PA193 super bech32m Phase II

Mikita Valatovich, Illia Kostenko

March 14, 2022

Abstract

This report summarize performing of the PA193 project, Phase II github account.

1 Introduction

In this phase is required to start signing the commits via GPG keys, create starting functionality (encoding/ decoding functionality passing test vectors).

In this project is used Kotlin lang. It is It is robust, statically typed and much less verbose than Java. Also, it is widely used in mobile development, what causes it's high popularity in this segment of development.

2 Setup commitment signing

2.1 Created

For the presenting project were succesfully created and added GPG key, also were performed signed commits.

2.2 Encountered obstacles

At the process of adding the GPG keys was encountered one problem. At the PC of one of the team members were generated GPG keys and they were added to the git configs.

Were performed various actions to solve this issue, but the only one working solution was to use a different computer (for installing GPG keys were performed the same actions!) and to commit securely to the main branch using changes that were pushed to additional branches.

2.3 How this part was created

Were used main recommendations from the Signing commits.

3 Functionality and structure

3.1 Structure of the project

For this project was used object oriented model, that can be use as with Java language as with Kotlin. According to that, structure of the project can be decomposed to the classes, each of that can implement functionality, that can be easily segregated.

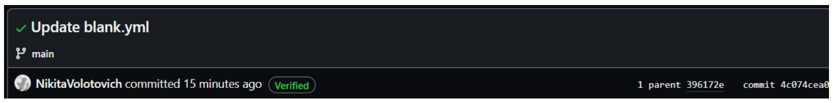
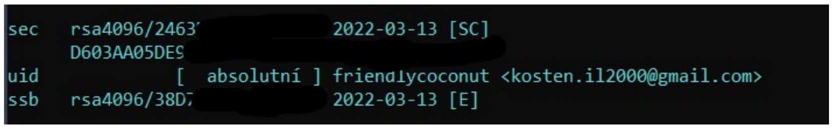
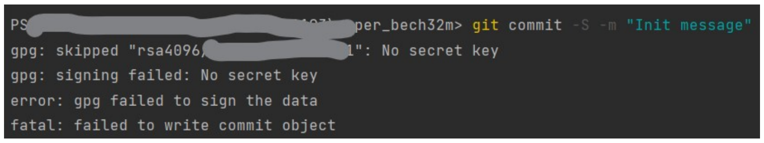

Update blank.yml
main
NikitaVolotovich committed 15 minutes ago Verified
1 parent 396172e commit 4c074cea0

Figure 1: Keys created.



```
sec   rsa4096/2463[redacted]          2022-03-13 [SC]
      D603AA05DE[redacted]
uid                 [  absolutní ] friendlycoconut <kosten.il2000@gmail.com>
ssb   rsa4096/38D[redacted]          2022-03-13 [E]
```

Figure 2: Keys created (Fails).



```
PS[redacted]per_bech32m> git commit -S -m "Init message"
gpg: skipped "rsa4096[redacted]": No secret key
gpg: signing failed: No secret key
error: gpg failed to sign the data
fatal: failed to write commit object
```

Figure 3: Errors at the commit (Fails).

According to that, was decided to implement class BechData (class-structure, that can be used for storing parts of the bech32m string).

Also, at the beginning, it was preferred to create classes CheckSum, Decode, Encode. But, at the process of designing application, was found, that it is not required to create more that one instance of these classes (i.e. for testing), and due to the features of the Kotlin language and therefore were created objects checkSum, Decode, Encode.

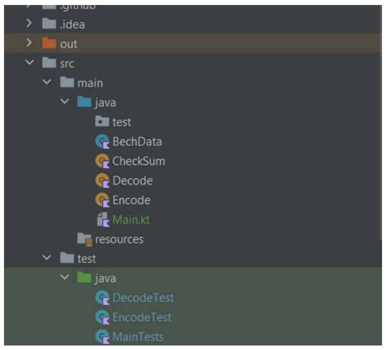
3.2 Implemented functionality

1. Object "Encode", that is used for encoding strings. the only one function 'encode', has as a parameters human readable part (i.e. bp part before the first 1) and ByteArray.
2. Object "Decode", that is used for the decoding bech32m string. The only one parameter is a string that is need to be decoded, as a result we obtain BechData object, that consists of human readable part (string), data (ByteArray).
3. Object "CheckSum", is created for the verifying a checksum, creating a checksum, finding the polynomial with value coefficients mod the generator as 30-bit.

3.3 Added testing methods

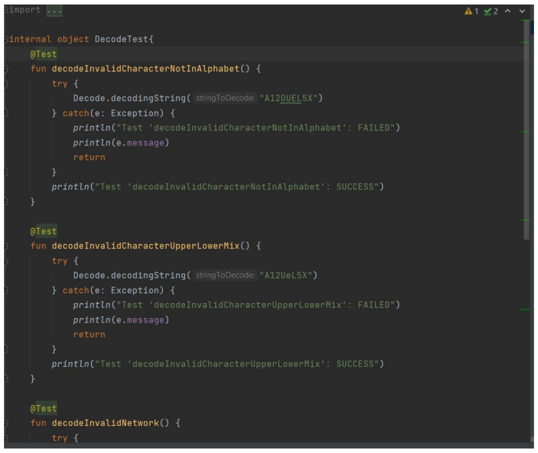
For the testing purposes were created objects: DecodeTest, EncodeTest, MainTests.

1. Object "DecodeTest", that is used for testing decoding method. In this class were implemented methods: decodeInvalidCharacterNotInAlphabet, decodeInvalidCharacterUpperLowerMix, decodeInvalidNetwork, decodeInvalidHumanPart.
2. Object "EncodeTest", that is used for testing encoding methods. The only one test function that is for this test is encodeCorrectInput.



```
> .github
> .idea
> out
v src
  v main
    v java
        test
        BechData
        CheckSum
        Decode
        Encode
        Main.kt
      resources
  v test
    v java
        DecodeTest
        EncodeTest
        MainTests
```

Figure 4: Structure of the project.



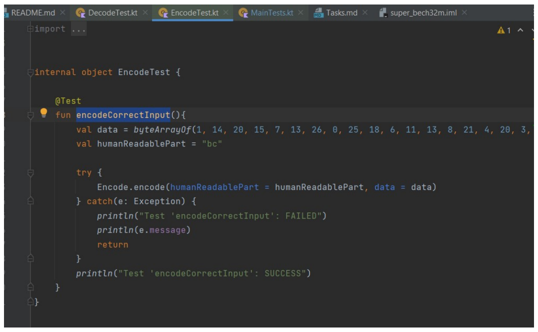
```
import ...

internal object DecodeTest{
    @Test
    fun decodeInvalidCharacterNotInAlphabet() {
        try {
            Decode.decodingString( stringToDecode: "A12OUEL5X")
        } catch(e: Exception) {
            println("Test 'decodeInvalidCharacterNotInAlphabet': FAILED")
            println(e.message)
            return
        }
        println("Test 'decodeInvalidCharacterNotInAlphabet': SUCCESS")
    }

    @Test
    fun decodeInvalidCharacterUpperLowerMix() {
        try {
            Decode.decodingString( stringToDecode: "A12UeL5X")
        } catch(e: Exception) {
            println("Test 'decodeInvalidCharacterUpperLowerMix': FAILED")
            println(e.message)
            return
        }
        println("Test 'decodeInvalidCharacterUpperLowerMix': SUCCESS")
    }

    @Test
    fun decodeInvalidNetwork() {
        try {
```

Figure 5: Decode test.



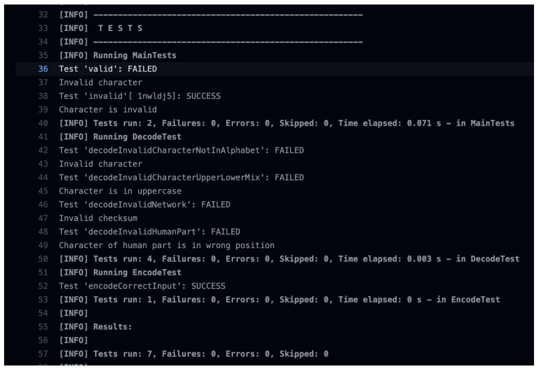
```
import ...


internal object EncodeTest {

    @Test
    fun encodeCorrectInput(){
        val data = byteArrayOf(1, 14, 20, 15, 7, 13, 26, 0, 25, 18, 6, 11, 13, 8, 21, 4, 20, 3,
        val humanReadablePart = "bc"

        try {
            Encode.encode(humanReadablePart = humanReadablePart, data = data)
        } catch(e: Exception) {
            println("Test 'encodeCorrectInput': FAILED")
            println(e.message)
            return
        }
        println("Test 'encodeCorrectInput': SUCCESS")
    }
}
```

Figure 6: encode test.



```
[INFO] ------------------------------------------------------------
[INFO]  T E S T S
[INFO] ------------------------------------------------------------
[INFO] Running MainTests
Test 'valid': FAILED
Invalid character
Test 'invalid'[ 1nwldj5]: SUCCESS
Character is invalid
[INFO] Tests run: 2, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.071 s - in MainTests
[INFO] Running DecodeTest
Test 'decodeInvalidCharacterNotInAlphabet': FAILED
Invalid character
Test 'decodeInvalidCharacterUpperLowerMix': FAILED
Character is in uppercase
Test 'decodeInvalidNetwork': FAILED
Invalid checksum
Test 'decodeInvalidHumanPart': FAILED
Character of human part is in wrong position
[INFO] Tests run: 4, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.003 s - in DecodeTest
[INFO] Running EncodeTest
Test 'encodeCorrectInput': SUCCESS
[INFO] Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0 s - in EncodeTest
[INFO]
[INFO] Results:
[INFO]
[INFO] Tests run: 7, Failures: 0, Errors: 0, Skipped: 0
```

Figure 7: CI output of Maven tests.

3. Object "MainTests", is created for testing if the inputs are valid. These checks were segregated from the checks of the DecodeTest and EncodeTest, because in those test objects are checked if strings are encoded and decoded correctly, but not if they are actually valid.

3.4

Usually the template you're using will have the page margins and paper size set correctly for that use-case. For example, if you're using a journal article template provided by the journal publisher, that template will be formatted according to their requirements. In these cases, it's best not to alter the margins directly.

If however you're using a more general template, such as this one, and would like to alter the margins, a common

4 Added Continuous Integration

4.1 Maven usage and usage of CI

For this project was used Maven as a build automation tool, because of the higher experience of using this tool rather than Gradle. For the running tests in Github actions and checking the results of the performed actions: Java CI with Maven. The main problem at this part was adding Maven framework, because was as issue with reuploading the libraries (some dependencies were unchanged). After every commit CI execute building Maven project during that all tests are execute (fig. 7).