CHAPTER 1

INTRODUCTION

1.1 Introduction to Kotlin

Kotlin is a statically-typed, modern programming language that runs on a Java Virtual Machine (JVM) by compiling Kotlin code into Java byte-code. It can also be compiled to JavaScript source code and to native executables. Kotlin is flexible. Kotlin is object-oriented language, and a "better language" than Java, but still be fully interoperable with Java code. Kotlin is officially supported by Google for Android development, meaning that Android documentation and tooling is designed with Kotlin in mind.

Kotlin is sponsored by Google, announced as one of the official languages for Android Development in 2017. Kotlin is multi-platform language, i.e. easily executable on a Java Virtual Machine.

Kotlin is Used for:

- Mobile applications (specially Android apps)
- Web development
- Server side applications
- Data science

Why Use Kotlin?

- Kotlin is fully compatible with Java.
- Kotlin works on different platforms (Windows, Mac, Linux, Raspberry Pi, etc.
- Kotlin is concise and safe.
- Kotlin is easy to learn, especially if you already know Java.
- Kotlin is free to us.
- Less Compilation time.
- User-Friendly.

1.2 Project intro and abstract

The NASA Gallery Android App is a feature-rich mobile application built using Kotlin and the Android platform. It allows users to explore the mysteries of the universe by providing access

to NASA's Astronomical Picture of the Day (APOD) and a vast library of captivating astronomical images. The app follows the Model-View-ViewModel (MVVM) architectural pattern and leverages NASA's API to seamlessly retrieve and display the latest astronomical images and associated information.

With its user-friendly interface, intuitive navigation, and educational resources, the app aims to inspire users of all ages to develop a deeper understanding and appreciation for astronomy. By delivering daily celestial wonders and enabling exploration of the image library, the NASA Gallery Android App offers an immersive and educational experience for space enthusiasts.

1.3 How the idea got into existence

The idea of a NASA Gallery project, like many other scientific endeavors, typically evolves through a combination of scientific goals, technological advancements, and funding opportunities. While I don't have access to specific details about the latest Gallery project by NASA due to my knowledge cutoff in September 2021, I can provide you with a general overview of how such projects come into existence based on historical examples.

1. Scientific Goals and Community Input: Scientists and researchers identify key questions and areas of study in astronomy or astrophysics that require advanced observational capabilities. These goals are often discussed and refined through collaborations, conferences, and scientific community input.

2. Technological Feasibility: Advances in technology, both in terms of instrumentation and spacecraft capabilities, play a significant role in enabling new Gallery projects. Scientists and engineers assess the feasibility of developing instruments or spacecraft that can achieve the desired scientific goals within the constraints of available technology.

3. Concept Development: NASA, in collaboration with scientists and engineers, initiates a concept development phase for the gallery project. This involves studying the feasibility, costs, and technical requirements of the proposed mission. Multiple design iterations and scientific assessments are performed to refine the mission concept.

CHAPTER 2

REQUIREMENT SPECIFICATIONS

Android Studio is the official integrated development environment (IDE) for Google's Android operating system, built on JetBrains' IntelliJ IDEA software and designed specifically for Android development. It is available for download on Windows, macOS and Linux based operating systems The first stable build was released in December 2014. Android Studio provides many excellent features that enhance productivity when building Android apps, such as a blended environment where one can develop for all Android devices, apply Changes to push code and resource changes to the running app without restarting the app, a flexible Gradle-based build system, a fast and feature-rich emulator, GitHub and Code template integration to assist you to develop common app features and import sample code, extensive testing tools and frameworks, C++ and NDK support, and many more.

2.1 Hardware Requirements

- Processor - Intel® Core i5 CPU M 370
- Processor Speed - 500 MHz or above
- RAM – Minimum of 8GB.
- Monitor resolution - A color monitor with a minimum resolution of 1000*700

2.2 Software Requirements

- Java Development Kit (JDK)
- Android Studio
- Android Emulator or a Physical device
- Kotlin Access.

2.3 Functional Modules

As a user navigates through, out of, and back to our app, the Activity instances in our app transition through different states in their lifecycle. The Activity class provides a number of callbacks that allow the activity to know that a state has changed, that the system is creating, stopping, or resuming an activity, or destroying the process in which the activity resides as shown in figure 1.1. To navigate transitions between stages of the activity lifecycle, the Activity class provides a core set of six callbacks: onCreate(),onStart(), onResume(), onPause(),

onStop(), and onDestroy(). The system invokes each of these callbacks as an activity enters a new state. In the onCreate() method, we perform basic application startup logic that should happen only once for the entire life of the activity. For example, implementation of onCreate() might bind data to lists, associate the activity with a ViewModel, and instantiate some class-scope variables. This method receives the parameter savedInstanceState, which is a Bundle object containing the activity's previously saved state. If the activity has never existed before, the value of the Bundle object is null. The onStart() call makes the activity visible to the user, as the app prepares for the activity to enter the foreground and become interactive. For example, this method is where the app initializes the code that maintains the UI. When the activity moves to the started state, any lifecycle-aware component tied to the activity's lifecycle will receive the ON_START event. When an interruptive event occurs, the activity enters the Paused state, and the system invokes the onPause() callback.

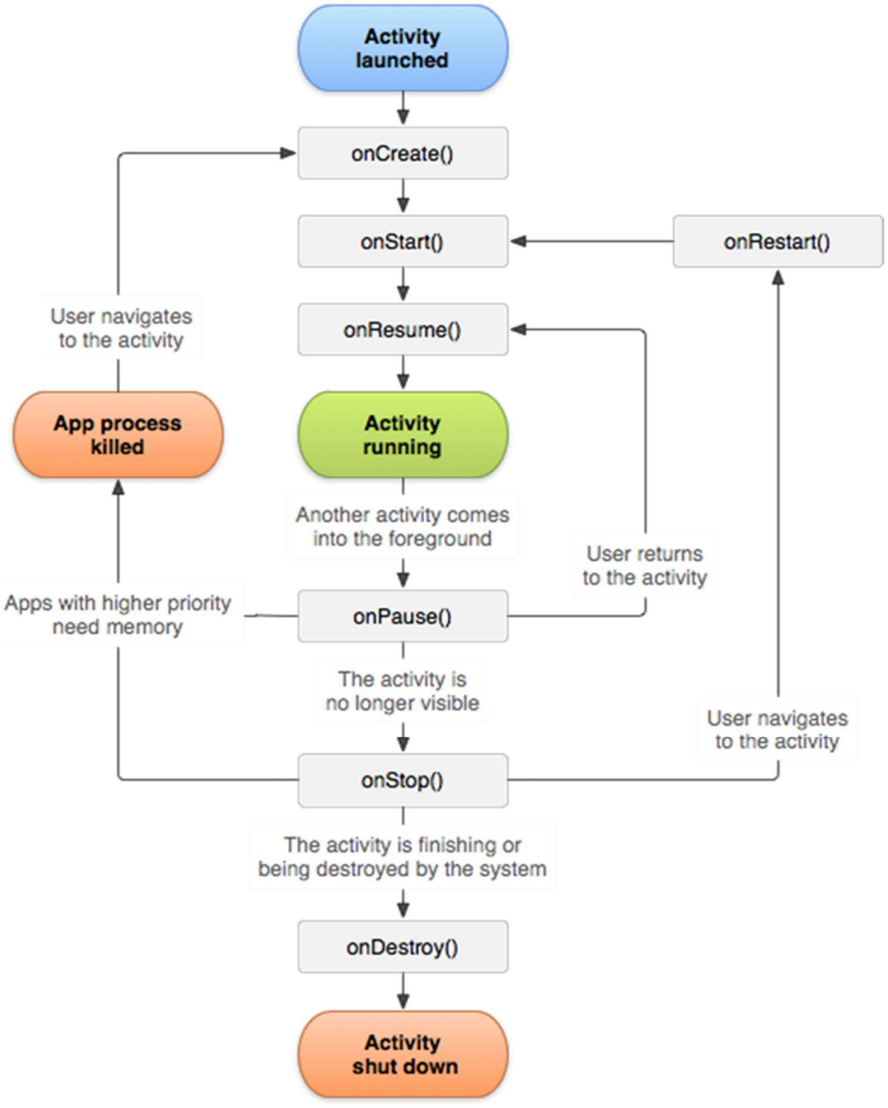

Figure 1.1: Android activity Life Cycle

When the activity enters the Resumed state, it comes to the foreground, and then the system invokes the **onResume()** callback. This is the state in which the app interacts with the user. The app stays in this state until something happens to take focus away from the app. Such an event might be, for instance, receiving a phone call, the user's navigating to another activity, or the device screen's turning off. When an interruptive event occurs, the activity enters the *Paused* state, and the system invokes the onPause() callback. When the activity is no longer visible to the user, it has entered the *Stopped* state, and the system invokes the **onStop()** callback. This may occur, for example, when a newly launched activity covers the entire screen. The system may also call onStop() when the activity has finished running, and is about to be terminated. **onDestroy()** is called before the activity is destroyed. The system invokes this callback either because:

- the activity is finishing (due to the user completely dismissing the activity or due to finish() being called on the activity), or
- the system is temporarily destroying the activity due to a configuration change (such as device rotation or multi-window mode)

The onDestroy() callback should release all resources that have not yet been released by earlier callbacks such as onStop().

CHAPTER 3

ABOUT THE PROJECT

3.1 Overview

The NASA Gallery Android App is a feature-rich mobile application built using Kotlin and the Android platform. It allows users to explore the mysteries of the universe by providing access to NASA's Astronomical Picture of the Day (APOD) and a vast library of captivating astronomical images. The app follows the Model-View-ViewModel (MVVM) architectural pattern and leverages NASA's API to seamlessly retrieve and display the latest astronomical images and associated information. With its user-friendly interface, intuitive navigation, and educational resources, the app aims to inspire users of all ages to develop a deeper understanding and appreciation for astronomy. By delivering daily celestial wonders and enabling exploration of the image library, the NASA Gallery Android App offers an immersive and educational experience for space enthusiasts.

3.2 Functionalities:

- Space Exploration.
- Technology Development.
- Satellite Communications.
- Data accessibility.
- Access the satellite picture2s of every day till today.
- Sky view of different planets.

3.3 Key features:

- Space-based Platforms:
- Education and public outreach.
- Multi-wavelength capabilities.
- Easy access.

3.4 Tech Stack

1. **Android Development:**

- Programming Language: Kotlin
- Integrated Development Environment (IDE): Android Studio

- Jetpack: Utilize various Jetpack components such as ViewModel, LiveData, and Navigation for efficient development.

2. **UI/UX:**
   - XML: Design user interfaces using XML layouts in Android Studio.
   - Material Design: Implement Material Design guidelines to ensure a consistent and visually appealing UI.
3. **Networking and API Integration:**
   - Retrofit: Communicate with backend APIs, if required, using Retrofit library for network requests.
   - Firebase Cloud Functions: Connect to custom Firebase Cloud Functions for additional backend functionality.

CHAPTER 4

SYSTEM DESIGN

4.1 Benefits of the application

- **Educational Content:** NASA apps often offer a wealth of educational content suitable for all ages. Users can access articles, videos, and interactive features that provide in-depth information about space science, astronomy, astrophysics, and other related topics. Such content can inspire curiosity and foster a love for science among users.
- **Astronomical Events:** The app can alert users about significant astronomical events, such as meteor showers, eclipses, and planetary alignments. Users can receive notifications about upcoming events, learn about their significance, and even receive tips on how to observe them effectively.
- **Space Image Galleries:** NASA is renowned for capturing breathtaking images of space and celestial objects. A NASA app can provide access to a vast collection of high-resolution images, including galaxies, nebulae, planets, and more. Users can explore these images, learn about the science behind them, and even set them as wallpapers.
- **Live Earth Observations:** Some NASA apps provide real-time access to satellite imagery and data about Earth's atmosphere, weather patterns, and climate. Users can monitor weather conditions, track storms, observe changes in sea ice, and gain insights into various Earth science phenomena.

4.2 List of Kotlin components used for application

An Android App is made of loosely coupled components that are separately and independently invoked but interoperate within the app ecosystem. App's AndroidManifest.xml file contains essential information about each component, how they interact, and their hardware configurations.

There are four main components:

- **Activity**: This is the presentation layer of an android application. They dictate the GUI and user interaction with the GUI. Illustratively, an email application can have one activity for sign-in/signup, another one for reading emails, and another for composing a new email. Additionally, activity keeps track of what 's currently on display and previously visited activities.

- **Services:** This is a backend operating component that manages operations long-running in the background. For example, service may be playing music in the background while the user is engaging a different application on the foreground or retrieving/sending information over the internet while the user is interacting with a separate app.

- **Broadcast Receivers:** This is an event-driven component, also known as Intent listeners. They listen and receive broadcasts, either from the system or other applications; they then act according to the triggers set in the broadcast. Examples are alarm sets and reminders.

- **Content Providers**: As the name suggests, they avail content stored in the file system, databases, or other applications. They act as a bridge between data and an app. For example, a media player application accesses music stored in phone storage through its content provider.

CHAPTER 5

IMPLEMENTATION AND TESTING

The app development process includes creating installable applications for the mobile devices and implementing backend services, for instance data access through an API. Testing the application on target devices is also part of the process.

1. NASAAPOD

```
class NASAAPOD : YouTubeBaseActivity(), YouTubePlayer.OnInitializedListener {
    private var mDay = 0
    private var mMonth = 0
    private var mYear = 0
    private var mNASAPhoto: ImageView? = null
    private var mNASATitle: TextView? = null
    private var mNASAExplanation: TextView? = null
    private var mVideoUrl: String? = null
    private var mDateSetListener: OnDateSetListener? = null
    private var mNASAVideo: YouTubePlayerView? = null
    private var myYouTubePlayer: YouTubePlayer? = null
    private var loadingAlert: LoadingAlert? = null
    private var player: MediaPlayer? = null
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_nasa_apod)
        loadingAlert = LoadingAlert( activity: this)
        val c = Calendar.getInstance()
        mDay = c[Calendar.DAY_OF_MONTH]
        mMonth = c[Calendar.MONTH]
        mYear = c[Calendar.YEAR]
        mNASAPhoto = findViewById(R.id.NASA_photo)
        mNASAVideo = findViewById(R.id.NASA_video)
        mNASAExplanation = findViewById(R.id.explanation)
        mNASATitle = findViewById(R.id.title_of_view)
        mVideoUrl = ""
        player = MediaPlayer.create( context: this, R.raw.sound2)
```

```
        mDateSetListener =
            OnDateSetListener { view: DatePicker?, year: Int, month: Int, dayOfMonth: Int ->
                loadingAlert!!.startLoading()
                mYear = year
                mMonth = month + 1
                mDay = dayOfMonth
                val calendar = Calendar.getInstance()
                calendar[Calendar.YEAR] = year
                calendar[Calendar.MONTH] = month
                calendar[Calendar.DAY_OF_MONTH] = dayOfMonth - 1
                if (calendar.before(Calendar.getInstance())) {
                    val date = date
                    initRetrofit(date)

                    if (myYouTubePlayer != null) myYouTubePlayer!!.release()
                } else {
                    loadingAlert!!.dismissDialog()
                    Toast.makeText( context: this, text: "Enter a day before or on today", Toast.LENGTH_SHORT)
                        .show()
                }
            }
        initNew()
    }
```

Figure 5.1: NASAAPOD

2. SearchResultImage

```
class SearchResultImage : AppCompatActivity() {
    private var mNASA_ID: String? = null
    private var mNASADesc: String? = null
    private var mNASAImage: ImageView? = null
    private var mNASATitle: TextView? = null
    private var mNASAName: String? = null
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_search_result_image)
        mNASA_ID = ""
        mNASADesc = ""
        mNASAName = ""
        if (intent.extras != null) {
            mNASA_ID = intent.extras!!.getString( key: "id")
            mNASADesc = intent.extras!!.getString( key: "desc")
            mNASAName = intent.extras!!.getString( key: "name")
        }
        mNASAImage = findViewById(R.id.nasa_image)
        mNASATitle = findViewById(R.id.nasa_title)
        initRetrofit()
    }
```

Figure 5.2: SearchResultImage

3. SplashScreen

```
    fun apod(view: View?) {
        MediaPlayer.create( context: this@SplashScreen, R.raw.sound2).start()
        println(animationView!!.duration)
        animationView!!.playAnimation()
        val time = animationView!!.duration
        apod!!.visibility = View.GONE
        apod!!.isClickable = false
        lib!!.visibility = View.GONE
        lib!!.isClickable = false
        apod!!.startAnimation(fade)
        lib!!.startAnimation(fade)
        cred!!.visibility = View.VISIBLE
        cred!!.startAnimation(fade1)
        welcome!!.visibility = View.INVISIBLE
        welcome!!.startAnimation(fade)
        object : CountDownTimer( millisInFuture: time / 2,  countDownInterval: time / 2) {
            override fun onTick(millisUntilFinished: Long) {}
            override fun onFinish() {
                startActivity(Intent( packageContext: this@SplashScreen, NASAAPOD::class.java))
                MediaPlayer.create( context: this@SplashScreen, R.raw.start).start()
                overridePendingTransition(R.anim.fadein, R.anim.fadeout)
                finish()
            }
        }.start()
    }
```

```
    fun library(view: View?) {
        MediaPlayer.create( context: this@SplashScreen, R.raw.sound2).start()
        println(animationView!!.duration)
        animationView!!.playAnimation()
        val time = animationView!!.duration
        apod!!.visibility = View.GONE
        apod!!.isClickable = false
        lib!!.visibility = View.GONE
        lib!!.isClickable = false
        apod!!.startAnimation(fade)
        lib!!.startAnimation(fade)
        cred!!.visibility = View.VISIBLE
        cred!!.startAnimation(fade1)
        welcome!!.visibility = View.INVISIBLE
        welcome!!.startAnimation(fade)
        object : CountDownTimer( millisInFuture: time / 2,  countDownInterval: time / 2) {
            override fun onTick(millisUntilFinished: Long) {}
            override fun onFinish() {
                startActivity(Intent( packageContext: this@SplashScreen, NASAImageAndVideo::class.java))
                MediaPlayer.create( context: this@SplashScreen, R.raw.start).start()
                overridePendingTransition(R.anim.fadein, R.anim.fadeout)
                finish()
            }
        }.start()
    }
}
```

Figure 5.3: SplashScreen

TESTING:

Verification and validation are a generic name given to checking processors, which ensures that the software conforms to it's specifications and meets the demands of the users.

- **Validation:** Are we building the right product? Validation involves checking that the program has implanted meets the requirement of the users.
- **Verification:** Verification involves checking that the program confirms to its specification.

Test case ID	Steps to execute the test case	Expected result	Actual result	Pass/ Fail
1	Click the start APOD button	Option to choose the date	Option to choose the date.	Pass
2	Select the date you wish to see the APOD	Go to APOD page that you chose	Go to APOD page that you chose	Pass
3	Click the start NASA library button	Directs you to the search page	Directs you to the search page	Pass
4	Search the objects you want to explore in the Universe.	Multiple Info about search object.	Info about search object.	Pass
5	Select the info for search object.	Description or picture is available.	Description or picture is available.	Pass

Table 6.1 Testing for NASA Gallery

CHAPTER 6

RESULTS AND SNAPSHOTS

By using relevant images of news application, using News API we get to know the different features of this application. The result obtained in each step are demonstrated in the following snapshots.

Snapshot of Home Page:

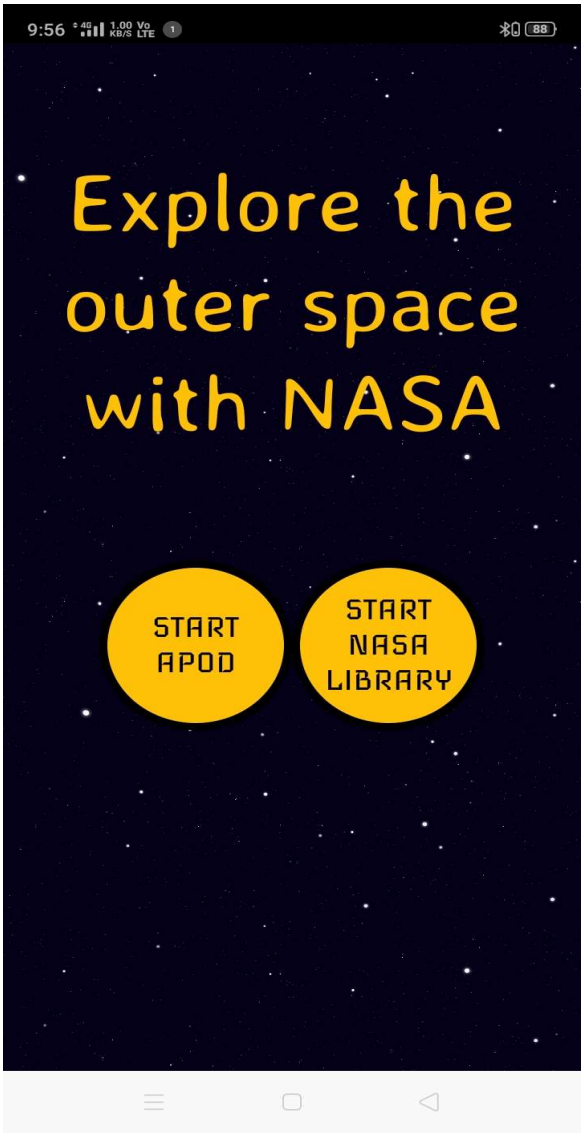

Figure 6.1: Home Page

This is the home page of NASA Gallery application. It has two options which are Start Astronomical Picture of the Day(APOD) and Start NASA Library.

Snapshot of APOD Page:

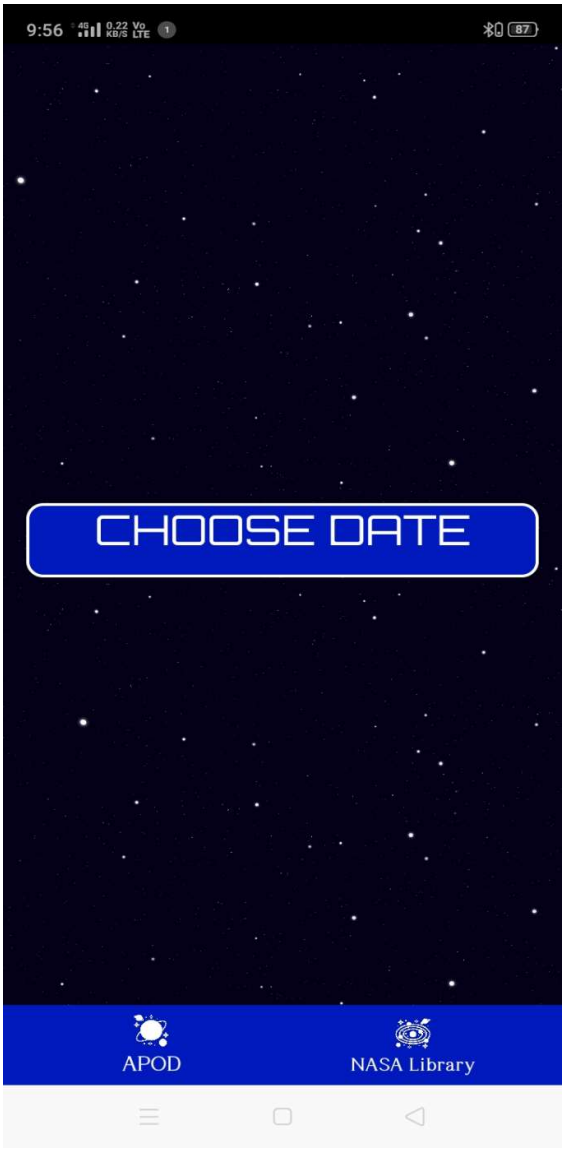

Figure 6.2: APOD page

In this page we can choose date to view the astronomical event of that day. Only present date and past date can be choosen because the events have been captured by NASA. It shows the event image and description of that event.

Snapshot of Calender Page:

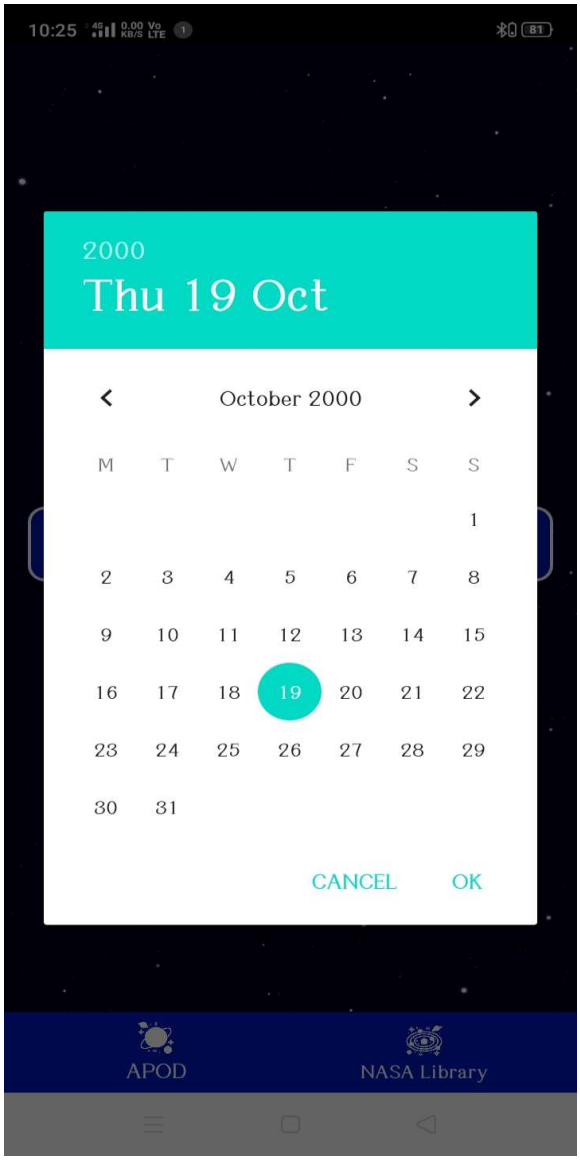

Figure 6.3: Calender page

In page we can choose the past date for which we wish to see the astronomical events.

Snapshot of Read APOD Page:

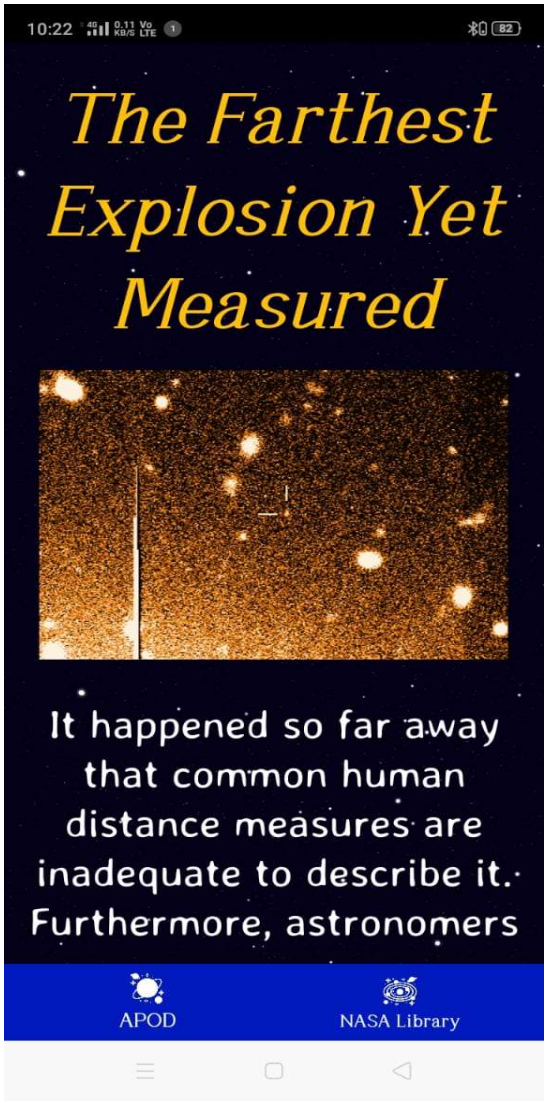

Figure 6.4: Read APOD page

In this page we can read the information related to the astronomical picture of the day to the given date. It shows the name of the event occurred on that particular date and provides small description about that event.

Snapshot of NASA library page:

Figure 6.5 NASA library page

In this page we can search the name of any Celestial Objects. It provides the topics related to searched objects and any topic can be selected. Selected topic provides an image related to the topic

Snapshot of Search Result Page:

Figure 6.6 Search Result Page

In this page we can see the top down list of topics related to the searched subject. All these topics are extracted through NASA API where it contents only pictures/graphs/diagrams of related topics.

Snapshot of Read NASA library Page:

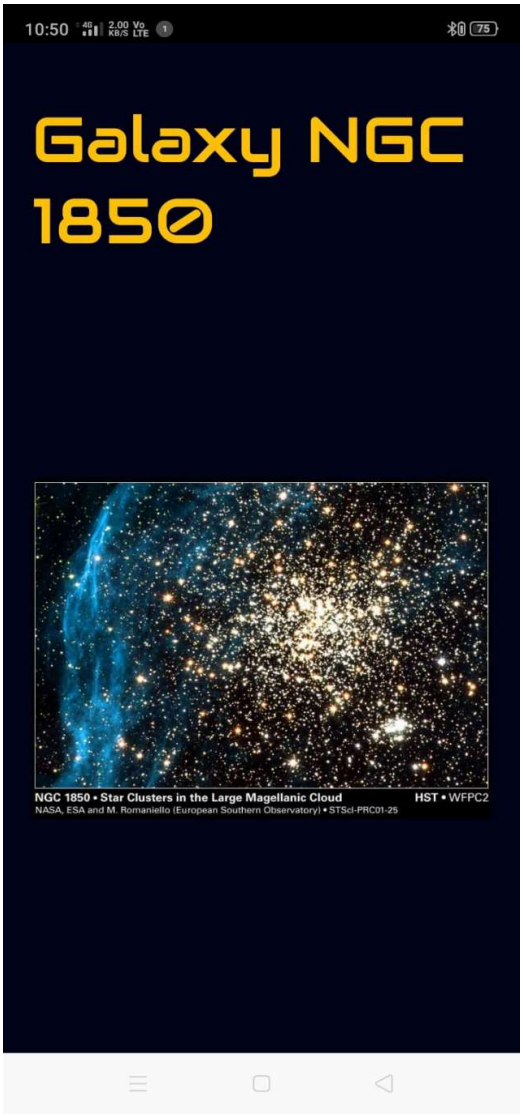

Figure 6.7 Read NASA library page

In this page we can view the image related to the searched subject. Only image/graph/diagram are shown in this page which are extracted through NASA API.

CHAPTER 7

CONCLUSION AND FUTURE ENHANCEMENTS

In conclusion, the NASA Gallery app is a remarkable platform that brings the wonders of space exploration and scientific discovery to the fingertips of users. Through a diverse collection of breathtaking images and interactive content, the app provides a captivating experience that engages and educates users about NASA's missions and discoveries. The app serves as a digital window into the vast universe, allowing users to explore stunning imagery captured by NASA's space telescopes, rovers, and satellites. From mesmerizing views of distant galaxies to close-up images of celestial bodies within our own solar system, the app showcases the beauty and grandeur of the cosmos.

One of the app's key strengths is its ability to provide context and information alongside the visual content. Users can access detailed descriptions, scientific explanations, and historical background about each image deepening their understanding of the scientific significance and the technology involved in capturing these remarkable visuals.

The NASA Gallery app is a captivating and educational platform that brings the wonders of the universe to the palm of our hands. It combines stunning visuals, informative content, and interactive features to provide a rich and immersive experience for users of all ages. Through this app, We inspire and ignite curiosity about space, encouraging a deeper appreciation for the scientific achievements and discoveries that have expanded our understanding of the cosmos.

FUTURE ENHANCEMENT:

In the future, there are several potential enhancements that could be implemented in the NASA Gallery Android App. One possibility is to incorporate additional features such as real-time satellite tracking, allowing users to visualize the positions of satellites in the sky. Another enhancement could involve integrating with social media platforms, enabling users to share their favourite astronomical images and discoveries with their friends and followers. Additionally, implementing a search functionality within the image library would provide users with the ability to easily find specific images or topics of interest. These enhancements would further enrich the user experience and expand the app's capabilities, making it an even more comprehensive and interactive tool for exploring the wonders of the universe.

1. Augmented Reality (AR) Integration: Implementing augmented reality technology would allow users to overlay virtual objects and spacecraft onto their real-world surroundings.

This feature could enable users to explore and interact with 3D models of rovers, satellites, and even planets, providing a more immersive and engaging experience.

2.Virtual Reality (VR) Experiences: Introducing virtual reality capabilities would take users on virtual tours of space missions, allowing them to experience the sensation of being inside a spacecraft or walking on the surface of another planet. VR technology could provide a sense of presence and transport users to distant celestial bodies, offering a truly immersive experience.

3.Live Streaming and Updates: Incorporating live streaming capabilities would enable users to witness significant NASA events in real-time, such as rocket launches, spacewalks, or scientific discoveries. Additionally, providing regular updates and news alerts within the app would ensure that users stay informed about the latest developments in space exploration.

4.Personalized Recommendations: Implementing a recommendation engine based on user preferences and interests would enhance the user experience by offering personalized content suggestions. By analyzing users' viewing history and interactions, the app could intelligently recommend related images, videos, and articles, fostering continued engagement and exploration.

5.Collaborative Features: Introducing collaborative features would allow users to engage with each other and share their experiences within the app. Users could create and join communities, participate in discussions, and share their own space-related content, fostering a sense of community and encouraging knowledge sharing.

6.Enhanced Education and Learning Tools: Expanding the educational aspect of the app could include interactive quizzes, educational games, and informative tutorials. These tools would enable users to deepen their understanding of space science and exploration while making the learning process interactive and fun.

7.Expanded Data Visualization: Integrating data visualization techniques within the app would allow users to explore scientific data collected by NASA missions in an intuitive and visually appealing way. Users could manipulate and interact with data visualizations, gaining insights into complex scientific concepts and discoveries.

8. Multi-Language Support: Adding support for multiple languages would broaden the app's accessibility and make it available to a wider global audience. By offering localized content and translations, NASA could engage with users from diverse backgrounds and foster a global interest in space exploration.

BIBLIOGRAPHY

- https://stackoverflow.com/
- https://www.geeksforgeeks.org/
- https://www.w3schools.com/java/
- https://www.javapoint.com/java-tutorial
- https://www.nasa.gov