Para a consumação do trabalho, foi necessário desenvolver uma *App* que englobava todo o trabalho no seu todo, usando as varias camadas abaixo, que por si só, já usavam camadas mais baixas. Com isto têm-se uma parte *Hardware* e uma parte de *Software* que permite controlar estes elementos, acrescentando algo para uma melhor iteração com o utilizador. A Figura 1, demonstra o resultado final pretendido, em forma de esquema.

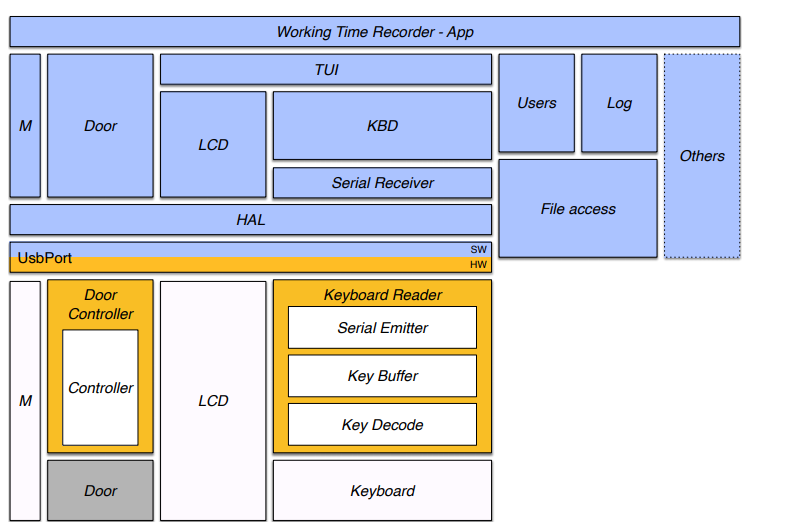
****

Figura 1- Diagrama lógico do Relógio de Ponto

# *File Access*

Considerando que este sistema está implementado numa empresa, é necessário termos uma lista com os empregados, com as informações respetivas. O seu *User, Pass,* Nome, tempo acumuladoe data entrada. Tambem é necessário ler e escrever e ler os *logs*(registos), de entradas e saídas dos trabalhadores, para saber se ele está dentro da empresa ou fora, por exemplo. Para isto, criou-se uma classe File Access que permite ler ficheiros *.txt,* e escrever neles. Para isto usou-se varias funções do *kotlin,* tais como, o *inputStream(), bufferReader()* e *o fileWriter.* (A implementação teve por base o enunciado do trabalho). O respetivo código implementado, , encontra-se no anexo A

# Users

Aproveitando a objeto desenvolvido em cima, foi criado um objeto, que permite a manipulação dos *user*, tais como adicionar um novo trabalhador, remover, ou até mesmo dar lhe *update*, a algum dos seus elementos. Para além destas funções mais elementares, ainda se tem a possibilidade, de só retornar o user, dar *init* ao objeto e dar *update* a lista. Esta lista terá o tamanho máximo de 1000 utilizadores diferentes. O código respetivo implementado, encontra-se no anexo B

# LogFile

Para registar as entradas ou saídas dos trabalhadores, para se saber quantas vezes veio trabalhar, ou até mesmo, para o sistema saber se o individuo encontra-se fora ou dentro da empresa, é necessário um objeto que permita, num ficheiro *.txt,* anotar estes registos. Para isso é necessário saber-se as horas, dia, mês, ano, e dia da semana que foi efetuado o registo. Para isso, criou-se funções que permitissem essa manipulação dos horários. Agora é necessário, de acordo com o registo saber se o trabalhador está fora ou dentro, ou seja se estiver dentro, quer dizer que vai sair e é preciso indicar este mesmo individuo, saiu as tais horas, e quantas horas trabalhou. Caso queira entrar na empresa, é necessário registar que vai entrar a tais horas. É fácil perceber se ele saiu ou entrou devido aos símbolos “->”(indicação de entrada) ou “<-“ (indicação de saída). O respetivo código implementado, , encontra-se no anexo C

Com estes três elementos, temos a partes dos *Users* e *Log* implementadas, tal como na figura 2

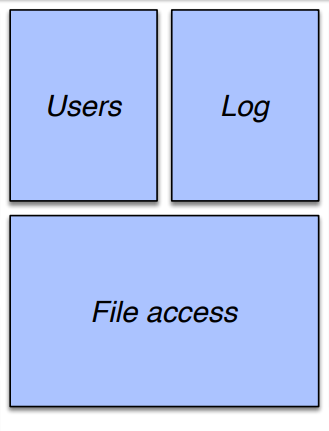


Figura 2-- Diagrama lógico dos *Users* e *Log*

# Maintenence

Para se poder adicionar, remover, desligar ou ver a lista de *Users,* é necessário colocar o sistema em modo de manutenção, através do botão M (Em laboratório, foi usado o *switch* 4 da ATB). Neste estado, no ecrã do computador, irá aparecer uma interface, que permite uma de cinco opções:

* addUser()-Permite adicionar um user
* removeUser()- Permite remover um user
* userList()- Mostra a userList corrente
* APP.mode()- Permite voltar ao modo de registo
* shutDown()-Permite desligar o sistema

Todas estas opções são funções que constituem este objeto. Caso o *user* tente dar um valor que não corresponde a nenhuma destas opções irá aparecer uma mensagem de erro. O código respetivo implementado, encontra-se em anexo D.

# APP

Na *App,* já munida com todas as funções das camadas inferiores, criou-se funções que juntassem estes vários componentes, e permitissem criar um algoritmo coerente. Como é a função que tem mais iteração com o utilizador, é necessário confirmar e testar vários aspetos, tais como: Se o utilizador acertou a palavra passe de acordo com user que está na base de dados. Caso acerte, mostrar, de acordo se está a entrar ou a sair, as informações correspondentes. No fim de colocar a palavra passe, é deixado um tempo de 2 segundos, para o utilizador, carregar no ‘#’ para poder atualizar a sua palavra passe. Nesta fase é necessário escrever 2 vezes iguais a palavra passe nova, caso erre, tem de voltar ao inicio e escrever outra vez a nova palavra passe. O utilizador, tem um espaço de cinco segundos para meter a palavra passe, senão volta ao inicio e pede o *user.* Caso engane-se e queira escrever de novo, carrega no ‘\*’, que apaga o que já foi escrito e permite escrever outra vez. Caso acertemos um *user* e estamos na fase de pôr uma palavra passe, caso não tenha nada escrito, e carregue-se no ‘\*’, o sistema interrompe o sistema e volta a pedir o *user*. O código respetivo implementado, encontra-se em anexo E.

# Conclusões

Com *App* concretizada, fechamos o nosso trabalho. Foi um trabalho que exigiu, muita dedicação e cooperação entre os elementos do grupo, que em conjunto, fomos comaltando as falhas de cada um. Este projeto permitiu aprofundar conhecimentos obtidos no semestre anterior, com novas técnicas de trabalho, que são muito mais proveitosas para um mundo professional mais exigente. Seguindo o planeamento e preparando as aulas durante a semana, fomos capazes de cumprir os prazos propostos, com alguns contratempos, mas tudo ultrapassável. A parte que nos deu mais trabalho, foi a *door.* O seu funcionamento, não correspondia ao que era suposto, e demoramos mais tempo do que desejávamos. De resto, damos como alcançado o objetivo do trabalho, claro com partes que podem ser melhoradas, mas com um aproveitamento positivo.

1. Código do FileAccess

*object* FileAcess {  
 *fun* read(file:String): Array<String?> {  
 *val* list = File(file).inputStream()  
 *val* lineList = *arrayOfNulls*<String>(1000)  
 list.bufferedReader().useLines { lines -> lines.forEach { lineList[it.split(';')[0].toInt()] = it } }  
 *return* lineList  
 }  
  
 *fun* write(file:String, list: List<String>,logNUsers:Boolean) {  
  
 *val* text = FileWriter(file, logNUsers)  
 *for* (i *in* list.*indices*) {  
 *if* (logNUsers) text.append("\n${list[i]}")  
 *else* text.append(list[i])  
 }  
 text.close()  
 }  
}  
  
  
*fun* main(){  
 LogFile.awayUser(Ut(0,0,"ei",0,0L),656)  
}

1. Códigos dos Users

*data class* Ut(*val* user:Int, *var* pass:Int, *val* name:String, *var* acumulateTime:Long, *var* entryTime:Long)  
  
*object* Users {  
  
 *private var* userlist= *arrayOfNulls*<Ut>(1000)  
  
 *fun* init(){  
 listUser()  
 }  
  
 *private fun* toUser(userId:Int, it: List<String>):Ut{  
 *return* Ut(userId,it[1].toInt(),it[2],it[3].toLong(),it[4].toLong())  
 }  
  
 *fun* listUser(): Array<Ut?> {  
 *val* newList = userlist  
  
 *var* i = 0  
 FileAcess.read("USERS.txt").forEach **{** line **->** *if* (line != *null*){  
 *val* user = line.split(';')  
 *val* userId =user[0].toInt()  
 newList[userId]= toUser(userId,user)  
 i++  
 }  
 **}** *return* newList  
 }  
  
 *fun* removeUser(userId:Int){  
 *val* userList = listUser()  
 userList[userId] = *null* }  
  
 *fun* addUser(userPass:Int, userName:String):Ut?{  
 *for*(i *in* 0..999){  
 *if* (userlist[i] == *null*) {  
 userlist[i] = Ut(i,userPass,userName,0,0L)  
 *return* userlist[i]!!  
 }  
 }  
 *return null* }  
  
 *fun* getUser(indiceUser: Int): Ut? {  
 *val* list = userlist  
 *if* (indiceUser == -1) *return null  
 return* list[indiceUser]  
 }  
  
 *fun* updateList(){  
 *val* lista = *arrayOfNulls*<String>(1000)  
 *val* lastList = userlist  
 *for* (i *in* lastList.*indices*) {  
 *val* user = lastList[i]  
 *if* (user != *null*) {  
 lista[i]= "${i};${user.pass};${user.name};${user.acumulateTime};${user.entryTime}\n"  
 }  
 }  
 FileAcess.write("USERS.txt",lista.filterNotNull().toList(),*false*)  
 }  
  
 *fun* updateUser(userId: Int,userAcumulateTime:Long,userEntryTime:Long){  
 *val* list = userlist  
 list[userId]!!.acumulateTime=userAcumulateTime  
 list[userId]!!.entryTime=userEntryTime  
 updateList()  
 }  
}  
  
*fun* main(){  
 Users.init()  
 Users.addUser(123,"Teste123")  
 Users.updateList()  
  
}

1. Codigo do LogFile

*object* LogFile {  
 *private var* listLog= *mutableListOf*<String>()  
  
  
 *fun* entryUser(worker:Ut,time:Long){  
 *val* calendar = calendarLog(time)  
 *val* user = "-> ${worker.user}:${worker.name}"  
 *val* entry = calendar + user  
 listLog += entry  
 logUpdate(listLog.toList())  
 }  
  
 *fun* logUpdate(list: List<String>){  
 FileAcess.write("LOG.txt",list,*true*)  
 }  
  
  
 *fun* awayUser(worker:Ut,time:Long){  
 *val* calendar = calendarLog(time)  
 *val* user = "<- ${worker.user}:${worker.name}"  
 *val* away = calendar + user  
 listLog += away  
 logUpdate(listLog.toList())  
 }  
  
 *fun* calendarLog(time:Long):String{  
 *val* calendar=Calendar.getInstance()  
 calendar.setTimeInMillis(time)  
  
 *val* ampm = calendar.get(Calendar.*AM\_PM*)  
 *val* hour = timeAmPm(ampm,calendar.get(Calendar.*HOUR*))  
 *val* minute= calendar.get(Calendar.*MINUTE*)  
  
 *val* day = calendar.get(Calendar.*DATE*)  
 *val* month = calendar.get(Calendar.*MONTH*) + 1 */\*\* Because indeces \*\*/  
 val* year = calendar.get(Calendar.*YEAR*)  
  
 *return* "$day/$month/$year $hour:$minute "  
 }  
  
 *fun* calendarAway(time: Long):String{  
 *val* calendar=Calendar.getInstance()  
 calendar.setTimeInMillis(time)  
  
 *val* dayWeek = intToDay(calendar.get(Calendar.*DAY\_OF\_WEEK*))  
 *val* ampm = calendar.get(Calendar.*AM\_PM*)  
 *val* hour = timeAmPm(ampm,calendar.get(Calendar.*HOUR*))  
 *val* minute= calendar.get(Calendar.*MINUTE*)  
  
 *return* "$dayWeek.$hour:$minute "  
 }  
  
 *fun* timeAmPm(ampm:Int,time: Int):Int{  
 *return if* (ampm == 1 ) time + 12  
 *else* time  
 }  
  
 *fun* intToDay(day:Int):String{  
 *return when*(day){  
 1 -> "Sun"  
 2 -> "Mon"  
 3 -> "Tue"  
 4 -> "Wed"  
 5 -> "Thu"  
 6 -> "Fri"  
 *else* -> "Sat"  
 }  
 }  
}

1. Código do Maintenance

*object* Maintenance {  
  
 *fun* init() {  
 LCD.clear()  
 TUI.writecenter("Out Of Service...", 0)  
 TUI.writecenter("\*\_\*", 1)  
 systemModem()  
 }  
  
 *private fun* systemModem() {  
 *println*("What operation you want to proceed? \n 1 - Add User \n 2 - Remove User \n 3 - User List \n 4 - Change Mode \n 5 - ShutDown ")  
 *when* (*readLine*()!!) {  
 "1" -> addUser()  
 "2" -> removeUser()  
 "3" -> userList()  
 "4" -> APP.mode()  
 "5" -> shutDown()  
 *else* -> operationNotFound()  
 }  
 }  
  
  
 *private fun* operationNotFound() {  
 *println*("Operation Not Found \n ")  
 systemModem()  
 }  
  
 *private fun* addUser() {  
 *val* name = name()  
 *val* pass = pass()  
 *val* add = Users.addUser(pass, name)  
  
 *if* (add != *null*) *println*("You user is: $add")  
 *else println*("Sorry, userList is full xOxO")  
 systemModem()  
 }  
  
 *private fun* name(): String {  
 *print*("UserName(Máx 16 Chars): ")  
 *val* name = *readLine*()!!  
 *if* (name.length > 16) {  
 *println*("Please choose one UserName with maxium 16 chars")  
 name()  
 }  
 *return* name  
 }  
  
 *private fun* pass(): Int {  
 *print*("UserPass(4 Chars): ")  
 *val* pass = *readLine*()!!  
 *if* (pass.length != 4) {  
 *println*("Please choose one UserPass with 4 chars")  
 pass()  
 }  
 *return* pass.toInt()  
 }  
  
 *fun* removeUser() {  
 *println*("Insert UserID")  
 *val* userId = *readLine*()!!.toInt()  
 *val* user = Users.getUser(userId)  
 *if* (user != *null*) {  
 *println*(user.name + ",Is this the User to remove? Yes or No")  
 *val* confirmation = *readLine*()!!  
 *if* ("Y" *in* confirmation || "y" *in* confirmation) {  
 Users.removeUser(userId)  
 }  
  
 } *else* {  
 *println*("User Not Found \n")  
 }  
 systemModem()  
 }  
  
 *private fun* userList() {  
 Users.listUser().filterNotNull().forEach **{** *println*(**it**) **}** systemModem()  
 }  
  
 *fun* shutDown() {  
 LCD.off()  
 Users.updateList()  
 }  
}  
  
*fun* main(){  
 *// LCD.init()  
 //Maintenance.init()  
 //Maintenance.userList()* Users.init()  
 Maintenance.removeUser()  
 Maintenance.shutDown()  
  
}

1. Código App

*object* APP {  
 *private const val* DOOR\_OPEN\_VELOCITY= 11  
 *private const val* DOOR\_CLOSE\_VELOCITY= 11  
 *private const val* MMASK = 0x80  
 *private const val* TENTNUMB = 3  
 *private const val* WRITEACTIONSLINE = 1  
  
  
 *private fun* user():Ut {  
 TUI.writeleft("USER:", WRITEACTIONSLINE)  
 *val* userNumb = TUI.key(3, *true*)  
 *if* (userNumb == -1) appPlay()  
 *if* (getUser(userNumb)!= *null*) {  
 *return* getUser(userNumb)!!  
 } *else* {  
 TUI.lineClear(WRITEACTIONSLINE)  
 TUI.writeleft("USER NOT FOUND", WRITEACTIONSLINE)  
 Time.sleep(1000)  
 TUI.lineClear(WRITEACTIONSLINE)  
 *return* user()  
 }  
 }  
  
 *private fun* pass(){  
 *val* user = user()  
 TUI.lineClear(WRITEACTIONSLINE)  
 *for* (i *in* 1..TENTNUMB) {  
 TUI.writeleft("PASS:", WRITEACTIONSLINE)  
 *val* code = TUI.key(4, *false*)  
 *if* (code == -1) appPlay()  
 *if* (code == user.pass) {  
 verifyChangePass(user)  
 *return* doorAction(user)  
 }  
 TUI.lineClear(WRITEACTIONSLINE)  
 TUI.writeleft("PASS ERROR", WRITEACTIONSLINE)  
 Time.sleep(1000)  
 TUI.lineClear(WRITEACTIONSLINE)  
 }  
 *return* wrongPass()  
 }  
 *private fun* verifyChangePass(user:Ut) {  
 *if* (KBD.waitKey(2000) == '#') changePass(user)  
 }  
  
 *private fun* changePass(user:Ut){  
 LCD.clear()  
 TUI.writecenter("NEW PIN",0)  
 TUI.writeleft("PIN:",1)  
 *val* code1 = TUI.key(4,*false*)  
 *if* (code1 == -1) appPlay()  
  
 LCD.clear()  
 TUI.writecenter("CONFIRM PIN",0)  
 TUI.writeleft("PIN:",1)  
 *val* code2 = TUI.key(4,*false*)  
 *if* (code2 == -1) appPlay()  
  
 LCD.clear()  
 *if* (code1==code2){  
 user.pass=code1  
 TUI.writecenter("PIN CONFIRMED",0)  
 Time.sleep(1500)  
 *return* }  
 TUI.writecenter("PIN ERROR",0)  
 changePass(user)  
 *return* }  
 *private fun* wrongPass(){  
 *return* pass()  
 }  
  
 *private fun* doorAction(worker:Ut){  
 LCD.clear()  
 *if*(worker.entryTime > 0L) awayDoor(worker)  
 *else* entryDoor(worker)  
 }  
  
 *private fun* moveDoor(){  
 Door.open(DOOR\_OPEN\_VELOCITY)  
 Time.sleep(3000)  
 Door.close(DOOR\_CLOSE\_VELOCITY)  
 }  
  
 *private fun* entryDoor(worker:Ut){  
 *val* entryTime = Time.getTimeInMillis()  
  
 TUI.writecenter("Welcome", 0)  
 TUI.writecenter(worker.name, 1)  
  
 moveDoor()  
  
 LogFile.entryUser(worker,entryTime)  
 Users.updateUser(worker.user,worker.acumulateTime,entryTime)  
 }  
  
 *private fun* awayDoor(worker: Ut){  
 *val* awayTime = Time.getTimeInMillis()  
 *val* acumulateTime = awayTime - worker.entryTime  
  
 TUI.writeleft(LogFile.calendarAway(worker.entryTime),0)  
 TUI.writeleft(LogFile.calendarAway(awayTime),1)  
 TUI.writeright(millisToHours(acumulateTime),1)  
  
 Time.sleep(3000)  
 LCD.clear()  
 TUI.writecenter("Good-Bye",0)  
 TUI.writecenter(worker.name,1)  
 moveDoor()  
  
 LogFile.awayUser(worker,awayTime)  
 Users.updateUser(worker.user,acumulateTime,0L)  
 }  
  
 *private fun* millisToHours(millis:Long):String{  
 *var* time = millis  
 *val* hour = (time/(60\*60\*1000))  
 time -= hour \* (60 \* 60 \* 1000)  
 *val* minutes = (time/(60\*1000))  
 time -= minutes\*(60\*1000)  
 *return* ("${hour}:${minutes}")  
 }  
  
 *private fun* restart(){  
 Time.sleep(1500)  
 LCD.clear()  
 mode()  
 }  
  
 *private fun* appPlay (){  
 LCD.clear()  
 TUI.writeright(TUI.time(),0)  
 pass()  
 restart()  
 }  
  
 *fun* mode(){  
 *if* (HAL.readBits(MMASK) != 0) Maintenance.init()  
 *else* appPlay()  
 }  
}  
  
*fun* main() {  
 HAL.init()  
 KBD.init()  
 LCD.init()  
 Users.init()  
  
 APP.mode()  
}