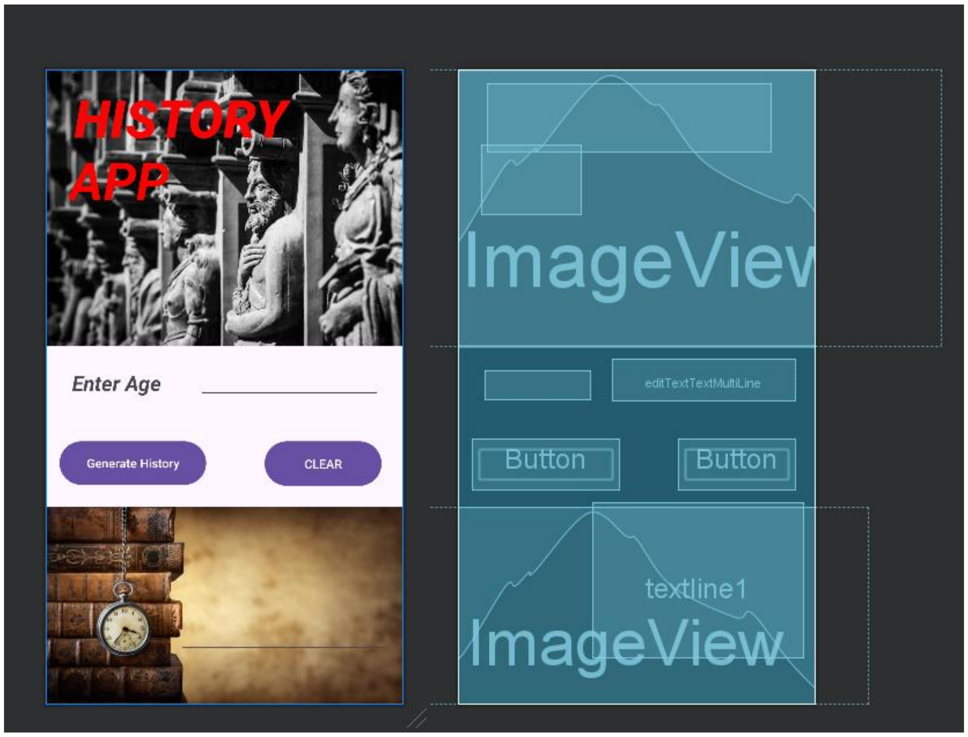
HISTORY APPLICATION ASSIGNMENT REPORT

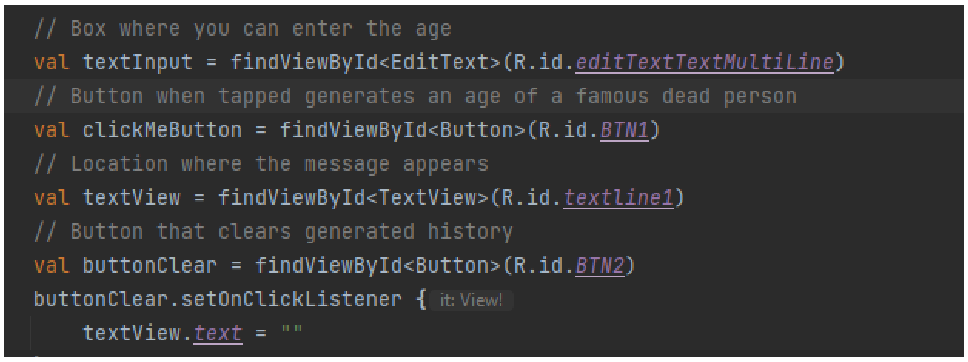
Mikaeel Mussett ST10437302

Tasked with creating a mobile application that would follow the aspect of taking a person's age and matching it with a famous person in history and when they passed away. The purpose of creating this app was to be an educational tool that students would be able to use but also have fun with. There were constraints that needed to be accounted for so the application to run successfully.

The first part consists of designing a UI that would captivate students and ensures that it is easy for them to interact with. As shown below the design of the UI, basic enough for students to interact with but also contains the necessity to be within the requirement guidelines. The UI contains two images representing history. Two buttons, one that generates the history when the age is entered on the text line. The second button is used to clear the information generated so that the information presented is reset and a new age can be entered so added information can be generated. A bottom text line is used and paired with the second image to show the information presented. Adding to the first part, importance of constraints needed to be utilized to ensure that the UI did not move around when the app is launched, and the content of the UI did not move around.

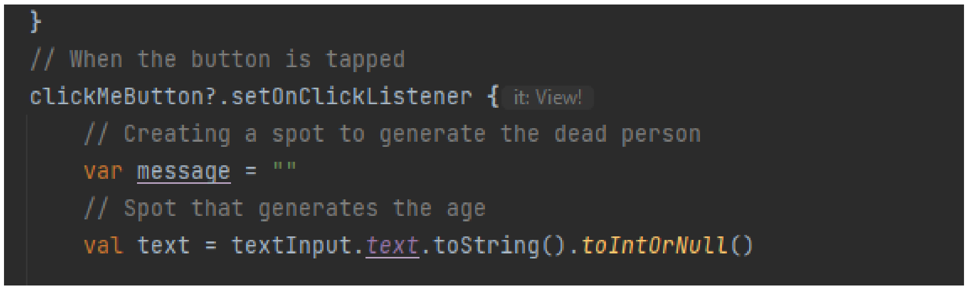


The second part consists of adding the code to ensure that the UI elements were now linked to a function. In this part of the code below we can see val is used multiple times. The use of val is to declare read only properties and assign it to a value once. Val is used below on both the buttons and the text lines found by the ID's that have been set during the process of designing the UI. Setting the use of the buttons below, so that when clicked it becomes interacted with.



```
// Box where you can enter the age
val textInput = findViewById<EditText>(R.id.editTextTextMultiLine)
// Button when tapped generates an age of a famous dead person
val clickMeButton = findViewById<Button>(R.id.BTN1)
// Location where the message appears
val textView = findViewById<TextView>(R.id.textline1)
// Button that clears generated history
val buttonClear = findViewById<Button>(R.id.BTN2)
buttonClear.setOnClickListener { it: View!
    textView.text = ""
```

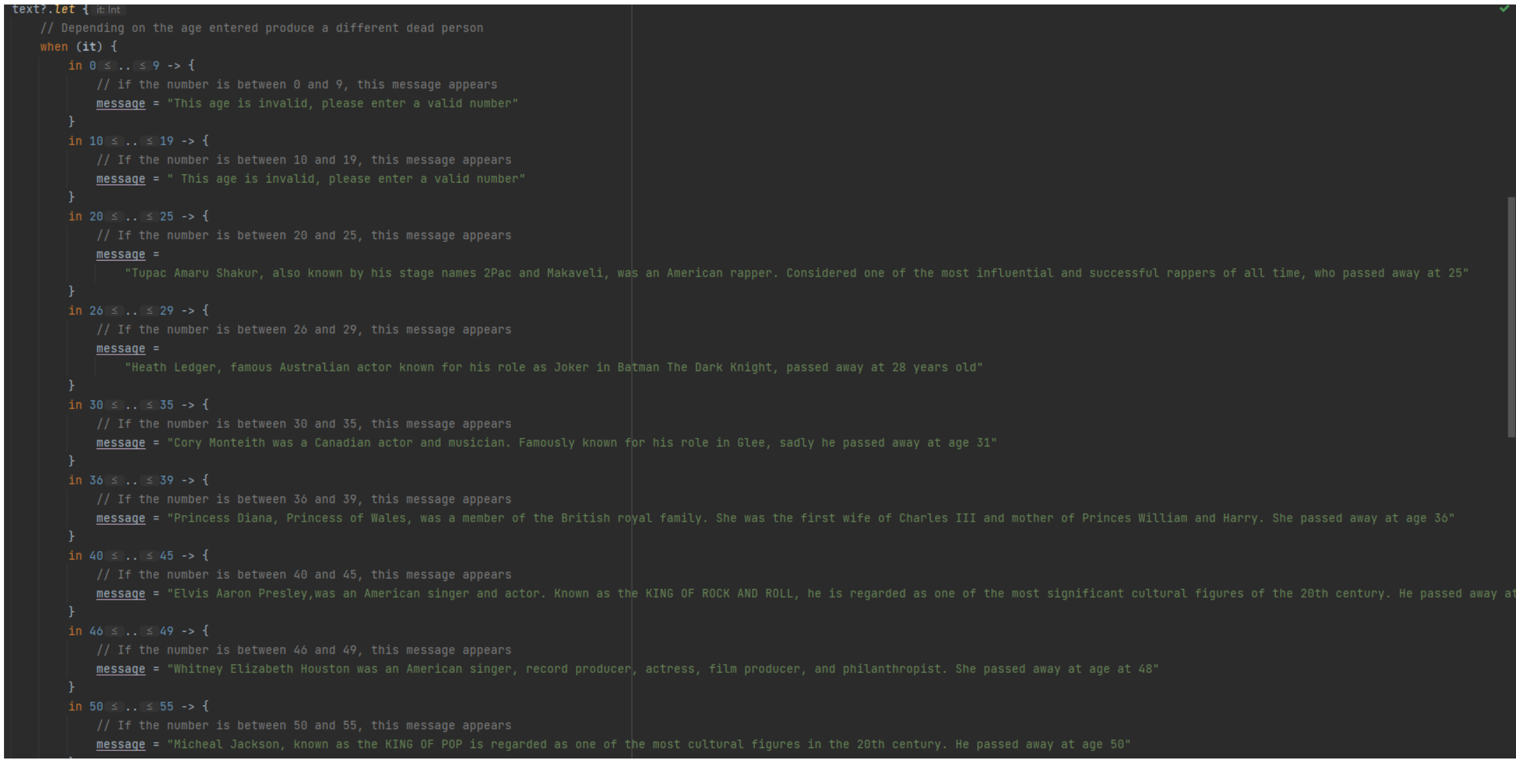
The next part entailed adding a function to when the button is clicked. The button named now has a click listener set up. The var messages declares a multiple variable that is named message, and it initializes it with an empty string. This also sets up an input field to follow for the next part of the code, which will attempt to convert it to an integer.



```
}
// When the button is tapped
clickMeButton?.setOnClickListener { it: View!
    // Creating a spot to generate the dead person
    var message = ""
    // Spot that generates the age
    val text = textInput.text.toString().toIntOrNull()
```

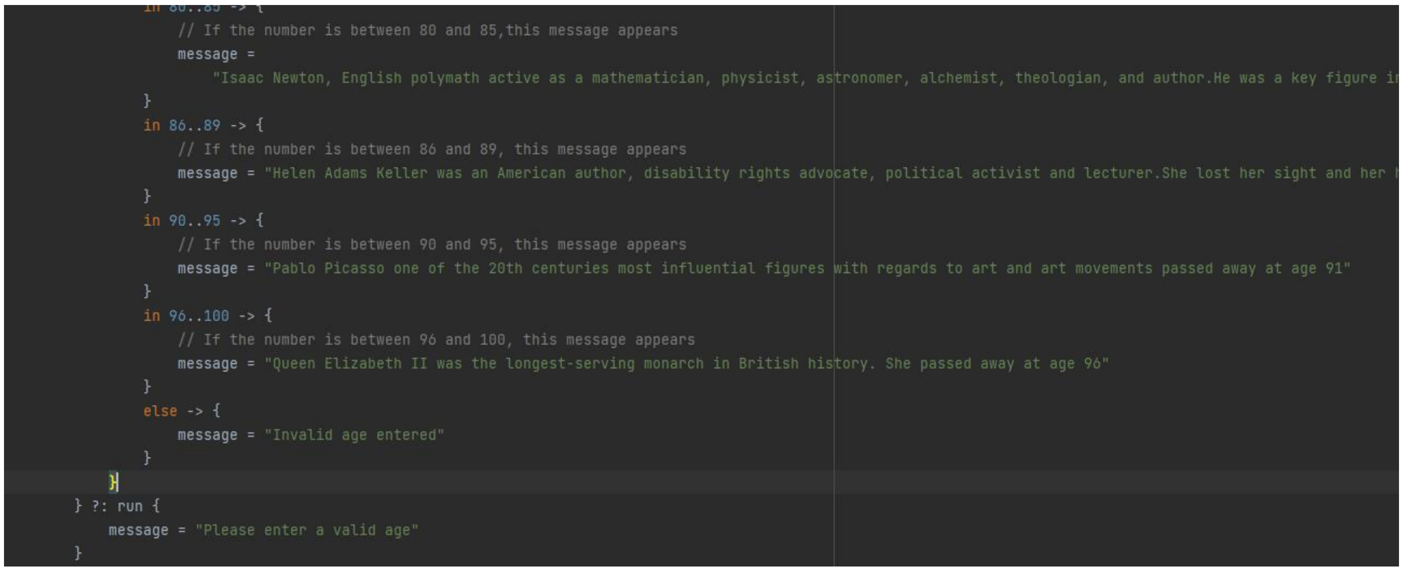
The code below begins to set up the information that would be displayed. This code also deals with one of the constraints we were given. The ages of the famous people that have passed away need to be between 20 – 100. This results in the constraint needing a message for those who input an age that is between 0 –19. The it statements below works like a switch statement in other languages and will represent the value of the integer input that is added to the code. The “in” that is a part of the code that is being used to specify a range of values which is going to be used for the age specification that is being used. The code below begins to put the age and information together so that when a user submits a number between the numbers set, a message will appear that matches the age or is proximity of the number entered a message will appear showing relevant information about a famous person and the age that they passed away. I defined a list of people through Chrome finding various famous people from sport stars, musicians, popular authors, movie stars and queens. Setting up the message input followed by the = then the

“” that will contain the message indicates for when the button is pushed the message input will be presented.



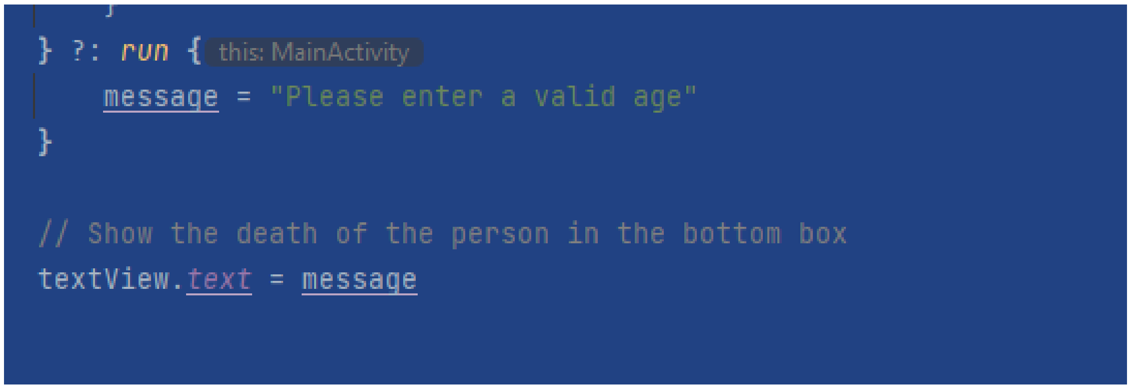
```
text?.let { it: Int
    // Depending on the age entered produce a different dead person
    when (it) {
        in 0 ≤ .. ≤ 9 -> {
            // if the number is between 0 and 9, this message appears
            message = "This age is invalid, please enter a valid number"
        }
        in 10 ≤ .. ≤ 19 -> {
            // If the number is between 10 and 19, this message appears
            message = " This age is invalid, please enter a valid number"
        }
        in 20 ≤ .. ≤ 25 -> {
            // If the number is between 20 and 25, this message appears
            message =
                "Tupac Amaru Shakur, also known by his stage names 2Pac and Makaveli, was an American rapper. Considered one of the most influential and successful rappers of all time, who passed away at 25"
        }
        in 26 ≤ .. ≤ 29 -> {
            // If the number is between 26 and 29, this message appears
            message =
                "Heath Ledger, famous Australian actor known for his role as Joker in Batman The Dark Knight, passed away at 28 years old"
        }
        in 30 ≤ .. ≤ 35 -> {
            // If the number is between 30 and 35, this message appears
            message = "Cory Monteith was a Canadian actor and musician. Famously known for his role in Glee, sadly he passed away at age 31"
        }
        in 36 ≤ .. ≤ 39 -> {
            // If the number is between 36 and 39, this message appears
            message = "Princess Diana, Princess of Wales, was a member of the British royal family. She was the first wife of Charles III and mother of Princes William and Harry. She passed away at age 36"
        }
        in 40 ≤ .. ≤ 45 -> {
            // If the number is between 40 and 45, this message appears
            message = "Elvis Aaron Presley,was an American singer and actor. Known as the KING OF ROCK AND ROLL, he is regarded as one of the most significant cultural figures of the 20th century. He passed away at
        }
        in 46 ≤ .. ≤ 49 -> {
            // If the number is between 46 and 49, this message appears
            message = "Whitney Elizabeth Houston was an American singer, record producer, actress, film producer, and philanthropist. She passed away at age at 48"
        }
        in 50 ≤ .. ≤ 55 -> {
            // If the number is between 50 and 55, this message appears
            message = "Micheal Jackson, known as the KING OF POP is regarded as one of the most cultural figures in the 20th century. He passed away at age 50"
```

The continuation of the code below shows the addition of statements until the age limit of 100. Statements added for the ages matching to the famous people in the integer zone permitted. In addition to the statements that have been added, the ending contains a line that deals with the constraint that if a user enters an age that isn't an integer or in the form of a word instead of a number then a message “Please enter a valid age” will be presented in order to ensure that the sure knows an integer will only be accepted. Due to the app being an educational app, the error provides a positive feedback mechanism to ensure the student is still able to learn from the app



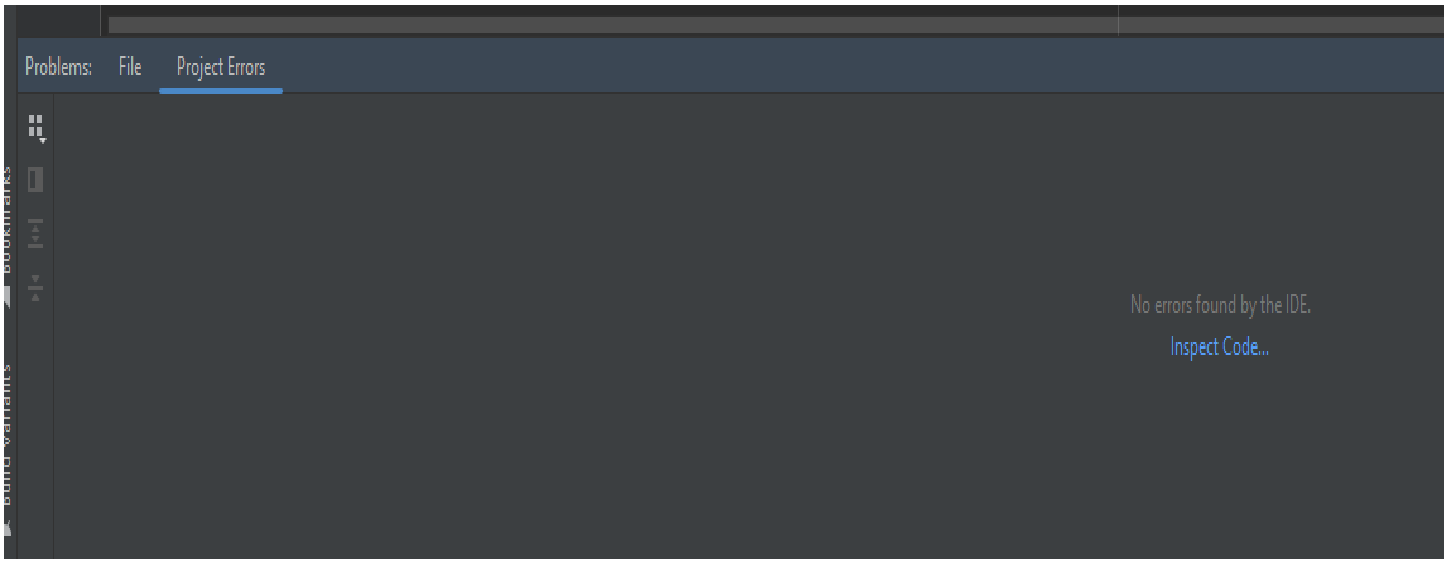
```
            in 80..85 -> {
                // If the number is between 80 and 85,this message appears
                message =
                    "Isaac Newton, English polymath active as a mathematician, physicist, astronomer, alchemist, theologian, and author.He was a key figure i
            }
            in 86..89 -> {
                // If the number is between 86 and 89, this message appears
                message = "Helen Adams Keller was an American author, disability rights advocate, political activist and lecturer.She lost her sight and her
            }
            in 90..95 -> {
                // If the number is between 90 and 95, this message appears
                message = "Pablo Picasso one of the 20th centuries most influential figures with regards to art and art movements passed away at age 91"
            }
            in 96..100 -> {
                // If the number is between 96 and 100, this message appears
                message = "Queen Elizabeth II was the longest-serving monarch in British history. She passed away at age 96"
            }
            else -> {
                message = "Invalid age entered"
            }
        }
    } ?: run {
        message = "Please enter a valid age"
    }
```

The last few lines of code are the most important as it deals with the users' interface and its interaction design. The update of the text view that is based on the user interaction as it provides communication with the user providing the information set at the input of the age entered, a response is generated. This enhances the users experience contributing to the success of the application.



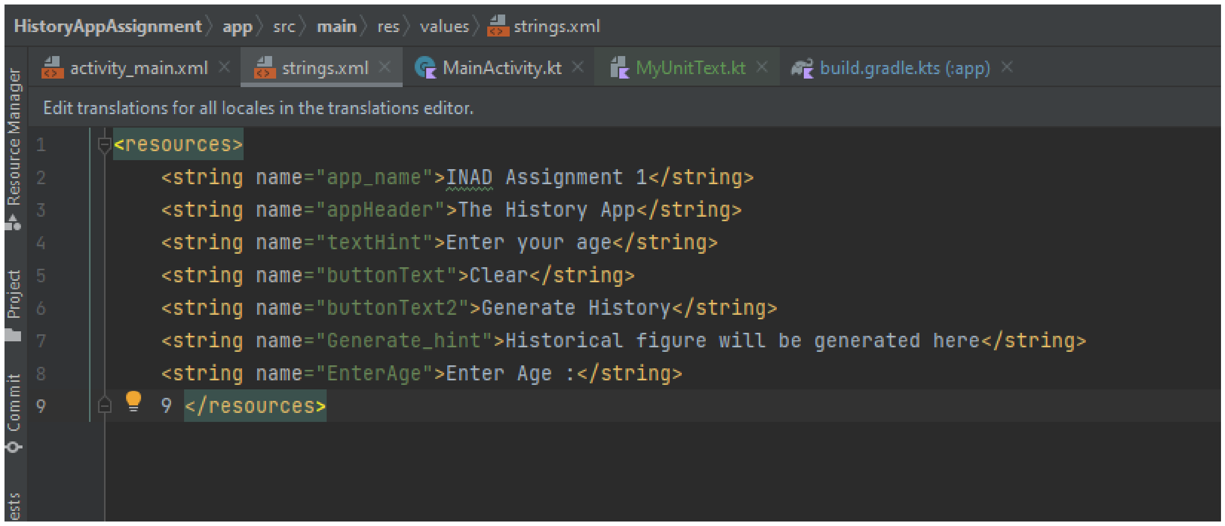
```
} ?: run { this: MainActivity
    message = "Please enter a valid age"
}

// Show the death of the person in the bottom box
textView.text = message
```

Lastly an error check can be done to see the project errors that may be available within the code. This shows the potential errors there could be affecting the application from running. If there are no problems and projects errors, then the application will run effectively.



No errors found by the IDE.
Inspect Code...

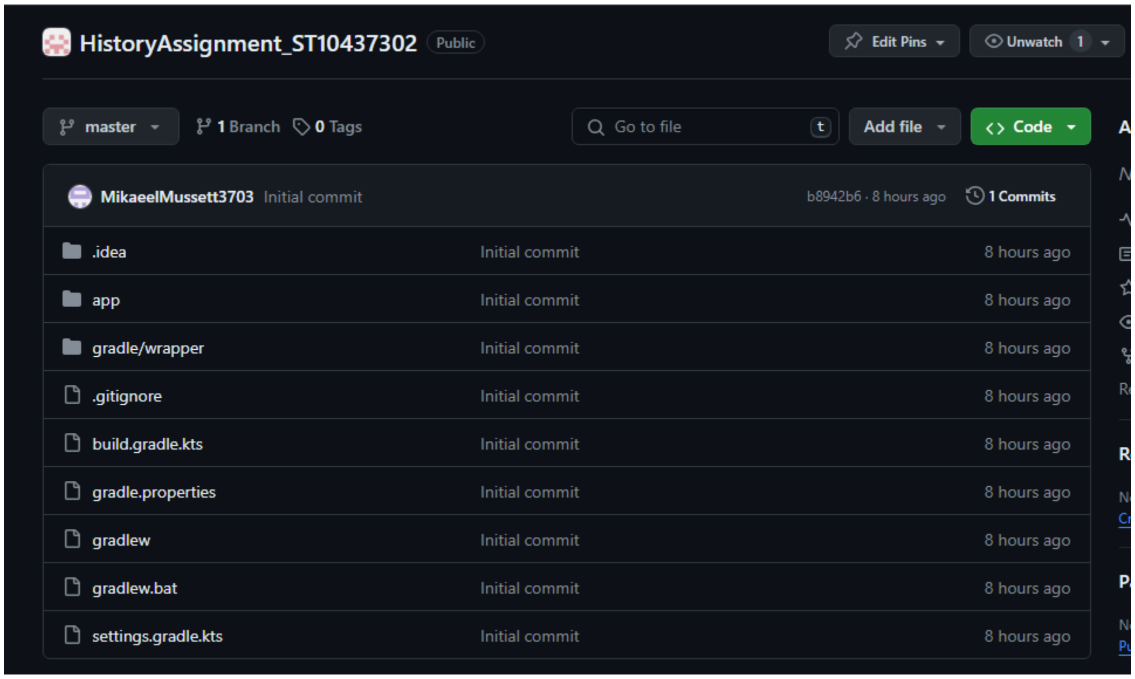
Apart from the code, it is important that you set the string.xml folder as the string resources centralize the text content that is used throughout the application. Each of the strings are referenced at a unique identifier, which allows consistency and maintenance in the codes database.


HistoryAppAssignment 〉 app 〉 src 〉 main 〉 res 〉 values 〉 strings.xml
activity_main.xml strings.xml MainActivity.kt MyUnitText.kt build.gradle.kts (:app)
Resource Manager
Project
Commit

Edit translations for all locales in the translations editor.

```xml
<resources>
    <string name="app_name">INAD Assignment 1</string>
    <string name="appHeader">The History App</string>
    <string name="textHint">Enter your age</string>
    <string name="buttonText">Clear</string>
    <string name="buttonText2">Generate History</string>
    <string name="Generate_hint">Historical figure will be generated here</string>
    <string name="EnterAge">Enter Age :</string>
9 </resources>
```

After ensuring that the code was running effectively, the next part was creating a github repository. This repository allows you to commit and push your code regularly ensuring that there is a backup and disaster recovery in case something has gone wrong. GitHub also provides powerful version control that allows developers to track their changes to code as time goes by. Below shows a screenshot of GitHub storing files that have been uploaded (committed and pushed) so the safe keeping of the code is kept.

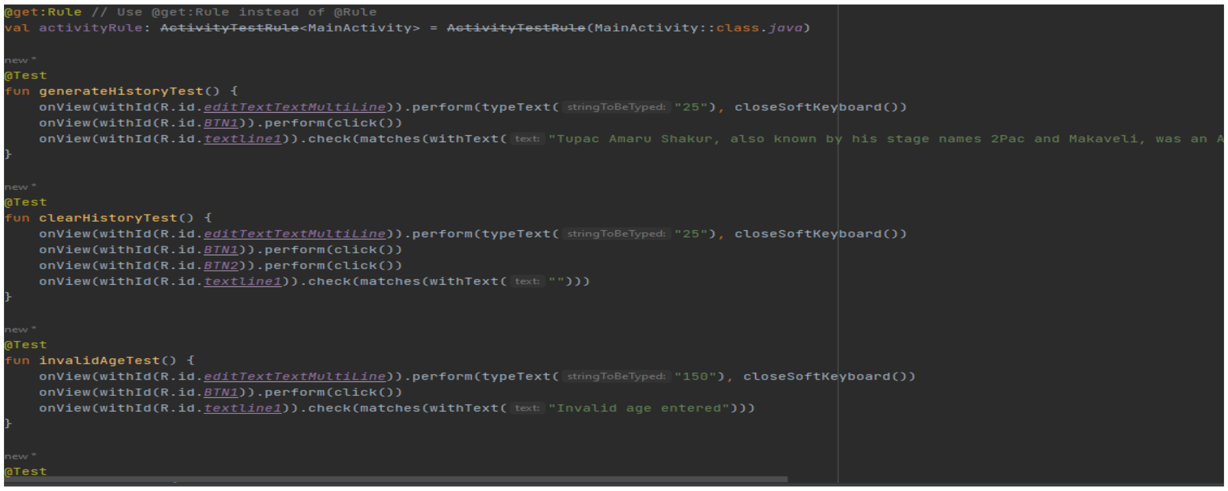

HistoryAssignment_ST10437302 Public

master · 1 Branch · 0 Tags

MikaeelMussett3703 Initial commit · b8942b6 · 8 hours ago · 1 Commits

.idea	Initial commit	8 hours ago
app	Initial commit	8 hours ago
gradle/wrapper	Initial commit	8 hours ago
.gitignore	Initial commit	8 hours ago
build.gradle.kts	Initial commit	8 hours ago
gradle.properties	Initial commit	8 hours ago
gradlew	Initial commit	8 hours ago
gradlew.bat	Initial commit	8 hours ago
settings.gradle.kts	Initial commit	8 hours ago

Testing and automated testing was done through creating a new Kotlin file and doing my unit test in android studio. Below shows the unit testing code that was done and came back with no errors ensuring that the app ran smoothly.


```kotlin
@get:Rule // Use @get:Rule instead of @Rule
val activityRule: ActivityTestRule<MainActivity> = ActivityTestRule(MainActivity::class.java)

new *
@Test
fun generateHistoryTest() {
    onView(withId(R.id.editTextTextMultiLine)).perform(typeText( stringToBeTyped: "25"), closeSoftKeyboard())
    onView(withId(R.id.BTN1)).perform(click())
    onView(withId(R.id.textline1)).check(matches(withText( text: "Tupac Amaru Shakur, also known by his stage names 2Pac and Makaveli, was an A
}

new *
@Test
fun clearHistoryTest() {
    onView(withId(R.id.editTextTextMultiLine)).perform(typeText( stringToBeTyped: "25"), closeSoftKeyboard())
    onView(withId(R.id.BTN1)).perform(click())
    onView(withId(R.id.BTN2)).perform(click())
    onView(withId(R.id.textline1)).check(matches(withText( text: "")))
}

new *
@Test
fun invalidAgeTest() {
    onView(withId(R.id.editTextTextMultiLine)).perform(typeText( stringToBeTyped: "150"), closeSoftKeyboard())
    onView(withId(R.id.BTN1)).perform(click())
    onView(withId(R.id.textline1)).check(matches(withText( text: "Invalid age entered")))
}

new *
@Test
```

The application's purpose was to help educate learners in a fun way. The UI design had specific requirements met to promote the application engagement with the user. The constraints were met with the application having to only work for ages between 20 – 100. An error message needed to be prompted if the age entered was not an integer, the age was entered as a word or an integer with a decimal was entered. The code needed to ensure that these specifications were met and did not force an error in the app. GitHub gained version control so that the code was backed up safely and constant committing and pushing was done so that the code stayed updated and worked. GitActions was a part of the actions used as it allowed the continuation of pushing code and continuous deployment in the repository that was created. After a unit test was conducted until the test rules, no errors came back and the lanch of the application on the testing device was a success as the application ran without any errors.