O módulo *Access Control System* é constituído por quatro módulos principais: i) um leitor de teclado, designado por Keyboard Reader; ii) um módulo de interface com o LCD, designado por Serial LCD Controller (SLCDC); iii) um módulo de interface com o mecanismo da porta (Door Mechanism), designado por Serial Door Controller (SDC); e iv) um módulo de controlo, designado por Control. Os módulos i), ii) e iii) são implementados em hardware e o módulo de controlo é implementado em software a executar num PC.

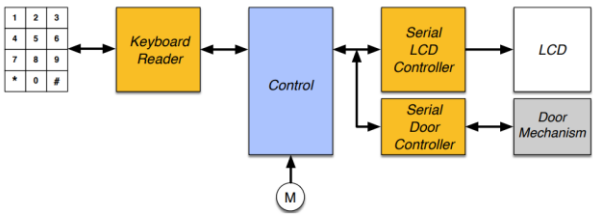

Figura 1 –Arquitetura do sistema que implementa o Sistema de Controlo de Acessos (Access Control System)

1 Access Control System

Esta arquitetura tem como objetivo permitir o controlo a acessos de zonas restritas através de um número de identificação de utilizador (UIN) e um código de acesso associado a esse utilizador (PIN). O sistema permite o acesso à zona restrita (que consiste na abertura da porta) após a inserção correta de um par UIN e PIN. Após o acesso válido, o sistema permite a entrega de uma mensagem de texto ao utilizador.
O utilizador regista o UIN e o PIN a partir do teclado matricial de 12 teclas. O módulo *Keyboard Reader* é responsável pela descodificação da tecla pressionada e disponibiliza o código desta em quatro bits ao *Control*, caso este esteja disponível para o receber. Caso este não esteja disponível para o receber imediatamente, o código da tecla é armazenado até ao limite de nove códigos (nove teclas). O *Control* processa e envia para o SLCDC a informação contendo os dados a apresentar no LCD. A informação para o mecanismo da porta é enviada através do SDC.

2 Interface com o control

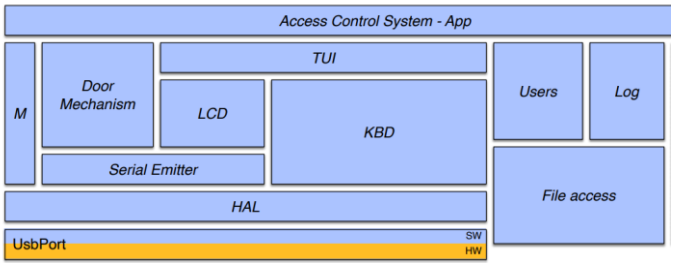

Figura 2 - Interface com o Control

2.1 APP

O módulo APP é responsável por apresentar os dados ao utilizador, recolher as informações que este coloca e processar esses dados, usando as funções criadas nos módulos M, *Door Mechanism*, TUI, *Users* e *Log* . Este módulo é composto por 9 funções sendo estas:

entry() → Responsável por apresentar no LCD informações sobre a data atual, UIN e PIN. Recolhe os dados do UIN e do PIN e verifica se estes são válidos ou não. Caso estes sejam válidos, então é efetuada a abertura e o posteriormente o fecho da porta, caso um destes ou ambos sejam inválidos a porta não irá abrir. Esta função é composta por um while(true) em que só irá sair deste loop quando a variável "flag" (inicializada a false) apresentar o valor true (valor alterado na função manutencao()). Responsável por carregar os valores dos users e adicionar o tempo atual e o id de um user valido à lista logIn.

deleteMensage() → Apresenta no LCD a pergunta "Clear msg?" e caso o utilizador pressione "*" a mensagem será removida caso contrário, a mensagem não é removida.

open() → Função responsável pela abertura da porta a partir da chamada da função open() do *Door Mechanism*. Antes da abertura da porta é possível apagar a mensagem (caso seja pressionado "*") ou alterar o PIN (caso seja pressionado "#").

changePassword() → Quando chamada, aparece a mensagem "change PIN", caso o utilizador não pressione "*" o comando é abortado, caso contrário é pedido ao utilizador que insira um novo PIN, após inserido é pedido o PIN novamente. Caso este seja igual o PIN é alterado, caso seja diferente, o comando é abortado e a palavra passe não é alterada.

close() → Função responsável pelo fecho da porta a partir da função close do *Door Mechanism*.

manutencao() → Esta função verifica se o sistema está em modo manutenção, a partir da chamada à função manutencao()

que está no módulo M. Caso esteja, a variável flag irá ficar com o valor true, caso contrário continua a false.

getFirstIdAvailable() → Função responsável por fornecer o primeiro UIN disponível entre os UIN já criados.

commands() → A função quando chamada só irá parar de ser executada quando o utilizador desligar o modo manutenção. Esta função tem 4 funcionalidades diferentes, sendo o utilizador que define qual dos comandos escolhe. As suas funcionalidades são:

"NEW" → Onde é criado um novo user, em que o utilizador define o nome e a password do mesmo e o UIN é atribuído automaticamente a partir da chamada da função getFirstIdAvailable().

"DEL" → Onde o utilizador introduz o UIN do user que deseja remover. Caso este exista é apresentado o UIN e o nome do user em questão e o utilizador confirma se quer apagar ou não, selecionado "y" ou "n" na consola. Caso o user não exista o comando é abortado.

"MSG" → O utilizador seleciona este comando caso pretenda adicionar uma mensagem, é solicitado o UIN para saber em qual user quer adicionar a mensagem, e caso este exista o utilizador coloca a mensagem no user.

"OFF" → Este comando tem como objetivo desligar o programa, escrever no ficheiro "user.txt" os user's existentes, e escrever no ficheiro "Log.txt" o que está na lista logIn.

main() → Na função main são iniciadas as classes do TUI e *DoorMechanism*. A porta é, inicialmente fechada, é efetuado um while loop a *true* para estar a chamar recorrentemente as funções entry,ao sair dessa função é chamada a função commands e de seguida é necessário voltar a colocar o valor da flag a *false*, caso contrário, ao voltar para a função entry iria-se ter o erro de detetar o estado manunteção a *true*.

2.2 TUI

O ficheiro TUI existe de forma a criar uma ligação entre o LCD e APP, uma vez que a APP só tem acesso ao LCD através deste - como mostra a Figura 2. Assim as funções writeLCD(), setCursor(line: Int, column: Int) e clearLCD() presentes neste ficheiro apenas têm como objetivo, chamar as funções write(), cursor() e clear(), respetivamente, do ficheiro LCD.
Para além das funções descritas acima, foram criadas as seguintes funções:

readInt() → Chama a função waitKey do KBD para aguardar a receção de uma tecla e guarda esse resultado num char. Desta forma, o resultado guardado é convertido no code da tecla definido pela tabela ASCII.

useriD() → Tem como principal objetivo solicitar que o utilizador introduza um id. Assim, primeiro é guardado o id introduzido pelo utilizador de forma a poder ser feita a sua validação. Enquanto o id introduzido pelo utilizador tiver um tamanho igual ou inferior a 2, é criada uma variável char que guarda a chamada da função readInt com um timeout de 5000ms convertido em char. Depois, se cada char introduzido for igual a KBD.NONE, significa que nenhum caractere foi introduzido e por isso o loop é interrompido; se o id não estiver vazio e o char for igual a "*" significa que o utilizador quer limpar o id que foi introduzido e por isso a variável id é redefinida como uma string vazia; por fim, se char for igual a "*" significa que o utilizador terminou de introduzir o id e o loop termina, caso contrário o caractere é adicionado à variável id.

userPIN() → A função userPIN() tem como principal objetivo, solicitar ao utilizador que introduza um PIN. Desta forma, inicialmente é criada uma variável pass que guarda o PIN introduzido pelo utilizador. Posto isto, foi criado um loop onde vai ser lido e guardado cada caractere escrito pelo utilizador chamando a função readInt com um timeout de 5000ms convertido em char. Dependendo do valor de cada char, se o char for igual a KBD.NONE significa que nenhum caractere foi escrito e o loop termina. Caso contrário, se a pass não estiver vazia e o char for igual a "*" significa que o utilizador que apagar o que introduziu e por isso a variável pass é redefinida como uma string vazia.

2.3 USERS

Neste ficheiro, foi implementado um HashMap que guarda utilizadores e o seu id como chave. Foi também implementada uma classe User de forma a poder aceder às propriedades do utilizador sendo estas o id, a password, o nome e a mensagem (pode ser opcional). Dentro desta classe existem dois construtores, o primeiro recebe uma string com o formato "id;password;name;mensagem" que vai, posteriormente, ser repartida por ";" (usando o método split do kotlin) sendo, depois, os valores resultantes atribuídos a cada propriedade. Posto isto, é feita a verificação do tamanho da lista, ou seja, se esta tem um tamanho igual a 5, se for o caso, o quarto valor é atribuído à propriedade mensagem. Caso contrário a propriedade mensagem recebe uma string vazia. Já o segundo construtor, recebe os argumentos id, password, name e mensagem e estes valores são atribuídos diretamente às propriedades correspondentes do User.
Posto isto, dentro deste ficheiro, foram ainda criadas as seguintes funções:

loadUser() → Lê os dados dos utilizadores presentes no ficheiro "USERS.txt" e adiciona-os à userlist (HashMap) retornando, no fim, a userlist.

addID() → Recebe um conjunto de objetos do tipo User e adiciona cada um deles à userlist (HashMap), utilizando o id do utilizador como chave.

getID() → Recebe o id de um utilizador como parâmentro e percorre a userlist à procura de um utilizador com o id correspondente. Se encontrar, retorna o id do utilizador, caso contrário retorna null.

getName() → Recebe um utilizador como parâmetro e percorre a userlist à procura de um utilizador com o nome correspondente. Se encontrar retorna o nome do utilizador, caso contrário retorna null.

write() → Tem como objetivo conseguir relacionar o ficheiro FileAccess com a APP uma vez que esta só consegue aceder a esse ficheiro através do ficheiro Users ou Log como mostra a Figura 2.

2.4 FILE ACCESS

Este ficheiro contém funções que facilitam a leitura e escrita de ficheiros.

createReader() → Cria um objeto BufferedReader que permite a leitura de um ficheiro de texto. Esta recebe o nome do ficheiro como parâmetro e retorna o BufferedReader correspondente.

createWriter() → Cria um objeto PrintWriter que permite a escrita de um ficheiro de texto. Esta recebe o nome do ficheiro como parâmetro e retorna o PrintWrtiter correspondente.

read() → Lê os dados de um ficheiro de texto especificado pelo parâmetro file e retorna um conjunto (HashSet) de objetos do tipo Users.User. Esta função faz uso da função createReader() para criar um BufferedReader e, de seguida, lê cada linha do ficheiro, criando um objeto do tipo Users.User para cada linha e adicionando-o ao conjunto. No fim, o conjunto é retornado.

write() → Escreve os dados presentes no HashMap de objetos Users.User num ficheiro de texto especificado pelo parâmetro file. Esta função usa a função createWriter() para criar um PrintWriter e, de seguida, itera sobre cada entrada do HashMap, obtendo os dados do objeto Users.User e escrevendo-os no ficheiro.

writeLog() → Escreve os dados contidos numa lista mutável de objetos LOG.log num ficheiro de texto especificado pelo parâmetro file. Segue uma lógica semelhante à função write(), utilizando um PrintWriter para escrever dados no ficheiro.

2.5 LOG

Neste ficheiro, foi criada uma classe interna log que é responsável por representar um registo de log. Esta possui duas propriedades, data e uid.
Posto isto, é criada a função **writeLog()** que é responsável por escrever os dados de uma lista mutável de objetos log num ficheiro de texto. Esta função faz uso da função writeLog presente no ficheiro FileAccess para realizar a escrita de logs no ficheiro especificado como parâmetro.

2.6 M

Neste ficheiro foi criada a função manutencao() que verifica se um determinado bit está definido no sinal. Para isso, é utilizada a função isBit() que pertence à entidade HAL. Esta função retorna true se o bit correspondente ao valor hexadecimal 0x40 estiver definido no sinal e retorna false caso contrário.

3 Conclusões

O Access Control System (hardware) juntamente com o software cumpre os requisitos, no sentido em que guarda os utilizadores, não permite o acesso de um utilizador inválido, abre e fecha corretamente a porta, consegue detetar a presença de uma pessoa ao fechar a porta (abrindo-a novamente, e quando parar de detetar a pessoa a porta é fechada novamente).

A. Descrição VHDL do bloco *Access Control System*

```
LIBRARY ieee;
USE ieee.STD_LOGIC_1164.ALL;

entity accessControlSystem is
   port (
        Mclk : in std_logic;
        reset : in std_logic;
        Lines : in std_logic_vector (3 downto 0);
        Columns : out std_logic_vector (2 downto 0);
        rs: out std_logic;
        en: out std_logic;
        data: out std_logic_vector (7 downto 0)
    );

end accessControlSystem;

architecture arq of accessControlSystem is

component keyboard_reader is
    port (
        Mclk : in std_logic;
        reset : in std_logic;
        Kack : in std_logic;
        Lines : in std_logic_vector (3 downto 0);
        Columns : out std_logic_vector (2 downto 0);
        Q : out std_logic_vector (3 downto 0);
        Dval: out std_logic

    );
end component;

component UsbPort is port (
      inputPort:  IN  STD_LOGIC_VECTOR(7 DOWNTO 0);
      outputPort :  OUT  STD_LOGIC_VECTOR(7 DOWNTO 0)
   );
end component;

signal Kack_s, Dval_s: std_logic;
signal Q_s : std_logic_vector(3 downto 0);
signal di_s, oi_s: std_logic_vector (3 downto 0);
signal rt_s, wr_s: std_logic;


begin
   u_keyboard_reader: keyboard_reader port map(
      Kack => Kack_s,
      Lines => Lines,
      Columns => Columns,
      reset => reset,
      Mclk => Mclk,
      Q => Q_s,
      Dval =>  Dval_s
      );

   u_UsbPort: UsbPort port map(
      inputPort(0) => Dval_s,
      inputPort(4 downto 1) => Q_s,
      inputPort(5) => '0',
      inputPort(6) => '0',
      inputPort(7) => '0',
      outputPort(0) => Kack_s,
      outputPort(1) => data(4),
      outputPort(2) => data(5),
      outputPort(3) => data(6),
      outputPort(4) => data(7),
      outputPort(5) => rs,
      outputPort(6) => en
      );

      end arq;
```

B. Código *Kotlin – APP*

```
import LOG.writeLog
import Users.getName
import Users.getPassword
import Users.loadUser
import Users.userlist
import Users.write
import isel.leic.utils.Time
import java.time.*
import java.time.format.*
import java.util.*
import kotlin.collections.HashMap
import kotlin.system.exitProcess
// ver a diferenca entre writeStr e writeLCD
var flag = false
object App {

    val     logIn = emptyList<LOG.log>().toMutableList()

    fun entry() {
        userlist = loadUser()
        while (true) {

            TUI.clearLCD()
            TUI.setCursor(0, 0)
            val formatter = DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm")
            val current = LocalDateTime.now().format(formatter)
                TUI.writeLCD(current)
            TUI.setCursor(1, 0)

            val useriD = TUI.useriD()
            if (useriD == null) {
                manutencao()
                if (flag == true) {
                    break
                }
                continue
            }

            if (useriD != Users.getiD(id = useriD)) {

                TUI.setCursor(1,0)
                TUI.writeLCD("               ")
                TUI.setCursor(1,0)
                TUI.writeLCD("PIN:????")
                Time.sleep(100)
                TUI.userPIN()
                LCD.clear()
                TUI.setCursor(0,2)
                TUI.writeLCD("Login Failed")
                Time.sleep(2500)
                continue
            } else {
                TUI.setCursor(1,0)
                TUI.writeLCD("               ")
                TUI.setCursor(1,0)
                TUI.writeLCD("PIN:????")
                val userPIN = TUI.userPIN()
```

```
        if ((userPIN != userlist[useriD.toString()]?.password) || (userPIN == null)) {
            LCD.clear()
            TUI.setCursor(0,2)
            TUI.writeLCD("Login Failed")
            Time.sleep(2500)
            manutencao()
            if (flag == true) {
                break
            }
            continue
        }

        manutencao()

        if (flag == true) {
            break
        }

        logIn.add(LOG.log(current,useriD))
        open(userlist[useriD.toString()]!!) // ver o porque de ter !!

        close(userlist[useriD.toString()]!!)
    }
  }
}

fun deleteMensage(uid:String){
  TUI.clearLCD()
  TUI.setCursor(0,3)
  TUI.writeLCD("Clear Msg?")
  TUI.setCursor(1,5)
  TUI.writeLCD("(YES=*)")
  val key= TUI.readInt(5000).toChar()
  TUI.clearLCD()
  if ( key == '*'){
    TUI.setCursor(0,2)
    TUI.writeLCD("Msg has been")
    TUI.setCursor(1,4)
    TUI.writeLCD("cleared!")
    Time.sleep(2000)

      if (userlist[uid]?.mensagem != null) {
        userlist[uid]?.mensagem = ""
        // loadUser()
      }

    } else{
      TUI.clearLCD()
      TUI.setCursor(0,3)
      TUI.writeLCD("msg has been")
      TUI.setCursor(1,5)
      TUI.writeLCD("helded!")
    }

}

private fun open(user: Users.User) {
  TUI.clearLCD()
  TUI.setCursor(0,2)
```

```
    TUI.writeStr("welcome back")
    TUI.setCursor(1,2)
    TUI.writeLCD(user.name)

    Time.sleep(2000)
    //Time.sleep(2500)
    //TUI.setCursor(0,6)
    //println(userlist[user.mensagem])
    if (user.mensagem != "") {
        //print("hey")
        TUI.clearLCD()
        TUI.writeStr("${user.mensagem}")
        if (TUI.readInt(5000).toChar() == '*') deleteMensage(user.id.toString())
    }

    if (TUI.readInt(5000).toChar() == '#') changePassword(user.id.toString())
    TUI.clearLCD()
    TUI.setCursor(0,2)
    TUI.writeLCD(user.name)
    TUI.setCursor(1,0)
    TUI.writeLCD("Opening Door...")
    Time.sleep(500)
    DoorMechanism.open(2)
    while (DoorMechanism.finished() != true) {}
    Time.sleep(500)
    TUI.setCursor(1,0)
    TUI.writeLCD("                ")
    TUI.setCursor(1,3)
    TUI.writeLCD("Door Open")
    Time.sleep(500)
}

private fun changePassword(uid:String){
    TUI.clearLCD()
    TUI.setCursor(0,3)
    TUI.writeLCD("Change PIN?")
    TUI.setCursor(1,5)
    TUI.writeLCD("(YES=*)")

    val key= TUI.readInt(5000).toChar()
    TUI.clearLCD()
    if ( key == '*'){
        TUI.setCursor(0,4)
        TUI.writeLCD("Insert New")
        TUI.setCursor(1,0)
        TUI.writeLCD("PIN:????")


        val newPassword= TUI.userPIN()

        TUI.setCursor(0,2)
        TUI.writeLCD("Re-Insert New")
        TUI.setCursor(1,0)
        TUI.writeLCD("PIN:????")
        val confirmPassword = TUI.userPIN()
        if (newPassword==confirmPassword){
            TUI.clearLCD()
            TUI.setCursor(0,3)
            TUI.writeLCD("PIN has been")
```

```
            TUI.setCursor(1,5)
            TUI.writeLCD("changed!")
            Time.sleep(1000)
            TUI.clearLCD()
            if (newPassword != null) {
                userlist[uid]?.password = newPassword
              // loadUser()
            }
        } else{
            TUI.clearLCD()
            TUI.setCursor(0,3)
            TUI.writeLCD("PIN has been")
            TUI.setCursor(1,5)
            TUI.writeLCD("helded!")
        }



    }
}

private fun close (user: Users.User) {
    TUI.setCursor(0,2)
    TUI.writeLCD(user.name)
    TUI.setCursor(1,2)
    TUI.writeLCD("Closing Door")
    Time.sleep(500)
    DoorMechanism.close(2)
    while (DoorMechanism.finished() != true) {}
    TUI.clearLCD()
    TUI.setCursor(0,2)
    TUI.writeLCD(user.name)
    TUI.setCursor(1,2)
    TUI.writeLCD("Door Closed")
    Time.sleep(500)
}

private fun manutencao() {
    if (M.manutencao()) {
        flag = true
        TUI.clearLCD()
        TUI.setCursor(0,1)
        TUI.writeLCD("Out Of Service")
        TUI.setCursor(1,5)
        TUI.writeLCD("Wait")
    }
}

fun getFirstIDAvailable(): String {
    var id = 0
    //var i = "000"
    while (id < userlist.size) {
        if ( id.toString() !in userlist ) {
            return id.toString()
        }
        id++
    }
    return id.toString()
}
```

```
fun commands() {
   println("Turn M key to off, to terminate the maintenance mode.")
   println("Commands: NEW, DEL, MSG, or OFF")
   while (M.manutencao()) {
      print("Maintenance> ")
      val str = readln().uppercase()
      var password: Int

      when (str) {
         "NEW" -> {
            print("User name? ")
            var name = readln()
            while (name.length > 16) {
               println("The $name has more than 16 chars")
               print("User name? ")
               name = readln()
            }
            print("PIN? ")
            password = readln().toInt()
            val id = getFirstIDAvailable()
            val user = Users.User(id.toInt(), password, name)
            userlist.put(id, user)
            println( userlist.put(id, user)?.name)
            println("Adding user ${user.id}:${user.name}")
            userlist.forEach{
            println("${it.value.id}   ${it.value.name}")
            }
         }

         "DEL" -> {
            print("UIN? ")
            val uin = readln()
            val user = userlist.get(uin)
            if (user != null) {
               println("Remove user ${user.id}:${user.name}")
            }
            print("Y/N? ")
            val read = readln()
            if (read == "Y" || read == "y")
               if (user != null) {
                  println("User ${user.id}:${user.name} removed.")
               }
            else
               println("Command aborted.")
            if (user != null) {
               userlist.remove(uin)

            }

         }

         "MSG" -> {
            print("UIN? ")
            val uin = readln().toInt()
            val user = userlist.get(uin.toString())
            println(user)
            print("Message? ")
```

```
                val read = readln()
                if (user != null) {
                    println("The message $read has been associated to ${user.id}:${user.name}")
                    userlist[user.id.toString()]?.mensagem = read
                    //println(userlist[user.id.toString()]?.mensagem)
                }
                //println(" ")

            }

            "OFF" -> {
                write("USERS.txt", userlist)
                writeLog("LOG.txt", logIn)
                //ADICIONAR 1000 USERS
                exitProcess(0)
            }
        }

    }
  }
}

fun main(){
  TUI.init()
  DoorMechanism.init()
  DoorMechanism.close(15)
  while (true) {
    App.entry()
    App.commands()
    flag = false

  }
}
```

C. Código *Kotlin - TUI*

```
object TUI {

    fun readInt(key: Long): Int {
        val char = KBD.waitKey(key)
        return char.code
    }

    fun userID(): Int? {
        var id = ""
        writeStr("ID:???")
        LCD.cursor(1,3)
        while (id.length <= 2) {
            //println("id: ${id.length}")
            val char = readInt(5000).toChar()
            when{
                char == KBD.NONE -> {
                    break
                }
                id.isNotEmpty() && char == '*' -> {
                    id = ""
                    LCD.cursor(1,0)
                    writeStr("ID: ")
                    LCD.cursor(1,3)
                }
                char == '*' -> break
                else -> {
                    id += char
                    LCD.write(char)
                }
            }
        }
        return if (id.length == 3) id.toIntOrNull() else null
    }

    fun userPIN(): Int?{
        var pass = ""
        LCD.cursor(1,4)
        while (pass.length <= 3 ) {
            //println("pass: ${pass.length}")
            val char = readInt(5000).toChar()
            when{
                char == KBD.NONE -> {
                    break
                }
                pass.isNotEmpty() && char == '*' -> {
                    pass = ""
                    LCD.cursor(1,0)
                    writeStr("PIN:????")
                    LCD.cursor(1,4)
                }
                char == '*' -> break
```

```
                    else -> {
                        pass += char
                        LCD.write("*")
                    }
                }
            }
            return if (pass.length == 4) pass.toIntOrNull() else null
        }


        fun writeStr (txt: String) {
            if (txt.length >= 16) {
                for (i in 0 until 16) {
                    LCD.write(txt[i])
                }
                LCD.cursor(1,0)
                for (i in 16 until txt.length) {
                    LCD.write(txt[i])
                }

            } else
                LCD.write(txt)
        }


        fun writeLCD(text: String) {
            LCD.write(text)
        }

        fun init() {
            KBD.init()
            LCD.init()
            LCD.cursor(0, 0)
            //writeStr("batatas")
        }

        fun setCursor(line: Int, column: Int) {
            LCD.cursor(line, column)
        }

        fun clearLCD() {
            LCD.clear()
        }

}

fun main() {
    TUI.init()
}
```

D. Código *Kotlin – USERS*

```
object Users {

    var userlist: HashMap<String, User> = HashMap()

    fun loadUser (): HashMap<String, User> {
        val read = FileAccess.read("USERS.txt")
        addID(read)
        return userlist
    }

    fun read (file: String): HashSet<Users.User> =
        FileAccess.read(file)


    fun addID (read: HashSet<User>) {
        for (user in read) {
            userlist[user.id.toString()] = user
        }
    }
    fun getiD(id: Int): Int? {
        for (user in userlist) {
            if (user.value.id == id) return user.value.id
        }
        return null
    }
    fun getName(name: String): String? {
        for (user in userlist) {
            if (user.value.name == name) return user.value.name
        }
        return null
    }
    fun getPassword(password: Int?): Int? {
        for (user in userlist) {
            if (user.value.password == password) return user.value.password
        }
        return null
    }

    fun write(file: String, mapa: HashMap<String, User>) =
            FileAccess.write(file, mapa)

    class User {
        val id: Int
        var password: Int
        val name: String
        var mensagem: String?

        constructor(str: String) {
            val list = str.split(";")
            //println(list)
            id = list[0].toInt()
            password = list[1].toInt()
            name = list[2]
            mensagem = if (list.size == 5) {
                list[3]
            } else {
                ""
            }
```

```
            //println(password)
            //println(name)
            //println(mensagem)
        }

        constructor(id: Int, password: Int, name: String, mensagem: String = "") {
            this.id = id
            this.name = name
            this.password = password
            this.mensagem = mensagem
            //println(this.password)
        }
    }
}

fun main () {
    Users.User("0;1249;Alan Turing;Turing machine is ready;")
    Users.User("1;2072;George Boole;")
}
```

E. Código *Kotlin* - *FILEACCESS*

```
import java.io.BufferedReader
import java.io.FileReader
import java.io.PrintWriter
import kotlin.collections.HashSet

object FileAccess {

	fun createReader(fileName: String): BufferedReader =
		BufferedReader(FileReader(fileName))

	fun createWriter(fileName: String?): PrintWriter =
		PrintWriter(fileName)

	fun read (file: String): HashSet<Users.User> {
		val br = createReader(file)
		var line: String?
		val list = HashSet<Users.User>()
		line = br.readLine()
		while (line != null) {
			val user = Users.User(line)
			list.add(user)
			line = br.readLine()
		}
		return list
	}


	fun write(file: String, mapa: HashMap<String, Users.User>) {
		val pw = createWriter(file)
		for (dados in mapa) {
			val writer = "${dados.value.id};${dados.value.password};
			${dados.value.name};${dados.value.mensagem};"
			pw.println(writer)
		}
		pw.close()
	}

	fun writeLog(file: String, mapa: MutableList<LOG.log>) {
		val pw = createWriter(file)
		for (log in mapa) {
			val writer = "${log.data} ${log.uid} "
			pw.println(writer)
		}
		pw.close()
	}
}
```

F. Código *Kotlin - LOG*

```
object LOG {
      class log (
            val data: String,
            val uid: Int
      )

      fun writeLog(file: String, lista: MutableList<log>) =
            FileAccess.writeLog(file, lista)
}
```

G. Código *Kotlin - M*

```
object M {
   val signal = 0x40
   fun manutencao(): Boolean =
         HAL.isBit(signal)

}
```