatabase db = this.getWritableDatabase();
    ContentValues contentValues = new ContentValues();
    contentValues.put("foodItem", foodItem);
    contentValues.put("expirationDate", expirationDate);
    contentValues.put("status", 0);
    db.insert(TABLE_NAME, null, contentValues);
    return true;
}
```

Similarly, the `updateFoodItem()` method will be used to update the food item name and the expiration date in the `foodItems` table. The `deleteFoodItem()` method will delete a food item with the received id.

Figure 28 updateFoodItem() method creation

```
public boolean updateFoodItem(String id, String foodItem, String
expirationDate) {
    SQLiteDatabase db = this.getWritableDatabase();
    ContentValues contentValues = new ContentValues();
    contentValues.put("foodItem", foodItem);
    contentValues.put("expirationDate", expirationDate);
    db.update(TABLE_NAME, contentValues, "id = ? ", new String[]{id});
    return true;
}

public boolean deleteFoodItem(String id) {
    SQLiteDatabase db = this.getWritableDatabase();
    db.delete(TABLE_NAME, "id = ? ", new String[]{id});
    return true;
}
```

Also, the method for updating a food item status is needed to be implemented. The method called `updateFoodItemStatus()` will handle a status update of the food item. It updates the status of the provided food item id.

Figure 29 updateFoodItemStatus() method creation

```
public boolean updateFoodItemStatus(String id, Integer status) {
    SQLiteDatabase db = this.getWritableDatabase();
    ContentValues contentValues = new ContentValues();
    contentValues.put("status", status);
    db.update(TABLE_NAME, contentValues, "id = ? ", new String[]{id});
    return true;
}
```

Later, there will be a need for retrieving the food items from the database for displaying in the user interface. For that reason, four methods are needed: `getSingleFoodItem()`, `getTodayFoodItem()`, `getTomorrowFoodItem()`, and `getLaterFoodItem()`. These methods implement SQLite database select queries. As a result, it will be possible to get food items as a single food item, result-sets of food items with today's, tomorrow's, and upcoming expiration dates. Today's food items will be ordered by the food item name, while the upcoming by the expiration date

Figure 30 Methods for retrieving the food items from database

```
public Cursor getSingleFoodItem(String id) {
    SQLiteDatabase db = this.getReadableDatabase();
    Cursor res = db.rawQuery("select * from " + TABLE_NAME + " WHERE id
= '" + id + "' order by foodItem desc", null);
    return res;
}

public Cursor getTodayFoodItem() {
    SQLiteDatabase db = this.getReadableDatabase();
    Cursor res = db.rawQuery("select * from " + TABLE_NAME + " WHERE
date(expirationDate) = date('now', 'localtime') order by id desc",
null);
    return res;
}

public Cursor getTomorrowFoodItem() {
    SQLiteDatabase db = this.getReadableDatabase();
    Cursor res = db.rawQuery("select * from " + TABLE_NAME + " WHERE
date(expirationDate) = date('now', '+1 day', 'localtime')  order by id
desc", null);
    return res;
}

public Cursor getLaterFoodItem() {
    SQLiteDatabase db = this.getReadableDatabase();
    Cursor res = db.rawQuery("select * from " + TABLE_NAME + " WHERE
date(expirationDate) > date('now', '+1 day', 'localtime') order by
expirationDate asc", null);
    return res;
}
```

4.2.6 **Custom ListView class**

After the database helper class `DBHelper` is ready, the custom `ListView` class should be implemented. A new Java class called `NoScrollListView` should be created. This class is a subclass of the `ListView` class that is not scrollable. `NoScrollListView` class will be used by the `MainActivity` class to display all available food items.

Figure 31 NoScrollListView class implementation

```
public class NoScrollListView extends ListView {

    public NoScrollListView(Context context) {
        super(context);
    }
    public NoScrollListView(Context context, AttributeSet attrs) {
        super(context, attrs);
    }
    public NoScrollListView(Context context, AttributeSet attrs, int
defStyle) {
        super(context, attrs, defStyle);
    }
    @Override
    public void onMeasure(int widthMeasureSpec, int heightMeasureSpec) {
        int heightMeasureSpec_custom = MeasureSpec.makeMeasureSpec(
                Integer.MAX_VALUE >> 2, MeasureSpec.AT_MOST);
        super.onMeasure(widthMeasureSpec, heightMeasureSpec_custom);
        ViewGroup.LayoutParams params = getLayoutParams();
        params.height = getMeasuredHeight();
    }
}
```

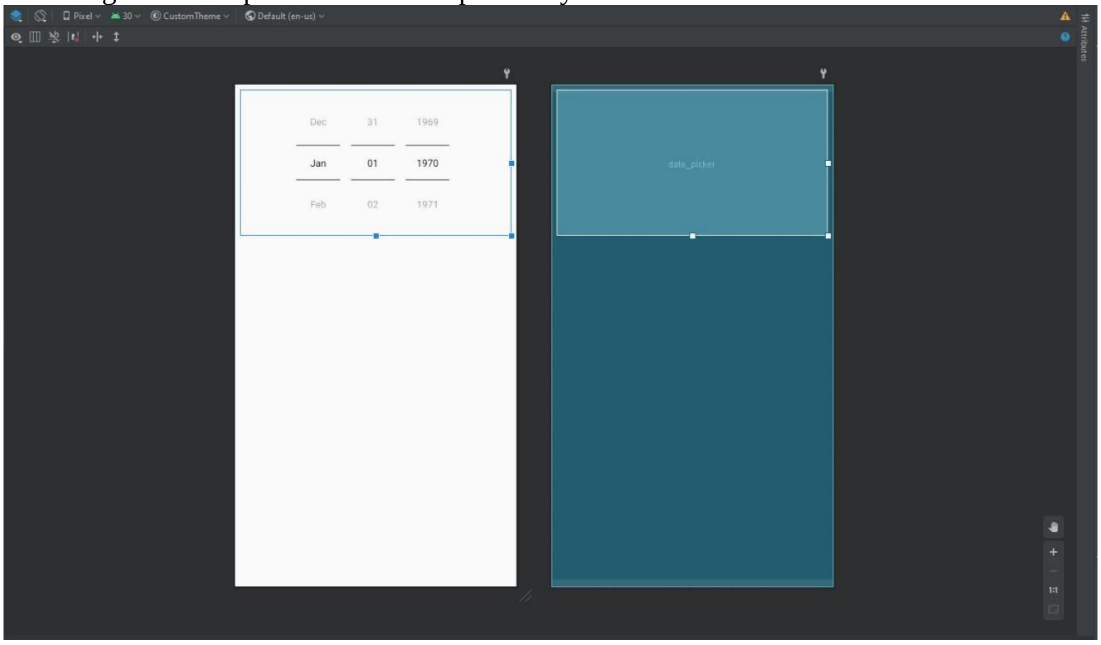
4.2.7 **Date picker layout**

Android provides controls for the user to pick a date as ready-to-use dialogues. A date picker will be used as a spinner for selecting a date. A picker ensures that the user can pick a date that is valid, correctly formatted, and adjusted to the user's locale. The date will be selected using day, month, and year spinners. Firstly, a `date_picker` XML layout file should be created in the res subfolder of the project folder. The `date_picker` will be displayed as a spinning picker in the upper part of the screen with 10 density-independent pixel units (dp) from each part of the view.

Figure 32 Date picker XML layout

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:padding="10dp">
    <DatePicker
        android:id="@+id/date_picker"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:calendarViewShown="false"
        android:datePickerMode="spinner" />
</LinearLayout>
```

Figure 33 Design and Blueprint view of date picker layout

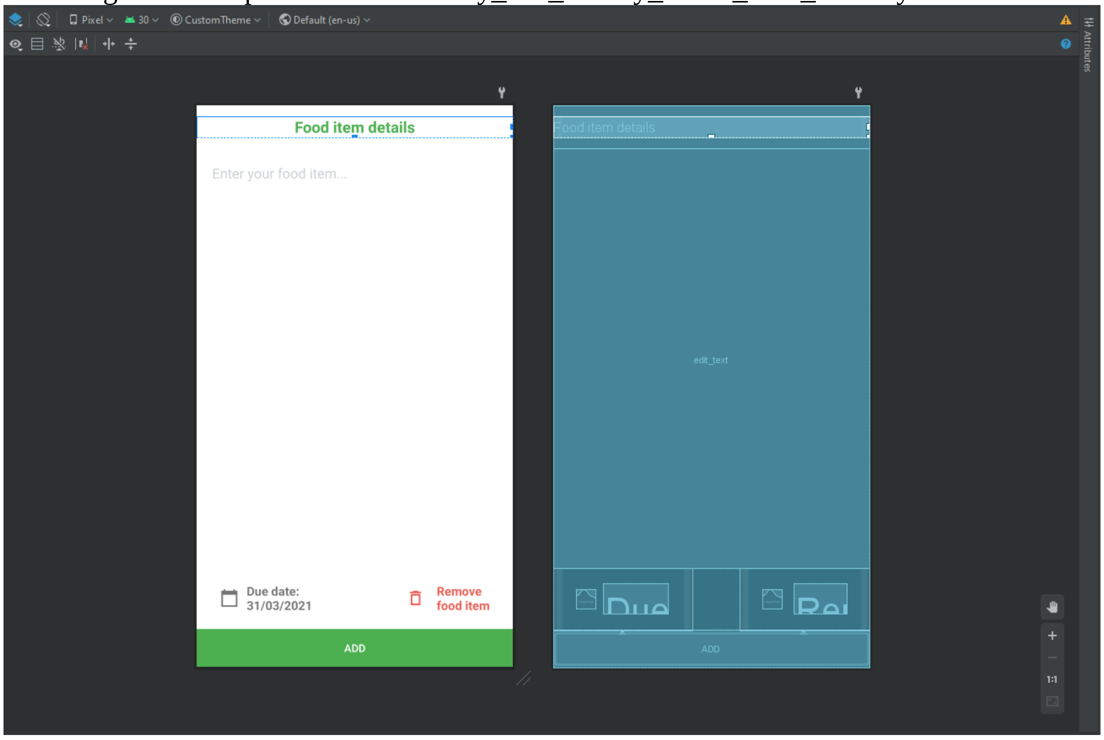


4.2.8 **AddModifyDeleteFoodItem class**

Adding, modifying, and deleting features are the core features of the FoodNoWaster application. All these functions will be implemented in one *activity* called `AddModifyDeleteFoodItem`. If the food item already exists, the *activity* will act like Modify food item and Delete food item, otherwise, it will act like Add new food item. To implement this class, a new Empty *activity* called `AddModifyDeleteFoodItem` should be created. By adding a new Empty *activity*, the Android Studio will automatically create a new Java class `AddModifyDeleteFoodItem` in the java subfolder and an XML layout file called `activity_add_modify_delete_food_item` in the res subfolder under the project folder.

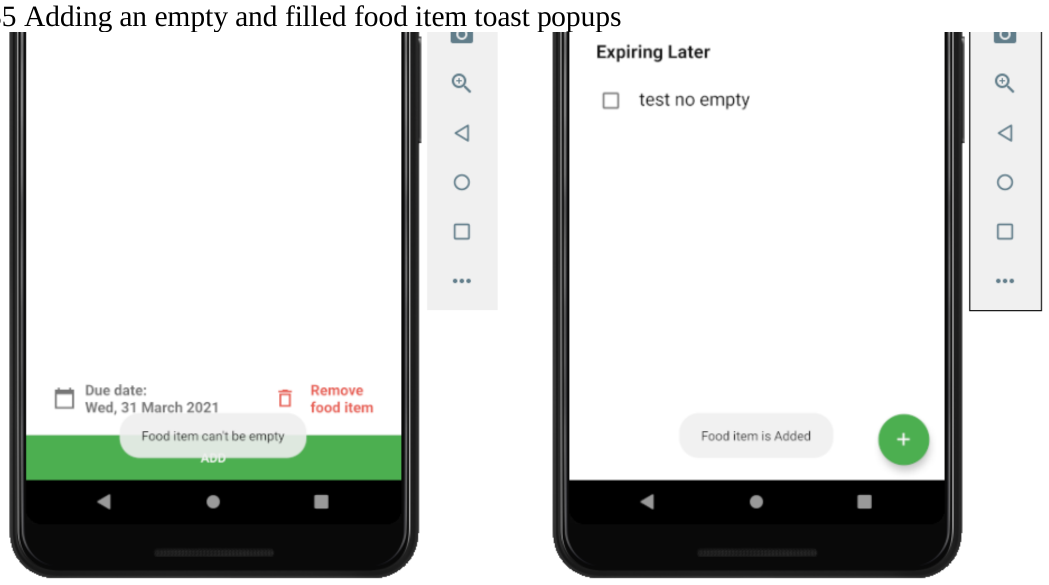
The layout of the `AddModifyDeleteFoodItem` class will be defined in the XML file `activity_add_modify_delete_food_item`. The upper part of the screen layout will consist of green coloured "Food item details" title, and the grey "Enter your food item…" hint in the `EditText` area. The bottom part will have the Vector Asset Clip Art "calendar today” icon with the title “Due date:” and the default date “31/03/2021”, the red coloured Vector Asset Clip Art “delete” icon with the title “Remove food item”. At the very bottom of the screen layout, the white “ADD” button will be placed on the green coloured background.

Figure 34 Design and Blueprint view of activity_add_modify_delete_food_item layout



The `AddModifyDeleteFoodItem` Java class will call different functions written in `DBHelper` class. The modification, creation, and deleting of the food item will be handled in this class. The `init_modify()` function will allow the user to change a food item name or expiration date. The expiration date change will be handled by the `chooseDate()` method, which will use a `date_picker` layout to display a date picker. The user will be able to choose a new date using this picker. The `saveFoodItem()` function will handle the saving of the food item into the database. This function uses two methods of the `DBhelper` class. The `updateFoodItem()` method is called when an existing food was modified and should be updated in the database. When the item is modified, the user will receive a small toast popup with the text “Food item is Updated” as feedback. The `insertFoodItem()` method is used to insert a new food item into the database when the item is first created. After the new item is added into the database, the user will receive a small toast popup feedback with the text “Food item is Added “. If the user will try to save an empty food item, a small toast popup with the text “Food item can't be empty” will be triggered, and the food item will not be inserted or updated. The `deleteFoodItem()` function will handle the food item delete operation. It will call a `deleteFoodItem()` method of the `DBHelper`, which will delete the food item from the database. After the food item is deleted, the user will receive a small toast popup with the text “Food item is Removed” as feedback.

Figure 35 Adding an empty and filled food item toast popups

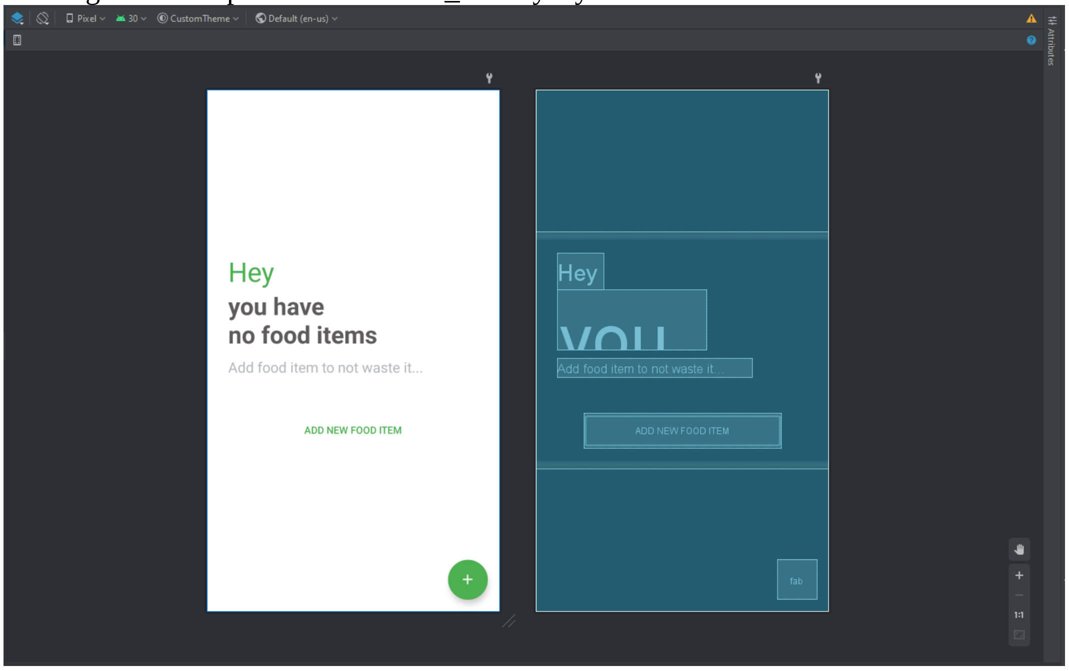


4.2.9 **MainActivity class**

The `MainActivity` will be launched right after the loading splash screen. It will show the user the list of food items and divide them into three categories. “Expiring today” – here will be placed food items with the expiration date equal to the current date. The next category is “Expiring Tomorrow”, which will consist of food items that will expire tomorrow. The last category is “Expiring Later”, where food items with an expiration date of more than 2 days will be placed. If the user has no food items with a close expiration date in the database, the main screen will consist of a greeting for the user.

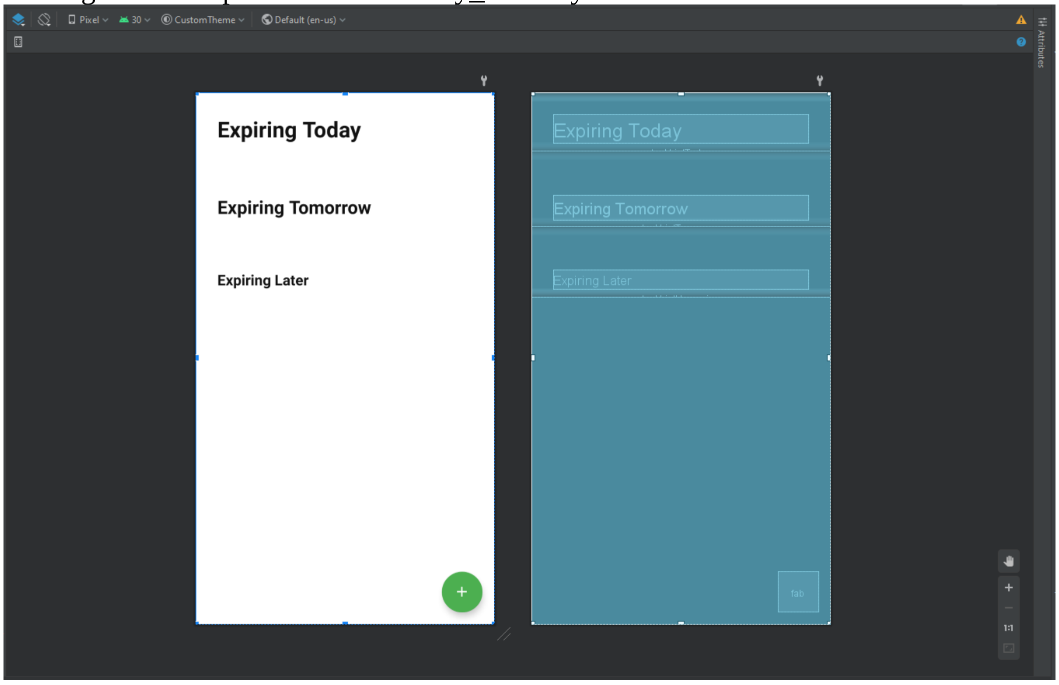
The main screen with a greeting will consist of the main title “Hey you have no food” with a grey hint under the main title telling “Add food item to not waste it…”. Under the hint, a green coloured button with the text “ADD NEW FOOD ITEM” will be placed. In the right bottom corner of the view, a green coloured floating action button will be added. This button will also allow the user to add a new food item.

Figure 36 Design and Blueprint view of main_activity layout with no items



If the user has put food items with an active expiration date in the database, the main screen will consist of three categories with active food items under the categories title. The checkbox describing the food item status will be placed on the left side of the food item' name. Also, a floating action button, which lets the user add a new food item, will be placed in the bottom right corner of the view.

Figure 37 Design and Blueprint view of activity_main layout

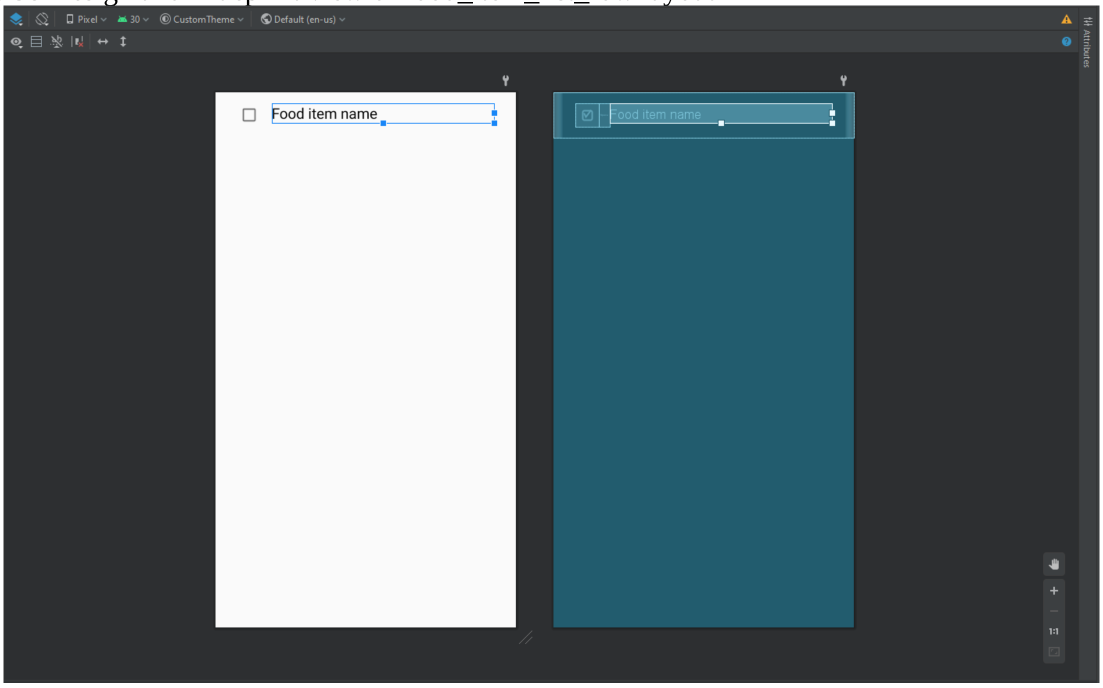


The `MainActivity` class will handle the usage of the `AddModifyDeleteFoodItem` class which will be done using the `openAddModifyDeleteFoodItem()` method, onResume method will be called after resuming the application and re-set the data from the database using the `populateData()` function, which will retrieve data from the database using the `fetchDataFromDB()` method. This method will populate the `ArrayLists` with the data from the database. The `fetchDataFromDB()` method will also set visibility for today, tomorrow, and upcoming list containers. The layout will be visible only if the list is not empty. `DataList` and `ListView` will be created using the `loadDataList()` and the `loadListView()` methods accordingly. The `loadDataList()` method will create a HashMap table and will populate it with data from the `ArrayList`. The `setAlarm()` method, which is used to set a repeated *alarm* is also a part of the `MainActivity` class.

4.2.10 **ListFoodItemAdapter**

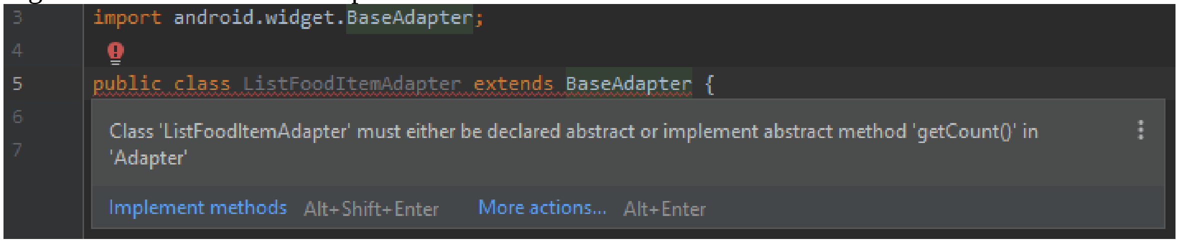
To display food items in `ListView`, the custom `ListView` adapter needs to be implemented. Firstly, the `food_item_list_row` should be implemented to handle the food item name and checkbox which defines their status.

Figure 38 Design and Blueprint view of food_item_list_row layout



After the XML layout is ready, the `ListFoodItemAdapter` Java class should be created in the java subfolder of the project folder. This class will handle the `ListView`, which will display a list of food items and a checkbox with their status. The `ListFoodItemAdapter` class inherits from the `BaseAdapter` class.

Figure 39 Automatic class implementation in Android Studio

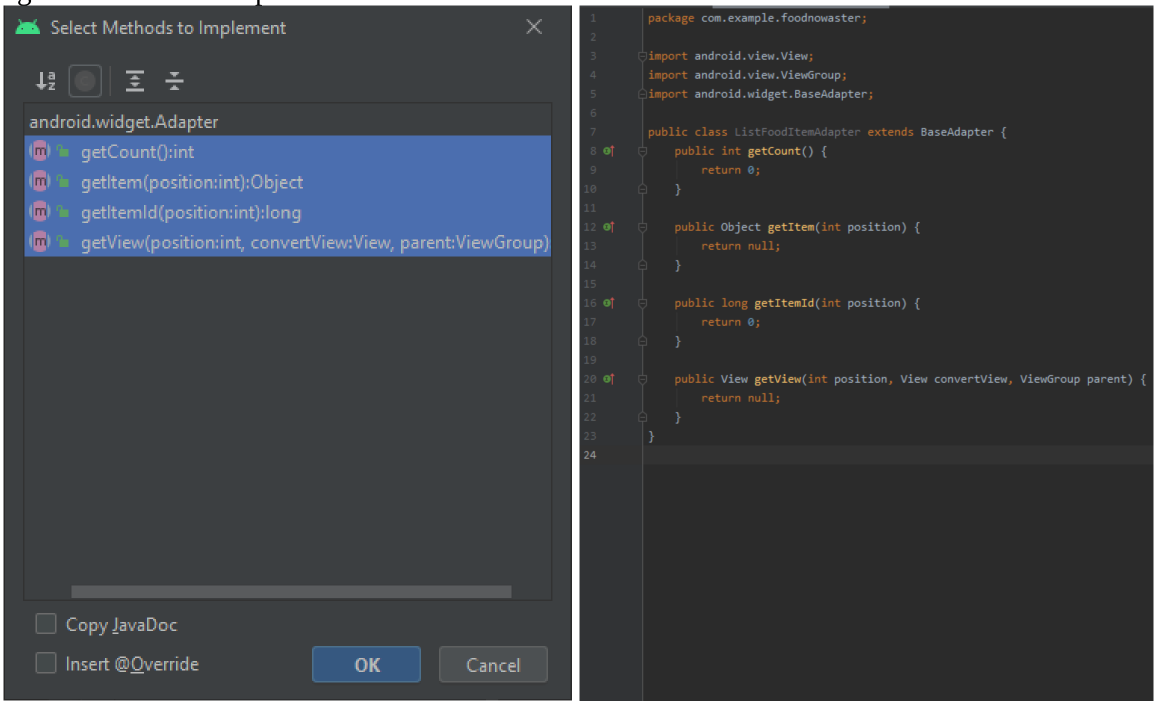


```
import android.widget.BaseAdapter;

public class ListFoodItemAdapter extends BaseAdapter {
```

Class 'ListFoodItemAdapter' must either be declared abstract or implement abstract method 'getCount()' in 'Adapter'

The Android Studio alerts the developer that the class which is inherited from the `BaseAdapter` should implement the abstract method `getCount()`. It also offers to implement selected methods.

Figure 40 Methods implementation in Android Studio



```
android.widget.Adapter
getCount():int
getItem(position:int):Object
getItemId(position:int):long
getView(position:int, convertView:View, parent:ViewGroup)
```

```
package com.example.foodnowaster;

import android.view.View;
import android.view.ViewGroup;
import android.widget.BaseAdapter;

public class ListFoodItemAdapter extends BaseAdapter {
    public int getCount() {
        return 0;
    }

    public Object getItem(int position) {
        return null;
    }

    public long getItemId(int position) {
        return 0;
    }

    public View getView(int position, View convertView, ViewGroup parent) {
        return null;
    }
}
```

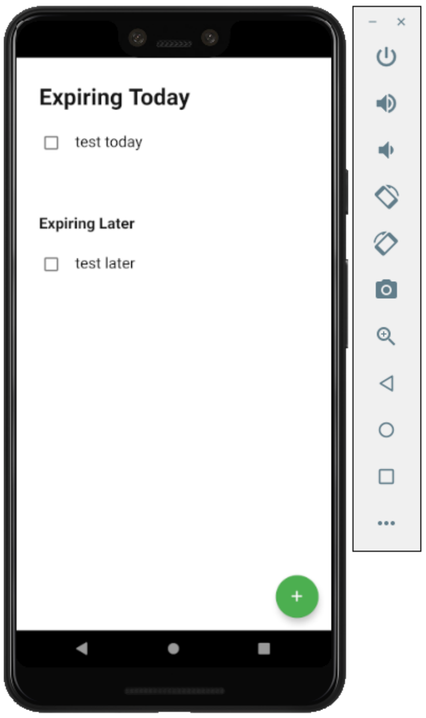
After methods are created, the methods should be implemented. The `getItem()` and `getItemId()` methods will return the position, while `getCount()` method will return the total number of elements from the array list. The `getCount()` function returns the total number of food items to be displayed in a list.

Figure 41 getCount(), getItem(), and getItemId() methods implementation

```
public int getCount() {
    return data.size();
}

public Object getItem(int position) {
    return position;
}

public long getItemId(int position) {
    return position;
}
```

When the list item view is ready to be displayed or about to be displayed, the `getView()` function is automatically called. The layout for displaying food items is set in the `getView()` method. When the `View` is ready, the main screen will be displayed to the user. This screen will contain lists of food items combined into sections by expiration dates (Expiring Today, Expiring Tomorrow, Expiring Later).

Figure 42 FoodNoWaster main screen ListView with sections

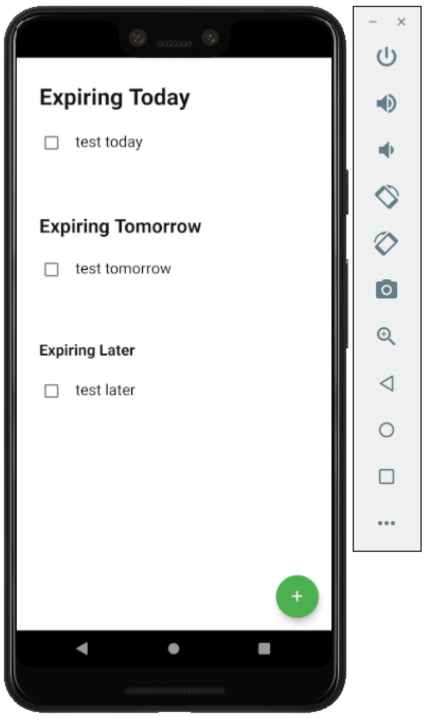


Expiring Today

☐ test today

Expiring Later

☐ test later



Expiring Today

☐ test today

Expiring Tomorrow

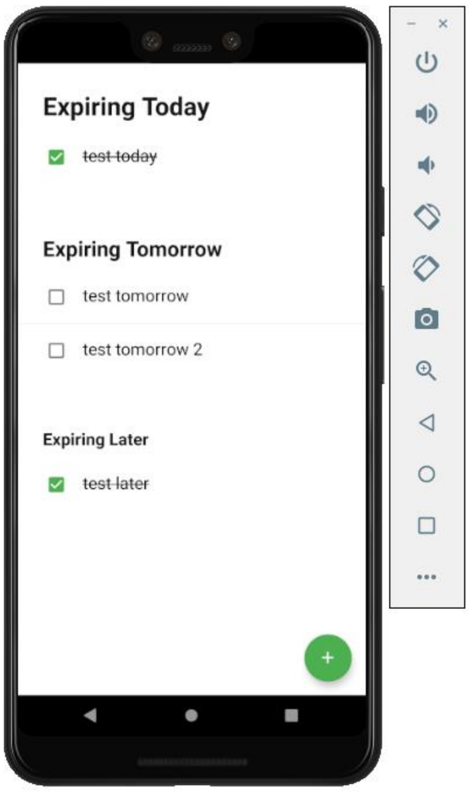
☐ test tomorrow

Expiring Later

☐ test later

The food items inside the sections will be ordered in two ways. In the “Expiring Today” and the “Expiring Tomorrow” lists, they will be ordered by name, while the “Expiring Later” list will be ordered by food items’ expiration date. There also will be a checkbox that will define the status of the food item. When it is checked, the food item is marked as non-active, and the food item name will be decorated with a stroke line.

Figure 43 Food items order and status checkboxes



4.2.11 **Notifications**

To increase the usability of the FoodNoWaster application, the user will be provided with a daily notification with a total number of the food items with the expiration date within the next two days. To do that, a new Java class called `MyAlarm` should be created in the java subfolder in the project folder. This class will be inherited from the `BroadcastReceiver` class and it will make it possible to start an *alarm* even when the application is not running.

First, the total number of food items with the expiration date within the next two days should be calculated using the data from the database.

Figure 44 Food items number calculation with expiration date within next 2 days

```
DBHelper mydb;
mydb = new DBHelper(context);
Cursor today = mydb.getTodayFoodItem();
Cursor tomorrow = mydb.getTomorrowFoodItem();
int upcomingItemsInTwoDays = today.getCount() + tomorrow.getCount();
```

After the total number is calculated, a `NotificationCompat` object should be defined to access features in Notifications.

Figure 45 NotificationCompat object creation

```
String message = "FoodNoWaster notification";

NotificationCompat.Builder builder = new
NotificationCompat.Builder(context, "1")
        .setSmallIcon(R.drawable.food_bank_notification)
        .setContentTitle("Food items expiring within next two days:")
        .setContentInfo(String.valueOf(upcomingItemsInTwoDays))
        .setContentText(String.valueOf(upcomingItemsInTwoDays));
```

To be able to launch the application when the user will click on the notification, the `resultIntent` and `resultPendingIntent` should be created.

Figure 46 resultIntent creation

```
Intent resultIntent = new Intent(context.getApplicationContext(),
MainActivity.class);
PendingIntent resultPendingIntent = PendingIntent.getActivity(context,
1, resultIntent, PendingIntent.FLAG_UPDATE_CURRENT);
```

The `resultPendingIntent` should be passed to the `NotificationCompat` builder.

Figure 47 resultPendingIntent passing to the NotificationCompat

```
.setContentIntent(resultPendingIntent);
```

When the `NotificationCopmat` builder is ready, the behaviour of the notification should be defined. The approach is different for different API versions. For systems with API version 26 and newer, the NotificationChannel will be created, while for older API versions a `NotificationManagerCompat` will be created.

Figure 48 NotificationChannel support for different API versions

```
if(Build.VERSION.SDK_INT>= Build.VERSION_CODES.O) {
    NotificationChannel channel = new NotificationChannel("1",
"FoodNoWasterNumber", NotificationManager.IMPORTANCE_DEFAULT);
    channel.setDescription("FoodNoWasterNumber");
    NotificationManager manager = (NotificationManager)
context.getSystemService(Context.NOTIFICATION_SERVICE);
    manager.createNotificationChannel(channel);
    manager.notify(1, builder.build());
}else {
    NotificationManagerCompat managerCompat =
NotificationManagerCompat.from(context);
    managerCompat.notify(1, builder.build());
};
```

The notification will be woken by the repeating alarm. A pending *intent* will fire the alarm when it is triggered. The notification will be triggered and displayed to the user every day at 9 o'clock in the morning.

Figure 49 Alarm setting in the MainActivity class

```
private void setAlarm() {
    //getting the alarm manager
    AlarmManager alarmMgr = (AlarmManager)
getSystemService(Context.ALARM_SERVICE);

    //creating a new intent specifying the broadcast receiver
    Intent intent = new Intent(this, MyAlarm.class);

    //creating a pending intent using the intent
    PendingIntent alarmIntent = PendingIntent.getBroadcast(this, 0,
intent, 0);

    // Set the alarm to start at 9:00 a.m.
    Calendar calendar = Calendar.getInstance();
    calendar.setTimeInMillis(System.currentTimeMillis());
    calendar.set(Calendar.HOUR_OF_DAY, 9);
    calendar.set(Calendar.MINUTE, 00);

    //repeat alarm daily
    alarmMgr.setRepeating(AlarmManager.RTC, calendar.getTimeInMillis(),
            60000 * 60 * 24, alarmIntent);
}
```

After the notification is deployed, the *broadcast receiver* should be declared in the *manifest file*. By default, all *alarms* in the Android operating system are cancelled when a device shuts down. To prevent this, the `RECEIVE_BOOT_COMPLETED` permission should be set in the *manifest file*.

Figure 50 Receive boot permission in the manifest file

```
<uses-permission
android:name="android.permission.RECEIVE_BOOT_COMPLETED" />
```

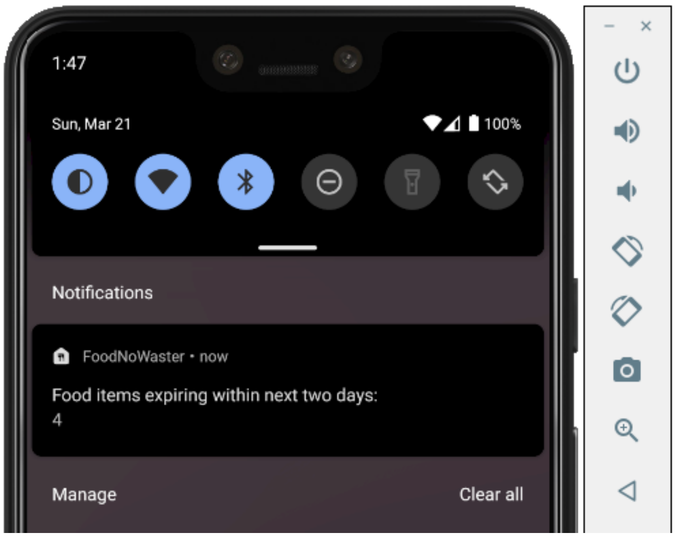
After permission is set, the *intent* filter that filters on the `ACTION_BOOT_COMPLETED` action should be added to the receiver.

Figure 51 Intent filter after reboot

```
<!-- registering the receiver-->
<receiver
    android:name=".MyAlarm"
    android:enabled="true"
    android:exported="true">
    <intent-filter>
        <action android:name="android.intent.action.BOOT_COMPLETED"></action>
        <action android:name="android.intent.action.MAIN"></action>
    </intent-filter>
</receiver>
```

When the implementation is ready, the user will daily receive a notification reminding how many food items have an expiration date in the next two days. The notification will be displayed only if the total number of the food items with the expiration date within the next two days are greater than zero. The notification will be also displayed after the device reboot. The notification will have a small, rounded Vector Asset Clip Art "food bank" icon. After the tap on the notification, the application will be launched. The notification can be removed by an away swipe.

Figure 52 FoodNoWaster notification about expiring food items

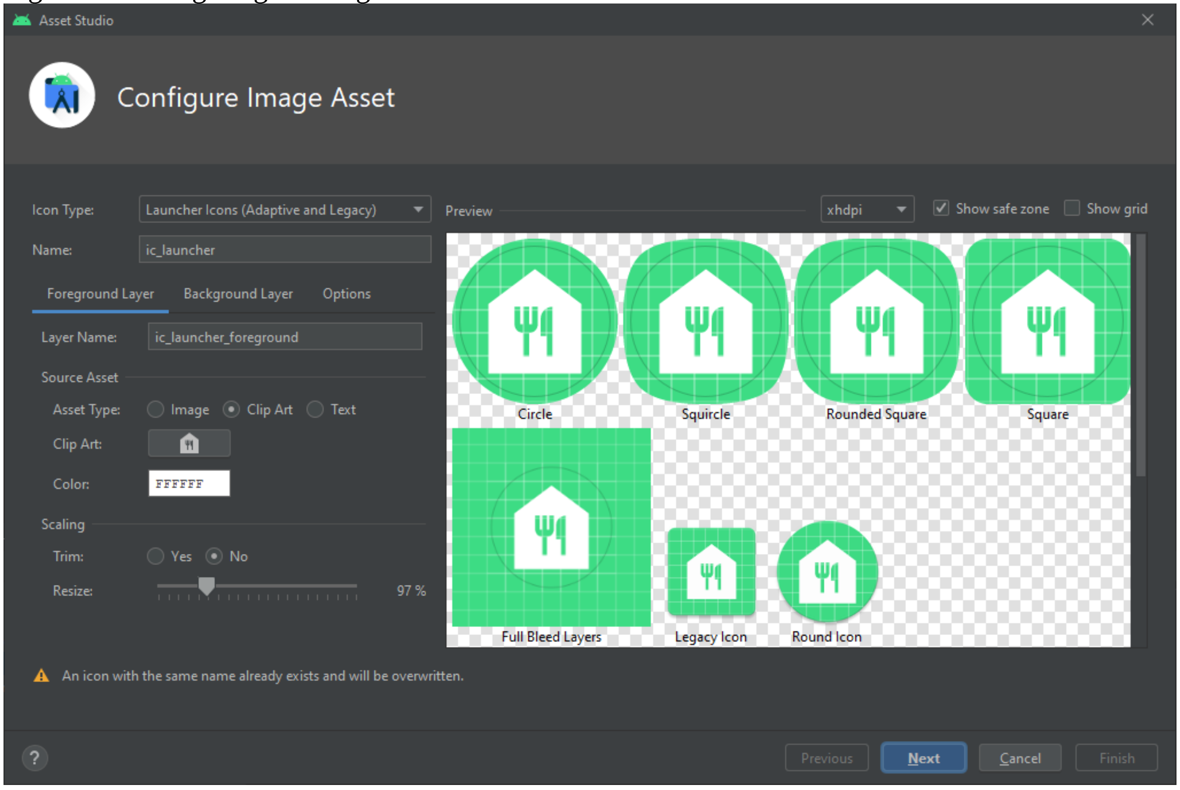


Food items expiring within next two days:
4

4.2.12 **Application icon**

When a new project is created in Android Studio, the default application icon in the form of an android is set automatically. To change the default icon, a new Image Asset should be created in the res subfolder of the project folder. A food bank Clip Art will be chosen as an icon for the FoodNoWaster application. A food bank launcher icon will be white coloured on the green background. The icon will be adaptive to different screen resolutions and system theme styles. It will consist of two layers: a foreground and a background layer. The green and white grid make the background layer, while the white mask will be applied on top of the foreground layer to produce a shaped app icon. The foreground layer will be stacked on top of the background layer.

Figure 53 Configuring an Image Asset in Android Stuudio


Asset Studio

Configure Image Asset

Icon Type: Launcher Icons (Adaptive and Legacy)

Name: ic_launcher

Layer Name: ic_launcher_foreground

Asset Type: Image [✓ Clip Art] Text

Color: FFFFFF

Trim: Yes [✓ No]

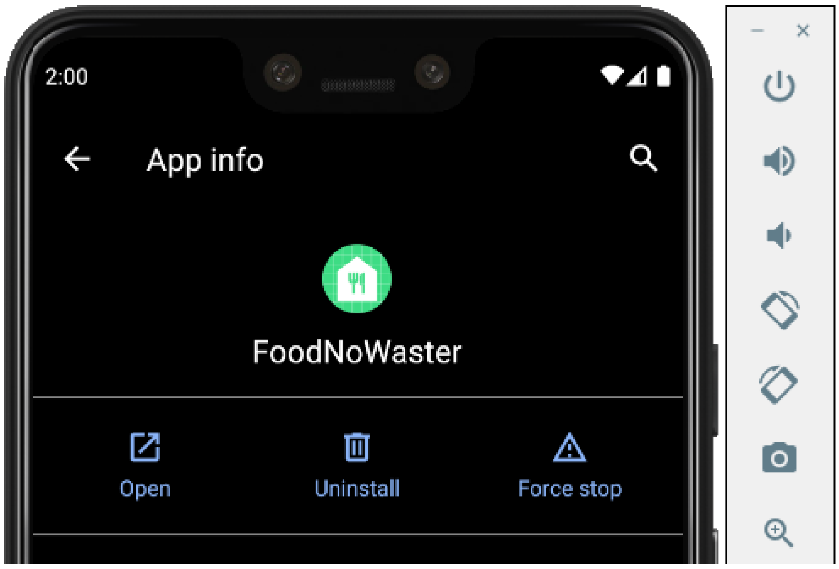
Resize: 97 %

xhdpi; [✓ Show safe zone]; Show grid

⚠ An icon with the same name already exists and will be overwritten.

When the launcher icon is ready, it will be displayed in the list of applications installed on the device, it will represent shortcuts into the application, will be used by launcher applications, and will help users to find an application on Google Play.

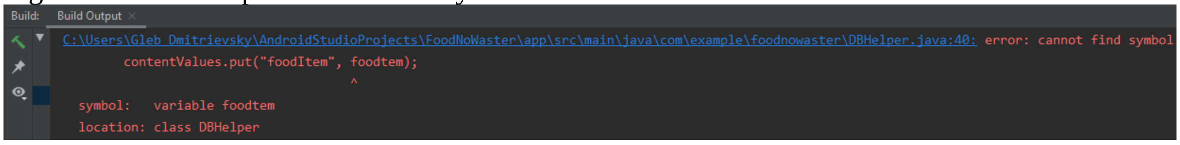
Figure 54 FoodNoWaster application icon



4.2.13 **Debugging the application**

The Android Studio provides a developer with a debug output. This output allows the developer to track possible mistakes in the code, misspelling and more. For example, the output can provide the information that the variable was not found. It will provide the information about the location of the error.

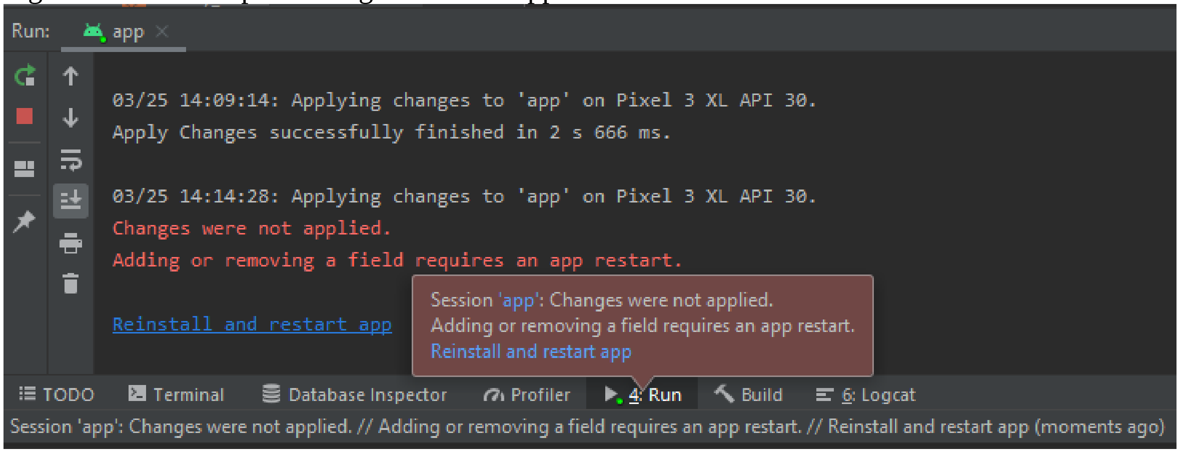
Figure 55 Build Output - cannot find symbol



```
C:\Users\Gleb Dmitrievsky\AndroidStudioProjects\FoodNoWaster\app\src\main\java\com\example\foodnowaster\DBHelper.java:40: error: cannot find symbol
        contentValues.put("foodItem", foodtem);
                                      ^
  symbol:   variable foodtem
  location: class DBHelper
```

It can also provide some hints on how to eliminate the error. For example, if the developer has provided some core changes to the code and try to apply them on-the-go in the Android Studio Emulator. In such a case, the developer will be notified that the changes were not applied and that the restart is needed. The direct link to reinstall and restart the application will be also provided.

Figure 56 Build Output - changes were not applied



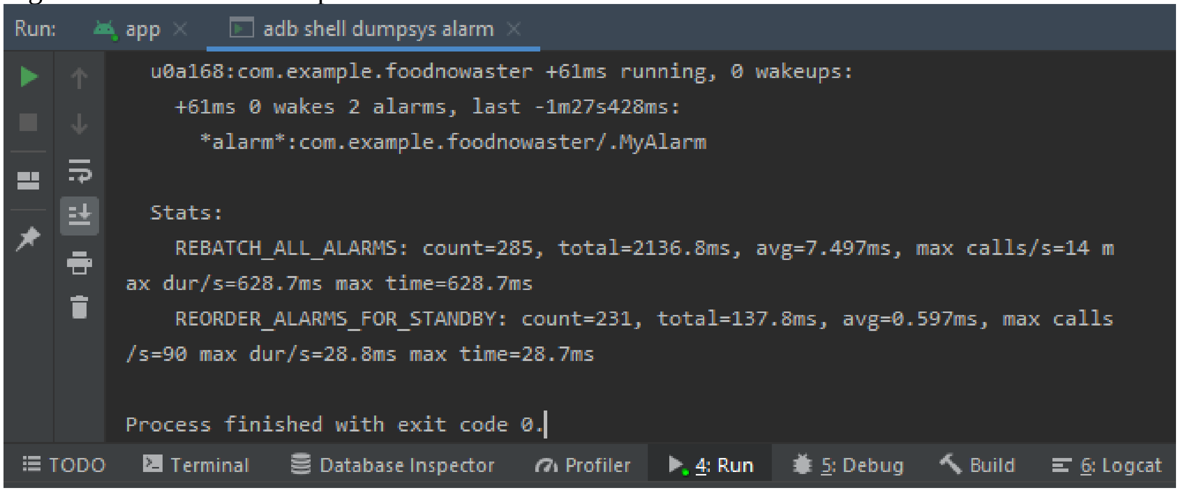
```
03/25 14:09:14: Applying changes to 'app' on Pixel 3 XL API 30.
Apply Changes successfully finished in 2 s 666 ms.

03/25 14:14:28: Applying changes to 'app' on Pixel 3 XL API 30.
Changes were not applied.
Adding or removing a field requires an app restart.

Reinstall and restart app
```

Session 'app': Changes were not applied.
Adding or removing a field requires an app restart.
Reinstall and restart app

Session 'app': Changes were not applied. // Adding or removing a field requires an app restart. // Reinstall and restart app (moments ago)

ADB is a very powerful tool that can also help the developer to debug the application. For example, the developer needs to debug *alarm* behaviour. To do that, the ADB command `adb shell dumpsys alarm` should be executed. As an output, the developer will be provided with information about running *alarms*.

Figure 57 ADB alarms output

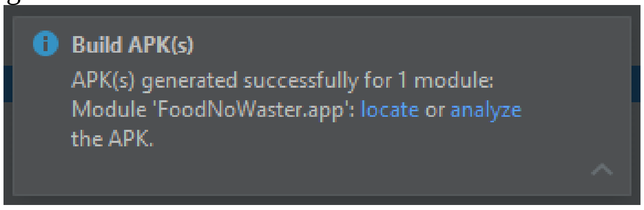


```
  u0a168:com.example.foodnowaster +61ms running, 0 wakeups:
    +61ms 0 wakes 2 alarms, last -1m27s428ms:
      *alarm*:com.example.foodnowaster/.MyAlarm

  Stats:
    REBATCH_ALL_ALARMS: count=285, total=2136.8ms, avg=7.497ms, max calls/s=14 m
ax dur/s=628.7ms max time=628.7ms
    REORDER_ALARMS_FOR_STANDBY: count=231, total=137.8ms, avg=0.597ms, max calls
/s=90 max dur/s=28.8ms max time=28.7ms

Process finished with exit code 0.
```

4.2.14 **Application publishing**

The FoodNoWaster application will be tested within the narrow range of beta testers. For this purpose, beta testers will be provided with the `.apk` application. The Build APK(s) option is available under the Build option in the Android Studios' toolbar. After the build is done, the Android Studio will provide the user with a `.apk` file ready to install on devices. The generated APK can only be used for testing. To install this file, the user should allow the installation of the applications from unknown sources in the system settings of the device.

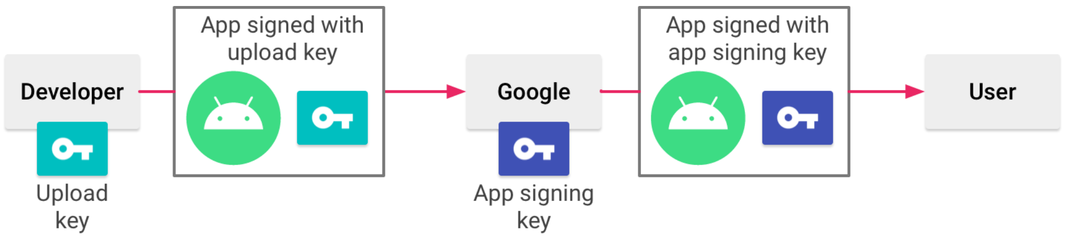
Figure 58 APK building in Android Studio


Build APK(s)

APK(s) generated successfully for 1 module:
Module 'FoodNoWaster.app': locate or analyze the APK.

After the FoodNoWaster application beta testing phase will be ended, the application can be published to Google Play. Before publishing the application to Google Play, upload key and keystore should be generated. After an upload key is generated, the application can be signed with that key. The next step is preparation and rolling out the release. After choosing a release track, the application signing should be configured. The next step is to upload the application to Google Play. The Google Play staff will inspect, test, and then publish the application to the marketplace.

Figure 59 Signing an application with Google Play signing



Source: [40]

5 Results and Discussion

5.1 FoodNoWaster application

The developed FoodNoWaster application is the result of this thesis. It is a working application where the user can put his food items to prevent food wasting. When the food items will come close to their expiration date, the user will be notified about that. During the pilot testing within 4 users, the application performed well, users did not report any issues and were classified the usage as satisfied.

The FoodNoWaster application has a simple user-friendly user interface that does not require any introduction or guides to work with. The application is also developed following Google's guides and using Google's Material Design. The FoodNoWaster application has not been released to a wide range of users yet. It is in a beta testing phase and is distributed within a narrow range of users through sharing `.apk` file.

The development phase of the FoodNoWaster application took about two months. The application consists of about 1200 lines of code. According to the basic COCOMO procedural software cost estimation model, the effort value of the FoodNoWaster application equals 2.4 (person/month), and the duration value is about 3.5 months.

5.2 Further enhancements

There are several possibilities on how to improve the FoodNoWaster application in future. One of them is to widen the range of beta users and get feedback on what functionalities could be better. The other one is the usage of an external database, like firebase. That will make it possible to create user accounts, so the users will be able to switch their phones or get a back-up of their data. The third possibility is to add the functionality of adding pictures of food items. There can also be implemented solutions like Google Lens to get the expiration date automatically read. One more future enhancement is to distribute the application within Android application marketplaces, for example, Google Play. It will make it easier for users to find, download, and install the application. Also, there will be an opportunity to reach a bigger audience with Google Plays' promotion activities.

6 Conclusion

At the beginning of writing this thesis, I had no experience with application development for the Android operating system. My goal was to examine how the applications are developed from the idea to the final product. During writing this thesis I was able to gain and practice knowledge from the theoretical background. I also had a chance to apply the knowledge from my study. I had an opportunity to develop an application and go through all stages of the application development process.

During the writing of this thesis, I examined the problem of food wasting and the theoretical base for Android application development. After that, I analyzed the user’s requirements, target audience, and determined the key functions of the application. Based on this theoretical background and analysis, I have developed a FoodNoWaster application.

The result of this thesis is ready for use application, which I, my friends and my family are using on a daily basis. The application allows users to input their food items into the database, see ordered lists, and receive notifications about food items that are approaching their expiration date. The main idea behind the technical part was to help people to prevent their food waste. Looking at my own observed statistics, people could prevent up to 90% of their probable food waste.

The application was developed in Android Studio using the Java programming language for core logic. The layouts of the application were built with XML layouts.

Even though the application is usable and stable, it is still in a beta testing phase. During that phase, I want to collect usage reports and ideas about the most needed enhancements. After collecting this data, fixing possible bugs, and implementing the best idea, the application can be released to the Google Play market.

7 References

1. SCHERHAUFER, Silvia, OBERSTEINER, Gudrun. FOOD WASTE REDUCTION AND ITS POTENTIAL TO MITIGATE GLOBAL WARMING. Stockholm, 2016. ISBN 978-91-88319-01-2.
2. No time to waste: assessing the performance of food waste prevention actions - ScienceDirect. ScienceDirect.com | Science, health and medical journals, full text articles and books [online]. 20 November 2020. https://www.sciencedirect.com/science/article/pii/S0921344920302640. 06 February 2021.
3. Mobile App Download and Usage Statistics (2021) - BuildFire. App Builder | The Best App Maker for High End Mobile Apps. https://buildfire.com/app-statistics. Accessed. 23 March 2021.
4. There are now 2.5 billion active Android devices - The Verge. The Verge [online]. https://www.theverge.com/2019/5/7/18528297/google-io-2019-android-devices-play-store-total-number-statistic-keynote. Accessed 25 November 2020.
5. The history of Android: The evolution of the biggest mobile OS in the world. Android Authority: Tech Reviews, News, Buyer's Guides, Deals, How-To [online]. https://www.androidauthority.com/history-android-os-name-789433. Accessed 25 November 2020.
6. How Google Play works | Android. Android | The platform pushing what’s possible [online]. https://www.android.com/play/how-google-play-works/.Accessed 25 November 2020.
7. Swift - Apple Developer. Swift is a powerful and intuitive programming language for iOS, macOS, tvOS, and watchOS [online]. https://developer.apple.com/swift/. Accessed 26 November 2020.
8. App Store - Developing for the App Store - Apple. Apple [online]. https://www.apple.com/app-store/developing-for-the-app-store/. Accessed 26 November 2020.
9. Set up for Android Development | Android Open Source Project. Android Open Source Project [online]. 01 September 2020. https://www.source.android.com/setup. Accessed 26 November 2020.
10. Platform Architecture | Android Developers. Android Developers [online]. https://developer.android.com/guide/platform. Accessed 22 November 2020.
11. File permissions and attributes - ArchWiki. ArchWiki [online]. https://wiki.archlinux.org/index.php/File_permissions_and_attributes/. Accessed 01 December 2020.
12. SCHILDT, Herbert. Java: The Complete Reference. 11th Edition. New York: McGraw-Hill, 2018. ISBN 9781260440249.
13. GRIFFITHS, Dawn and David GRIFFITHS. Head First Android Development. 2nd Edition. Newton: O'Reilly Media, 2017. ISBN 9781491974056.
14. Application Fundamentals | Android Developers. Android Developers [online]. https://developer.android.com/guide/components/fundamentals. Accessed 05 January 2021.

15. Understand the Activity Lifecycle | Android Developers. Android Developers [online]. https://developer.android.com/guide/components/activities/activity-lifecycle. Accessed 9 January 2021.
16. Services overview | Android Developers. Android Developers [online]. https://developer.android.com/guide/components/services. Accessed 9 January 2021.
17. Broadcasts overview | Android Developers. Android Developers [online]. https://developer.android.com/guide/components/broadcasts. Accessed 10 January 2021.
18. Content providers | Android Developers. Android Developers [online]. https://developer.android.com/guide/topics/providers/content-providers. Accessed 10 January 2021.
19. Content provider basics | Android Developers. Android Developers [online]. https://developer.android.com/guide/topics/providers/content-provider-basics. Accessed 10 January 2021.
20. Intents and Intent Filters | Android Developers. Android Developers [online]. https://developer.android.com/guide/components/intents-filters. Accessed 6 February 2021.
21. Notifications Overview | Android Developers. Android Developers [online]. https://developer.android.com/guide/topics/ui/notifiers/notifications. Accessed 18 January 2021.
22. Schedule repeating alarms | Android Developers. Android Developers [online]. https://developer.android.com/training/scheduling/alarms. Accessed 21 February 2021.
23. Toasts overview | Android Developers. Android Developers [online]. https://developer.android.com/guide/topics/ui/notifiers/toasts. Accessed 23 February 2021.
24. Debug your app | Android Developers. Android Developers [online]. https://developer.android.com/studio/debug. Accessed 23 February 2021.
25. Android Debug Bridge (adb) | Android Developers. Developers [online]. https://developer.android.com/studio/command-line/adb. Accessed 23 February 2021.
26. Publish your app | Android Developers. Developers [online]. https://developer.android.com/studio/publish. Accessed. 24 February 2021.
27. WEISFELD, Matt. The object-oriented thought process. 5th Edition. Boston: Addison-Wesley, 2019. ISBN 9780135182130.
28. FOWLER, Martin. UML distilled: a brief guide to the standard object modeling language. Addison-Wesley Professional, 2004.
29. SETH, Christophe. The SWOT Analysis: A key tool for developing your business strategy. 50Minutes.com, 2015.
30. MAYLOR, Harvey. Beyond the Gantt chart: Project management moving on. European management journal 19, no. 1 (2001): 92-100.
031. Download Android Studio and SDK tools | Android Developers. Android Developers [online]. https://developer.android.com/studio. Accessed 22 January 2021.
32. Eclipse IDE 2020-12 | The Eclipse Foundation. Enabling Open Innovation & Collaboration | The Eclipse Foundation [online]. https://www.eclipse.org/eclipseide/. Accessed 22 January 2021.
33. Visual Studio IDE, Code Editor, Azure DevOps, & App Center - Visual Studio. Visual Studio IDE, Code Editor, Azure DevOps, & App Center - Visual Studio [online]. https://visualstudio.microsoft.com/. Accessed 22 January 2021.

34. Features - IntelliJ IDEA. JetBrains: Essential tools for software developers and teams [online]. https://www.jetbrains.com/idea/features. Accessed 22 January 2021.
35. Developer workflow basics | Android Developers. Android Developers [online]. https://developer.android.com/studio/workflow. Accessed 22 January 2021.
36. Karma App - Save food from being wasted [online]. https://karma.life/about. Accessed 14 February 2020.
37. The problem of food waste - OLIO. OLIO – The #1 Free Sharing App [online]. https://olioex.com/food-waste/the-problem-of-food-waste/. Accessed 14 February 2021.
38. FoodKeeper App | FoodSafety.gov. FoodSafety.gov [online]. https://www.foodsafety.gov/keep-food-safe/foodkeeper-app. Accessed 14 February 2021.
39. Create an Android project | Android Developers. Android Developers [online]. https://developer.android.com/training/basics/firstapp/creating-project. Accessed 18 January 2021.
40. Sign your app | Android Developers. Developers [online]. https://developer.android.com/studio/publish/app-signing. Accessed. 23 March 2021.