Logo, company name

Description automatically generated

**Licenciatura em Engenharia Informática e de Computadores**

Sistema de Controlo de Acessos

(*Access Control System*)

Vasco Costa (A49412@alunos.isel.pt)

Daniel Pojega (A49521@alunos.isel.pt)

João Ramos (A49424@alunos.isel.pt)

Projeto

de

Laboratório de Informática e Computadores

2022 / 2023 verão

19 de junho de 2023

[1 Introdução 2](#_Toc131410317)

[2 Arquitetura do sistema 3](#_Toc131410318)

[A. Interligações entre o HW e SW 4](#_Toc131410319)

[B. Código *Kotlin* - *HAL* 5](#_Toc131410320)

[C. Código *Kotlin* - *KBD* 6](#_Toc131410321)

[D. Código *Kotlin* - *SerialEmitter* 7](#_Toc131410322)

[E. Código *Kotlin* - *LCD* 8](#_Toc131410323)

[F. Código *Kotlin* - *Door Mechanism* 9](#_Toc131410324)

[G. Código *Kotlin* - *TUI* 10](#_Toc131410325)

[H. Código *Kotlin* - *FileAccess* 11](#_Toc131410326)

[I. Código *Kotlin* - *Users* 12](#_Toc131410327)

[J. Código *Kotlin* - *Log* 13](#_Toc131410328)

[L. Código *Kotlin* da classe *M* 14](#_Toc131410329)

[M. Código *Kotlin* – *Access Control System* *- App* 15](#_Toc131410330)

# Introdução

Neste projeto implementa-se um sistema de controlo de acessos (*Access Control System*), que permite controlar o acesso a zonas restritas através de um número de identificação de utilizador (*User Identification Number – UIN*) e um código de acesso (*Personal Identification Number - PIN*). O sistema permite o acesso à zona restrita 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 sistema de controlo de acessos é constituído por: um teclado de 12 teclas; um ecrã *Liquid Cristal Display* (LCD) de duas linhas de 16 caracteres; um mecanismo de abertura e fecho da porta (designado por *Door Mechanism*); uma chave de manutenção (designada por M) que define se o sistema de controlo de acessos está em modo de Manutenção; e um PC responsável pelo controlo dos outros componentes e gestão do sistema. O diagrama de blocos do sistema de controlo de acessos é apresentado na Figura 1.



Figura 1 – Sistema de controlo de acessos (*Access Control System*)

Sobre o sistema podem-se realizar as seguintes ações em modo Acesso:

* **Acesso -** Para acesso às instalações, o utilizador deverá inserir os três dígitos correspondentes ao *UIN* seguido da inserção dos quatro dígitos numéricos do *PIN*. Se o par *UIN* e *PIN* estiver correto o sistema apresenta no LCD o nome do utilizador e a mensagem armazenada no sistema se existir, acionando a abertura da porta. A mensagem é removida do sistema caso seja premida a tecla ‘\*’ durante a apresentação desta. Todas os acessos deverão ser registados com a informação de data/hora e *UIN* num ficheiro de registos (um registo de entrada por linha), designado por *Log File*.
* **Alteração do PIN –** Esta ação é realizada se após o processo de autenticação for premida a tecla ‘#’. O sistema solicita ao utilizador o novo *PIN*, este deverá ser novamente introduzido de modo a ser confirmado. O novo *PIN* só é registado no sistema se as duas inserções forem idênticas.

**Nota:** A inserção de informação através do teclado tem o seguinte critério: se não for premida nenhuma tecla num intervalo de cinco segundos, o comando em curso é abortado; se for premida a tecla ‘\*’ e o sistema contiver dígitos, elimina todos os dígitos, se não contiver dígitos, aborta o comando em curso.

Sobre o sistema, podem-se realizar também as seguintes ações em modo Manutenção. Ao contrário das ações em modo Acesso, as ações em modo Manutenção são realizadas através do teclado e ecrã do PC. As ações disponíveis neste modo são:

* **Inserção de utilizador -** Tem como objetivo inserir um novo utilizador no sistema. O sistema atribui o primeiro *UIN* disponível, e espera que seja introduzido pelo gestor do sistema o nome e o *PIN* do utilizador. O nome tem no máximo 16 caracteres.
* **Remoção de utilizador -** Tem como objetivo remover um utilizador do sistema. O sistema espera que o gestor do sistema introduza o *UIN* e pede confirmação depois de apresentar o nome.
* **Inserir mensagem -** Permite associar uma mensagem de informação dirigida a um utilizador específico a ser exibida ao utilizador no processo de autenticação de acesso às instalações.
* **Desligar –** Permite desligar o sistema de controlo de acessos. Este termina após a confirmação do utilizador e reescreve o ficheiro com a informação dos utilizadores. Esta informação deverá ser armazenada num ficheiro de texto (com um utilizador por linha) que é carregado no início do programa e reescrito no final do programa. O sistema armazena até 1000 utilizadores, que são inseridos e suprimidos através do teclado do PC pelo gestor do sistema.

**Nota:** Durante a execução das ações em modo manutenção, não podem ser realizadas ações no teclado do utilizador e no LCD deve constar a mensagem “*Out of Service*”.

# Arquitetura do sistema

O controlo (designado por *Control*) do sistema de acessos será implementado numa solução híbrida de *hardware* e *software*, como apresentado no diagrama de blocos da Figura 2. A arquitetura proposta é constituída 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* (S*DC*); e *iv*) um módulo de controlo, designado por *Control*. Os módulos *i*), *ii*) e *iii*) deverão ser implementados em *hardware* e o módulo de controlo deverá ser implementado em *software* a executar num PC.

****

Figura 2 – Arquitetura do sistema que implementa o Sistema de Controlo de Acessos (*Access Control System*)

O módulo *Keyboard Reader* é responsável pela descodificação do teclado matricial de 12 teclas, determinando qual a tecla pressionada e disponibilizando 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. 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*. Por razões de ordem física, e por forma a minimizar o número de sinais de interligação, a comunicação entre o módulo *Control* e os módulos *SLCDC e SDC* é realizada através de um protocolo série.



Figura 3 – Diagrama lógico do Sistema de Controlo de Acessos (*Access Control System*)

1. Interligações entre o HW e SW

Uma imagem com captura de ecrã, Retângulo, texto, diagrama

Descrição gerada automaticamenteUma imagem com captura de ecrã, texto, Retângulo, quadrado

Descrição gerada automaticamente

1. Código *Kotlin* - *HAL*

import isel.leic.UsbPort

// Virtualiza o acesso ao sistema UsbPort

object HAL {

private var lastWriting = 0

// Inicia a classe

fun init() {

UsbPort.write(lastWriting)

}

// Retorna true se o bit tiver o valor lógico ‘1’

fun isBit(mask: Int): Boolean {

val temp = mask and UsbPort.read()

return mask == temp

}

// Retorna os valores dos bits representados por mask presentes no UsbPort

fun readBits(mask: Int): Int = mask and UsbPort.read()

// Escreve nos bits representados por mask o valor de value

fun writeBits(mask: Int, value: Int) {

val a = mask and value

val b = mask.inv() and lastWriting

val c = a or b

UsbPort.write(c)

lastWriting = c

}

// Coloca os bits representados por mask no valor lógico ‘1’

fun setBits(mask: Int) {

writeBits(mask,0xFF)

}

// Coloca os bits representados por mask no valor lógico ‘0’

fun clearBits(mask:Int) {

writeBits(mask,0x00)

}

}

fun main() {

val mask = 0b00001111

HAL.init()

HAL.setBits(mask)

Thread.sleep(2000)

HAL.clearBits(mask)

Thread.sleep(2000)

println(HAL.isBit(mask))

Thread.sleep(2000)

HAL.writeBits(mask, 6)

Thread.sleep(2000)

// Change the value of the input port bits

val currentBits = HAL.readBits(mask)

println(currentBits)

}

1. Código *Kotlin* - *KBD*

import isel.leic.utils.Time

object KBD { // Ler teclas. Métodos retornam ‘0’..’9’,’#’,’\*’ ou NONE.

private const val NONE = 0

private val arrayTeclas = arrayListOf('1', '4', '7', '\*', '2', '5', '8', '0', '3', '6', '9', '#')

// Inicia a classe

fun init() {

HAL.init()

HAL.clearBits(ACK\_MASK)

}

// Retorna de imediato a tecla premida ou NONE se não há tecla premida.

fun getKey(): Char {

if (!HAL.isBit(DVAL\_MASK)) {

return NONE.toChar()

}

val key = HAL.readBits(Q3\_\_0\_MASK)

while (HAL.isBit(DVAL\_MASK)) { HAL.setBits(ACK\_MASK) }

HAL.clearBits(ACK\_MASK)

return arrayTeclas[key]

}

// Retorna a tecla premida, caso ocorra antes do ‘timeout’ (representado em milissegundos), ou NONE caso contrário.

fun waitKey(timeout: Long): Char {

var key = NONE.toChar()

val endTime = Time.getTimeInMillis() + timeout

while (endTime >= Time.getTimeInMillis()) {

key = getKey()

if (key != NONE.toChar()) { break }

}

return key

}

}

fun main() {

KBD.init()

while (true) {

val key = KBD.waitKey(5000)

if (key != 0.toChar()) {

println("Key pressed: $key")

}

}

}

1. Código *Kotlin* – *SerialEmitter*

// Envia tramas para os diferentes módulos Serial Receiver.

object SerialEmitter {

private const val clk: Long = 50

enum class Destination {

LCD,

DOOR

}

// Inicia a classe

fun init() {

HAL.init()

HAL.clearBits(nSDCsel\_MASK)

HAL.clearBits(nLCDsel\_MASK)

HAL.clearBits(SCLK\_MASK)

HAL.clearBits(SDX\_MASK)

}

// Envia uma trama para o SerialReceiver identificado o destino em addr e os bits de dados em ‘data’.

fun send(addr: Destination, data: Int) {

Thread.sleep(clk)

var nSSMask = nLCDsel\_MASK

if (addr == Destination.DOOR) {

nSSMask = nSDCsel\_MASK

while (isBusy()) {

Thread.sleep(clk \* 10)

}

}

HAL.clearBits(nSSMask)

for (i in 4 downTo 0) {

HAL.clearBits(SCLK\_MASK)

val sdx = (data shr i) and 1

if (sdx == 1) HAL.setBits(SDX\_MASK) else HAL.clearBits(SDX\_MASK)

HAL.setBits(SCLK\_MASK)

}

HAL.clearBits(SCLK\_MASK)

HAL.setBits(nSSMask)

Thread.sleep(clk)

}

// Retorna true se o canal série estiver ocupado

fun isBusy(): Boolean = HAL.isBit(BUSY\_MASK)

}

fun main() {

SerialEmitter.init()

for (i in 0..31) {

SerialEmitter.send(SerialEmitter.Destination.LCD, i)

Thread.sleep(250)

}

}

1. Código *Kotlin* - *LCD*

// Escreve no LCD usando a interface a 4 bits.

object LCD {

// Dimensão do display.

private const val LINES = 2

private const val COLS = 16

// Escreve um nibble de comando/dados no LCD em paralelo

private fun writeNibbleParallel(rs: Boolean, data: Int) {

if (rs) HAL.setBits(LCD\_RS\_MASK) else HAL.clearBits(LCD\_RS\_MASK)

HAL.writeBits(LCD\_DATA\_MASK, data)

Thread.sleep(1)

HAL.setBits(LCD\_E\_MASK)

Thread.sleep(1)

HAL.clearBits(LCD\_E\_MASK)

}

// Escreve um byte de comando/dados no LCD em série

private fun writeNibbleSerial(rs: Boolean, data: Int) {

val rsValue = if (rs) 1 else 0

SerialEmitter.send(SerialEmitter.Destination.LCD, data shl 1 or rsValue)

}

// Escreve um nibble de comando/dados no LCD

private fun writeNibble(rs: Boolean, data: Int) = writeNibbleSerial(rs, data)

// Escreve um byte de comando/dados no LCD

fun writeByte(rs: Boolean, data: Int) {

writeNibble(rs, data shr 4)

writeNibble(rs, data)

}

// Escreve um comando no LCD

fun writeCMD(data: Int) = writeByte(false, data)

// Escreve um dado no LCD

fun writeDATA(data: Int) = writeByte(true, data)

// Envia a sequência de iniciação para comunicação a 4 bits.

fun init() {

SerialEmitter.init()

Thread.sleep(16) // Esperar x ms

writeNibble(false, 3)

Thread.sleep(5) // Esperar x ms

writeNibble(false, 3)

Thread.sleep(1) // Esperar x ms

writeNibble(false, 3)

writeNibble(false, 2)

writeCMD(40)

writeCMD(8)

writeCMD(1)

writeCMD(6)

writeCMD(15)

}

// Escreve um caráter na posição corrente.

fun write(c: Char) = writeDATA(c.code)

// Escreve uma string na posição corrente.

fun write(text: String) {

for (c in text) write(c)

}

// Envia comando para posicionar cursor (‘line’:0..LINES-1 , ‘column’:0..COLS-1)

fun cursor(line: Int, column: Int) = writeCMD((line \* 0x40 + column) or 0x80)

// Envia comando para limpar o ecrã e posicionar o cursor em (0,0)

fun clear() {

writeCMD(1)

cursor(0,0)

}

}

fun main() {

LCD.init()

println(" LCD INITIALIZED ")

var count = 0

while (true) {

LCD.write("LCD COUNT: $count")

Thread.sleep(1000)

LCD.clear()

count++

if (count == 10) {

LCD.clear()

LCD.cursor(1, 0)

LCD.write("WE REACHED 10!")

LCD.clear()

count = 0

}

}

}

1. Código *Kotlin* - *DoorMechanism*

// Controla o estado do mecanismo de abertura da porta.

object DoorMechanism {

// Inicia a classe, estabelecendo os valores iniciais.

fun init() {

SerialEmitter.init()

}

// Envia comando para abrir a porta, com o parâmetro de velocidade

fun open(velocity: Int) {

while (!finished()) {

Thread.sleep(1000)

}

SerialEmitter.send(SerialEmitter.Destination.DOOR, velocity shl 1 or 1)

}

// Envia comando para fechar a porta, com o parâmetro de velocidade

fun close(velocity: Int) {

while (!finished()) {

Thread.sleep(1000)

}

SerialEmitter.send(SerialEmitter.Destination.DOOR, velocity shl 1 or 0)

}

// Verifica se o comando anterior está concluído

fun finished() : Boolean = !SerialEmitter.isBusy()

}

fun main() {

DoorMechanism.init()

println("Finished DoorMechanism.init()")

while (true) {

DoorMechanism.open(10)

Thread.sleep(1000)

DoorMechanism.close(5)

}

}

1. Código *Kotlin* - *TUI*

import java.time.LocalDateTime

import java.time.format.DateTimeFormatter

object TUI {

private val DATE\_PATTERN = DateTimeFormatter.ofPattern("dd/MM/yyyy HH:mm")

private var date = LocalDateTime.now()

var dateFormatted = date.format(DATE\_PATTERN)

fun init() {

LCD.init()

KBD.init()

}

fun clearLine(line: Int) =writeText(" ", line)

fun clearScreen() = LCD.clear()

fun waitForKey(time: Long) = KBD.waitKey(time)

fun writeCentralized(text: String, line: Int, clearScreen: Boolean = false) {

if (clearScreen) LCD.clear()

LCD.cursor(line, 8 - text.length / 2)

LCD.write(text)

}

fun writeText(text: String, line: Int, column: Int = 0) {

LCD.cursor(line, column)

LCD.write(text)

}

fun writeDate() {

val currentDate = LocalDateTime.now()

val currentDateFormatted = currentDate.format(DATE\_PATTERN)

if (currentDate.minute != date.minute) {

date = currentDate

dateFormatted = currentDateFormatted.substring(11,16)

updateTime(dateFormatted)

}

}

private fun updateTime(text: String) = writeText(text, 0, 11)

fun writeFailedMessage(message: String) {

clearLine(1)

writeCentralized(message, 1)

clearLine(1)

Thread.sleep(2000)

}

}

1. Código *Kotlin* - *FileAccess*

import java.io.File

object FileAccess {

fun read(fileName: String): List<String> = File(fileName).bufferedReader().use { it.readLines() }

fun write(fileName: String, fileContent: List<String>) = File(fileName).bufferedWriter().use { out ->

out.write(fileContent.joinToString(System.lineSeparator()))

}

fun clear(fileName: String) = write(fileName, listOf(""))

}

fun main() {

val fileName = "Users.txt"

FileAccess.read(fileName).forEach { println(it) }

val fileContent = listOf("3;0;Vasco Costa;")

FileAccess.write(fileName, fileContent)

FileAccess.clear(fileName)

}

/\*

0;1249;Alan Turing;Turing machine is ready;

1;2072;George Boole;

2;0;Maurice Karnaugh;Simplification is possible;

3;0;John von Neumann;

6;0;Kathleen Booth;

\*/

1. Código *Kotlin* - *Users*

object Users {

data class User(val uin: String, var pin: String, val userName: String, var message: String)

private const val USERS\_FILE = "Users.txt"

private val users = mutableListOf<User>()

private val usersToWrite = mutableListOf<String>()

fun init() {

val fileLines = FileAccess.read(USERS\_FILE)

fileLines.forEach { line ->

val u = line.split(";")

require (users.size <= 1000)

users.add(User(u[0], u[1], u[2], u[3]))

}

}

fun isUser(uin: String, pin: String): Boolean = users.any { user -> user.uin == uin && user.pin == pin }

private fun getUserInfo(uin: String): User = users.first { user -> user.uin == uin }

fun getUserPin(uin: String): String = getUserInfo(uin).pin

fun getUserName(uin: String): String = getUserInfo(uin).userName

fun getUserMessage(uin: String): String = getUserInfo(uin).message

fun addUser(pin: String, userName: String): String {

val lastUin = users.last().uin.toInt()

val uin = if (users.isEmpty()) "000"

else {

if (lastUin + 1 >= 100) "${lastUin + 1}"

else if (lastUin + 1 >= 10) "0${lastUin + 1}"

else "00${lastUin + 1}"

}

users.add(User(uin, pin, userName, ""))

return uin

}

fun removeUser(uin: String) = users.removeIf { user -> user.uin == uin }

fun changeUserPin(uin: String, newPassword: String) = users.first { user -> user.uin == uin }.apply { pin = newPassword }

fun changeUserMessage(uin: String, newMessage: String) = users.first { user -> user.uin == uin }.apply { message = newMessage }

private fun toString(user: User): String = "${user.uin};${user.pin};${user.userName};${user.message}"

fun writeUsers() {

users.forEach {

usersToWrite.add(toString(it))

}

FileAccess.write(USERS\_FILE, usersToWrite)

}

fun clearUsersFile() = FileAccess.clear(USERS\_FILE)

}

fun main() {

println(Users.getUserPin("6"))

println(Users.getUserName("6"))

println(Users.getUserMessage("6") == "")

FileAccess.read("Users.txt").forEach { println(it) }

val uin = Users.addUser("1234", "Vasco Costa")

Users.changeUserPin("2", "4321")

Users.changeUserMessage(uin, "Your car is ready")

Users.removeUser("6")

Users.clearUsersFile()

println(FileAccess.read("Users.txt").isEmpty())

Users.writeUsers()

FileAccess.read("Users.txt").forEach { println(it) }

}

1. Código *Kotlin* - *Log*

import java.text.SimpleDateFormat

import java.util.\*

object Log {

data class Log(val uin: String, val date: String, var time: String)

private const val LOG\_FILE = "LogFile.txt"

private val log = mutableListOf<Log>()

private val logsToWrite = mutableListOf<String>()

private val dateFormat = SimpleDateFormat("dd/M/yyyy")

private val timeFormat = SimpleDateFormat("hh:mm:ss")

fun init() {

clearLogFile()

}

fun addLog(uin: String) = log.add(Log(uin, dateFormat.format(Date()), timeFormat.format(Date())))

fun writeLog() {

log.forEach { logsToWrite.add("${it.uin};${it.date};${it.time};") }

FileAccess.write(LOG\_FILE, logsToWrite)

}

fun clearLogFile() = FileAccess.clear(LOG\_FILE)

}

fun main() {

Log.addLog("123")

Thread.sleep(1000)

Log.addLog("456")

Thread.sleep(1000)

Log.addLog("789")

Thread.sleep(1000)

Log.writeLog()

FileAccess.read("LogFile.txt").forEach { println(it) }

}

1. Código *Kotlin* da classe *M*

object M {

// verificar se o bit M está ativo através do HAL

fun init() {

HAL.init()

}

fun verify() = HAL.isBit(M\_MASK)

}

1. Código *Kotlin* – *Access Control System* *- App*

import kotlin.system.exitProcess

object App {

fun init() {

M.init()

DoorMechanism.init()

TUI.init()

Users.init()

Log.init()

}

fun run() {

while (true) {

DoorMechanism.close(10)

TUI.writeText(TUI.dateFormatted, 0)

while (!M.verify()) {

TUI.writeDate()

try {

getAccess()

} catch (e: Exception) {

when(e.message) {

"Login failed" -> {

TUI.writeFailedMessage("Login failed")

getAccess()

}

"Command aborted" -> {

TUI.writeFailedMessage("Command aborted")

getAccess()

}

else -> getAccess()

}

}

}

TUI.writeCentralized("Out of service", 0, true)

TUI.writeCentralized("Wait", 1)

println("Turn M key to off, to terminate the maintenance mode.")

println("Commands: NEW, DEL, MSG, or OFF")

while (true) {

print("Maintenance> ")

when (readln().toUpperCase()) {

"NEW" -> Others.exceptionHandler { addUser() }

"DEL" -> Others.exceptionHandler { deleteUser() }

"MSG" -> Others.exceptionHandler { addMessage() }

"OFF" -> {

Others.exceptionHandler { turnOff() }

exitProcess(0)

}

else -> println("Unknown command.")

}

}

}

}

private fun getAccess() {

while (true) {

TUI.writeDate()

val uinText = "???".toCharArray()

TUI.writeText("UIN:${String(uinText)}", 1)

val uin = getUin(uinText)

val pinText = "????".toCharArray()

TUI.writeText("PIN:${String(pinText)}", 1)

val pin = getPin(pinText)

TUI.writeDate()

require(Users.isUser(uin, pin)) { "Login failed" }

access(uin)

return

}

}

private fun getUin(uin: CharArray): String {

for (colIdx in 4..6) {

uin[colIdx - 4] = TUI.waitForKey(5000)

require(uin[colIdx - 4] != 0.toChar()) { "Command aborted" } // No key pressed

require(uin[0] != '\*') { "Command aborted" }

require(uin[colIdx - 4] != '#') { "Invalid key" }

TUI.writeText("${uin[colIdx - 4]}", 1, column = colIdx)

}

return String(uin)

}

private fun getPin(pin: CharArray): String {

for (colIdx in 4..7) {

pin[colIdx - 4] = TUI.waitForKey(5000)

require(pin[colIdx - 4] != 0.toChar()) { "Command aborted" } //No key pressed

require(pin[0] != '\*') { "Command aborted" }

require(pin[colIdx - 4] != '#') { "Invalid key" }

TUI.writeText("\*", 1, column = colIdx)

}

return String(pin)

}

private fun access(uin: String) {

Log.addLog(uin)

Log.writeLog()

TUI.clearScreen()

TUI.writeCentralized("Hello", 0)

val userName = Users.getUserName(uin)

TUI.writeCentralized(userName, 1)

Users.getUserMessage(uin).also { message ->

TUI.clearScreen()

if (message != "") {

if (message.length >= 17) {

TUI.writeCentralized(message.substring(0, 16), 0)

TUI.writeCentralized(message.substring(17), 1)

}

else TUI.writeCentralized(message, 0, true)

TUI.clearScreen()

if (TUI.waitForKey(2000) == '\*') Users.changeUserMessage(uin, "")

}

}

if (TUI.waitForKey(2000) == '#') changePIN(uin)

TUI.writeText(userName, 0, 0)

doorManagement()

}

private fun changePIN(uin: String) {

try {

val pinText = "????".toCharArray()

val firstPin = getPin(pinText)

val secondPin = getPin(pinText)

require (firstPin == secondPin) { "PINs do not match" }

Users.changeUserPin(uin, firstPin)

TUI.writeText("New pin is $firstPin", 1)

} catch (e: Exception) {

when(e.message) {

"Command aborted" -> return

"Invalid key" -> changePIN(uin)

"PINs do not match" -> {

TUI.writeFailedMessage("PINs do not match")

return

}

}

}

}

private fun doorManagement() {

TUI.writeCentralized("Door opening", 1)

DoorMechanism.open(10)

while (!DoorMechanism.finished()) {

// waiting to open

}

TUI.clearLine(1)

TUI.writeCentralized("Door open", 1)

Thread.sleep(2000)

TUI.clearLine(1)

TUI.writeCentralized("Door closing", 1)

DoorMechanism.close(5)

while (!DoorMechanism.finished()) {

// waiting to close

}

TUI.clearLine(1)

TUI.writeCentralized("Door closed", 1)

TUI.clearScreen()

}

private fun addUser() {

print("User name? ")

val userName = readln()

require(userName.length <= 16) { "Username must be less or equal than 16 characters." }

print("PIN? ")

val pin = readln()

require(pin.length == 4 && pin.all { char -> char.isDigit() }) { "Pin must have 4 numbers." }

val user = Users.addUser(pin, userName)

println("Adding user $user:$userName")

}

private fun deleteUser() {

print("UIN? ")

val uin = readln()

val userName = Users.getUserName(uin)

println("Removing user $uin:$userName")

print("Y/N? ")

when (readln().toUpperCase()) {

"Y" -> {

Users.removeUser(uin)

println("User $uin:$userName removed.")

}

"N" -> println("Command aborted.")

else -> println("Unknown command.")

}

}

private fun addMessage() {

print("UIN? ")

val uin = readln()

val msg = Users.getUserMessage(uin)

val userName = Users.getUserName(uin)

if (msg != "") {

println("User has this message: $msg.")

print("Remove this message Y/N? ")

when (readln().toUpperCase()) {

"Y" -> {

Users.changeUserMessage(uin, "")

addNewMessage(uin, userName)

}

"N" -> println("Command aborted.")

else -> println("Unknown command.")

}

}

else {

addNewMessage(uin, userName)

}

}

private fun addNewMessage(uin: String, userName: String) {

print("Message? ")

val newMsg = readln()

Users.changeUserMessage(uin, newMsg)

println("The message \"$newMsg\" has been associated to $uin:$userName.")

}

private fun turnOff() {

Users.clearUsersFile()

Users.writeUsers()

}

}

fun main() {

App.init()

App.run()

}