IPCA INSTITUTO POLITÉCNICO DO CÁVADO E DO AVE

Engenharia em Desenvolvimento de Jogos Digitais

Relatório

Computação Móvel

Paulo André Moreira Macedo 17011

Luis Manuel Gonçalves da Silva 17012

Barcelos, fevereiro 2020

Cofinanciado por: POCH PORTUGAL 2020 UNIÃO EUROPEIA Fundo Social Europeu

Instituto Politécnico do Cávado e do Ave

Engenharia em Desenvolvimento de Jogos Digitais

Relatório como requisito para a melhoria do projeto da Cadeira Computação Móvel do Curso de Engenharia em Desenvolvimento de Jogos Digitais, sob a orientação do docente Lourenço Gomes.

Barcelos, fevereiro 2020

Cofinanciado por:

POCH PORTUGAL 2020 UNIÃO EUROPEIA Fundo Social Europeu

Índice

Índice de Ilustrações

Introdução

Este relatório constitui a fundamentação teórica sofre o nosso projeto e pretende dar a conhecer o mesmo por nós desenvolvido.

O nosso trabalho final apresenta então as seguintes funcionalidades:

- Sistema de autenticação com Firebase Authentication

- Sistema de reset de password com envio de email para um associado a uma conta

- Sistema de verificacao de conta por email

- Criação de conta no ambiente da aplicação com upload de imagem de camera/galeria, username personalizavel, email e password

- "History" onde é permitido ler um pouco de uma história ficticia para dar contexto à aplicação

- "Account Settings" onde é possivel o utilizador alterar a password, email(com envio de email para o anterior a confirmar), username e imagem de perfil

- Criação de bouquets personalizados quer em nome quer em número de flores, com imagem dinâmica conforme o maior número de flores

- Visualização dos bouquets disponíveis e alteração/ deleção dos mesmos(exceto os predefinidos pela florista) e seleção dos quais pretende adquirir

- Uma activity de checkout que acede aos bouquets selecionados pelo utilizador e lhe permite alterar a quantidade de cada um mudando os valores respetivos(preço a pagar por

X quantidade de um certo bouquet e valor total final)

- Pagamento real usando paypal ou cartão de crédito associado à sua conta Paypal (Paypal API)

- Visualização do histórico de compras em detalhe com data e hora, quantidades de cada bouquet e preço de cada conjunto

Para tal foram utilizadas as seguintes funcionalidades da linguagem "Kotlin" :

- Modelo de dados com overloading de construtores e hierarquia

- Singleton com metodos para evitar repetição de código e organização com method overloading

- Classes para wrap de primitiveTypes (kotlin não permite referencia a primitiveTypes logo são necessários wrappers)

- Modelos de dados Serializable para serem transportados entre activities através de Intents

- Uso de Intents de vários tipos para o acesso a câmera, galeria e acesso a browser/app (instagram links)

- Uso de Dialogs para um aspeto mais profissional

- Utilização dos métodos e propriedades dos utilizadores da Firebase Authentication

- Utilização de Firebase Database para armazenamento dos dados por cliente (transações e bouquets disponíveis)

- Acréscimo, alteração e deleção desses mesmos dados em tempo real

- Utilização da Firebase Storage para o upload de imagens e sua associação ao perfil da Firebase Authentication

- Utilização de conversãoes entre Uri e bitmap (imagens proveninentes de galeria vêm em URI)

- Update do profile da Firebase Authentication(Username e imagem, através do Builder)

- Utilização de adapters costumizados para apresentação de listas e manuseamento dos "clicks" nos items/elementos de cada view

- Implementação e uso de livrarias externas(ExpandingList e Glide)

Foi ainda realizado, para um aspeto mais profissional, um total redesign da aplicação:

- UI simples e intuitivo

- História para contexto da aplicação

- Usos de dialogs e ExpandingList para melhor aspeto técnico

Desenvolvimento

Capítulo I – Modelo de Dados

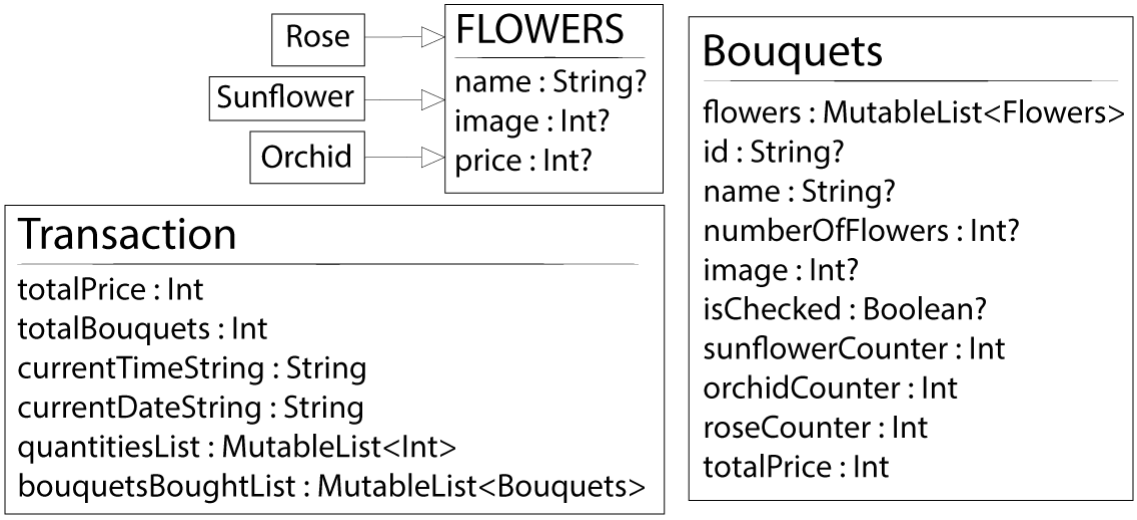


Figura 1 - Modelo de Dados

Os modelos de dados acima representados constituem a base na qual a aplicação se baseia.

A acrescentar, as Classes Bouquets e da hierarquia de flores herdam de Serializable visto que é necessário o passar destes objetos entre Activities.

Por fim, todas estas classes têm construtores vazios devido à leitura da FireBase.

Capítulo II – Esquema Geral (Lógica)

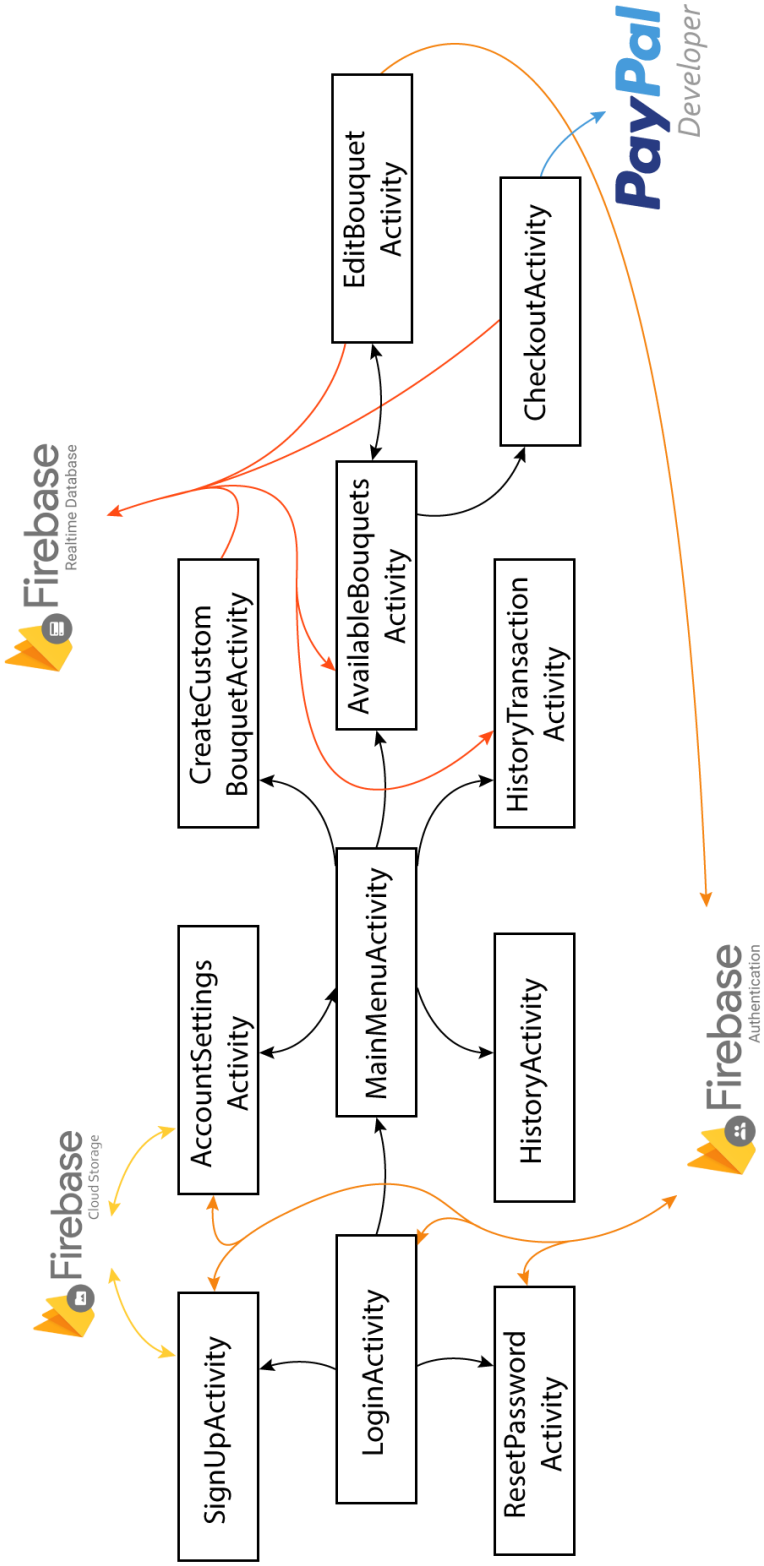


Figura 2 - Esquema Geral da App

Capítulo III –Livrarias Utilizadas

Glide

Documentação: https://github.com/bumptech/glide

Utilização: Utilizado para o manuseamento e load de URIs de imagens para imageViews

ExpandingViewLibrary

Documentação: https://github.com/diegodobelo/AndroidExpandingViewLibrary

Utilização: Utilizado para uma melhor estética na visualização do historio de transações.

Capítulo IV – Utilities

Para evitar a repetição de código e para uma melhor organização do manuseamento de imagens (updates em views e upload para a storage da Firebase) e a atualizacaoes de informações do perfil do utilizador foi criada um singleton 'ProfileAndImageManaging'

Class ProfileAndImageManaging

Métodos updateView

```
// Updates current Image Views
fun updateView(imageView : ImageView, bitmapImage : Bitmap){

    imageView.setImageBitmap(bitmapImage)

}

fun updateView(imageView : ImageView, uriImage : Uri, context : Context){

    Glide.with(context)
        .load(uriImage)
        .into(imageView)

}
//------------------------------
```

Figura 3 - ProfileAndImageManaing - updateView

Método overloading para a atualização de ImageViews através de Bitmap ou URI

Método imageStorageAndProfileUpdate

```
fun imageStorageAndProfileUpdate(bitmapImage : Bitmap, newUsername: String, referenceToUser : FirebaseAuth, context : Context, _callback : () -> Unit) {

    val baos = ByteArrayOutputStream()

    val storageRef = FirebaseStorage.getInstance().
        reference
        .child( pathString: "pics/${referenceToUser.currentUser!!.uid}")

    bitmapImage!!.compress(Bitmap.CompressFormat.PNG, quality: 100, baos)
    val image = baos.toByteArray()
    val upload = storageRef.putBytes(image)

    // Uploads image to Firebase Storage and gets its Uri to store in profile
    upload.addOnCompleteListener { uploadTask ->
        if (uploadTask.isSuccessful){
            storageRef.downloadUrl.addOnCompleteListener{ urlTask ->
                urlTask.result?.let{ it: Uri

                    Toast.makeText(context, text: "Image uploaded successfully", Toast.LENGTH_LONG).show()

                    // Updates profile with new data(including image Uri)
                    updateProfile(referenceToUser,newUsername,it, context,_callback)


                }
            }
        }else {
            uploadTask.exception!!.let{ it: Exception

                Toast.makeText(context, text: "Image Not Uploaded!", Toast.LENGTH_LONG).show()
            }
        }
    }
}
```

Figura 4 - ProfileAndImageManaging - imageStorageAndProfileUpdate

Método utilizado para o upload de imagens na FirebaseStorage e responsável por chamar um método para update do perfil com as novas informações.

Método updateProfile

```
private fun updateProfile(ref : FirebaseAuth, newUsername : String, imageUriToSave : Uri,context : Context, _callback :()-> Unit){

    val updates = UserProfileChangeRequest.Builder()
        .setDisplayName(newUsername)
        .setPhotoUri(imageUriToSave)
        .build()

    ref.currentUser!!.updateProfile(updates)
        ?.addOnCompleteListener{ task ->
            if (task.isSuccessful){

                Toast.makeText(context,  text: "Profile info Updated", Toast.LENGTH_LONG).show()

                _callback()

            } else {
                Toast.makeText(context, task.exception?.message!!, Toast.LENGTH_LONG).show()
            }

        }
}
```

Figura 5 - ProfileAndImageManaging - updateProfile

Método utilizado para o update das informações do perfil conforme recebidas e no final executa a função correspondente à referencia recebida.

Class FlowerSection

Devido à linguagem Kotlin não fornecer a alteração por Reference de variáveis de PrimitiveTypes foi necessário a criação de uma Classe que servisse para o armazenamento dos valores selecionados pelo utilizador no menu e para a criação da lista de Flores predefinidas (Disponíveis na loja).

```
// List for show which flowers are available
var allDifferentFlowerTypes: MutableList<Flowers> = ArrayList<Flowers>()

// Counters fo each type of flower selected
 var numberSunflowerSelected = 0
 var numberRoseSelected = 0
 var numberOrchidSelected = 0

constructor(){

    PredefinedListCreation()
}

constructor(bouquetReceivedForEdit : Bouquets){

    PredefinedListCreation()
    CountersDefinedByPreviousCreatedBouquet(bouquetReceivedForEdit)
}

private fun CountersDefinedByPreviousCreatedBouquet(bouquetReceivedForEdit : Bouquets){

    numberSunflowerSelected = bouquetReceivedForEdit.sunflowerCounter
    numberRoseSelected = bouquetReceivedForEdit.roseCounter
    numberOrchidSelected = bouquetReceivedForEdit.orchidCounter
}

// Creates List with all different type of flowers
private fun PredefinedListCreation(){

    allDifferentFlowerTypes.add(Sunflower())
    allDifferentFlowerTypes.add((Rose()))
    allDifferentFlowerTypes.add((Orchid()))
}
```

Figura 6 - FlowerSelection

Esta Classe serve-se de dois Construtores devido à necessidade dos valores na EditBouquetActivity terem de ser inicializados com valores recebidos (variáveis), enquanto que no CreateCustomBouquetActivity estes valores iniciam a zero.

Capítulo V – LoginActivity

Método onCreate & onStart

```
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_login)

    // Gets Firebase Authenticatiopn for login methods
    ref = FirebaseAuth.getInstance()

    // Manages button click
    buttonManager()
}

// Auto mail placement if there is any data of account in phone
override fun onStart() {
    super.onStart()

    // Checks for existing user
    val currentUser = ref.currentUser

    // If there is already data of a current account it writes the email automatically
    if (currentUser != null)
        emailView.text = Editable.Factory.getInstance().newEditable(currentUser.email)
}
```

Figura 7 - LoginActivity - onCreate & onStart

Metodo onCreate: Serve para a inicialização da referência à Firebase Authentication e por chamar a função responsável pelo manuseamento dos botões.

Metodo onStart: Serve para a introdução do email automático consoante o último Login efetuado

Método buttonManager

```
private fun buttonManager(){

    // Manages Sign Up button click
    SignUpButtonView.setOnClickListener{ it: View!
        startActivity(Intent( packageContext: this, SignUpActivity::class.java))
    }

    // Manages Reset Password button click
    ResetPasswordButtonView.setOnClickListener{ it: View!
        startActivity(Intent( packageContext: this, ResetPasswordActivity::class.java))
    }

    // Manages Login button click
    LoginButtonView.setOnClickListener{ it: View!
        login()
    }

}
```

Figura 8 - LogingActivity - buttonManager

Método responsável pelo manuseamento dos botões.

Método login

```
private fun login() {

    // Errors management-------------------------------//
    if (emailView.text.toString().isEmpty()){
        emailView.error = "Please Enter Email"
        emailView.requestFocus()
        return
    }

    if (!Patterns.EMAIL_ADDRESS.matcher(emailView.text.toString()).matches()){

        emailView.error = "Please Enter a Valid Email"
        emailView.requestFocus()
        return
    }

    if (passwordView.text.toString().isEmpty()){
        passwordView.error = "Please Enter Password"
        passwordView.requestFocus()
        return
    }
    //------------------------------------------------//

    // Checks if there is an account created with the corresponding email and password inserted
    ref.signInWithEmailAndPassword(emailView.text.toString(), passwordView.text.toString())
        .addOnCompleteListener(this) { task ->
            if (task.isSuccessful) {

                // Sign in success, update UI with the signed-in user's information
                val user = ref.currentUser
                updateUI(user)

            } else {
                // If sign in fails, display a message to the user.

                updateUI( currentUser: null)
            }
        }
}
```

Figura 9 - LoginActivity - login

Método responsável pela averiguação de erros tipo e pela verificação da autenticação com a Firebase Authentication.

Metodo updateUI

```
// According to result of matching password and email it takes action
private fun updateUI(currentUser : FirebaseUser?){

    // If there exists a current account associated it checks for the email verification before logging in
    if(currentUser!= null) {
        if(currentUser.isEmailVerified) {
            UserIdFirebase.UID = currentUser.uid

            startActivity(Intent( packageContext: this, MainMenuActivity::class.java))
        }
        else {
            Toast.makeText(baseContext,  text: "Email Not Verified.",
                Toast.LENGTH_SHORT).show()
        }
    }
    // If there is no currentUser it shows login error (not found user with current data inserted)
    else {
        Toast.makeText(baseContext,  text: "Wrong Email/Password. Try again",
            Toast.LENGTH_SHORT).show()

    }
}
```

Figura 10 - LoginActivity - updateUI

Meotdo responsável pela verificação da verificação por email e pelo começo de aplicação em si chamando a MainMenuActivity.

Capítulo VI – SignUpActivity

Método onCreate

```
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_signup)
    supportActionBar!!.hide()

    // Gets Firebase Auth Instance
    refToAcc = FirebaseAuth.getInstance()

    // Manages new account creation button click
    CreateAccountButtonView.setOnClickListener{ it: View!

        // Method responsible for account creation
        newAccountCreation()
    }

    // Manages Image click
    newAccountImageView.setOnClickListener{ it: View!

        // Method used for setting up confirmPasswordDialog for type of image choosing(camera or gallery)
        methodForImageChoosing()
    }
}
```

Figura 11 - SignUpActivity - onCreate

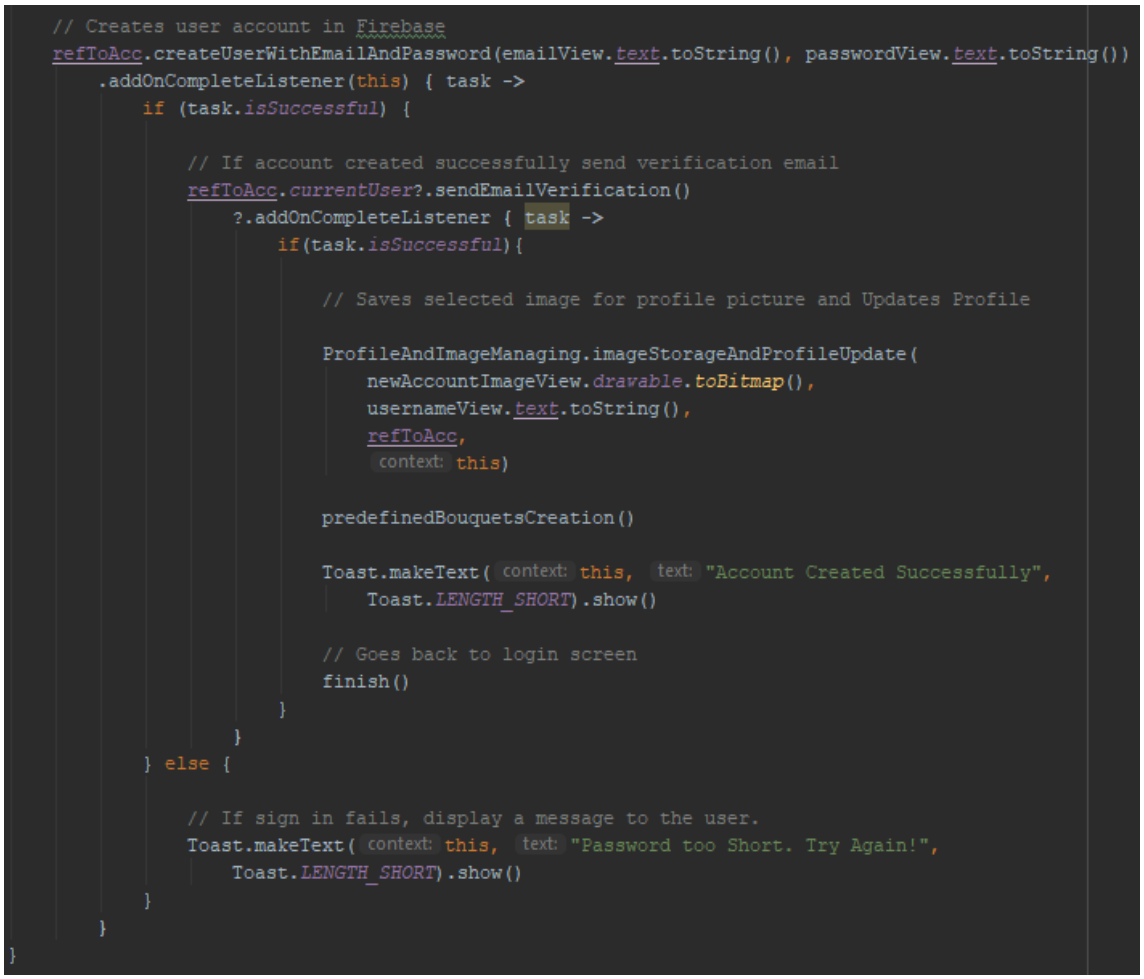
Método responsável inicialização da Firebase Authentication e manager dos botões.

Método newAccountCreation

```
private fun newAccountCreation(){

    // Errors Management--------------------------------------------------------//

    // Username Error
    if (usernameView.text.toString().isEmpty()){
        usernameView.error = "Username Required"
        usernameView.requestFocus()
        return
    }

    // Email Erros
    if (emailView.text.toString().isEmpty()){
        emailView.error = "Please Enter Email"
        emailView.requestFocus()
        return
    }
    if (!Patterns.EMAIL_ADDRESS.matcher(emailView.text.toString()).matches()){

        emailView.error = "Please Enter a Valid Email"
        emailView.requestFocus()
        return
    }

    // Password Errors
    if (passwordView.text.toString().isEmpty()){
        passwordView.error = "Please Enter Password"
        passwordView.requestFocus()
        return
    }

    if (confirmPassowrdView.text.toString() != passwordView.text.toString()){
        confirmPassowrdView.error = "Passwords do not Match"
        confirmPassowrdView.requestFocus()
        return
    }
    //-----------------------------------------------------------------------//
```

Figura 12 - SignUpActivity – newAccountCreation (1/2)

Parte do método responsável pelo manuseamento dos erros tipo da informação introduzida pelo utilizador.



```
    // Creates user account in Firebase
    refToAcc.createUserWithEmailAndPassword(emailView.text.toString(), passwordView.text.toString())
        .addOnCompleteListener(this) { task ->
            if (task.isSuccessful) {

                // If account created successfully send verification email
                refToAcc.currentUser?.sendEmailVerification()
                    ?.addOnCompleteListener { task ->
                        if(task.isSuccessful){

                            // Saves selected image for profile picture and Updates Profile

                            ProfileAndImageManaging.imageStorageAndProfileUpdate(
                                newAccountImageView.drawable.toBitmap(),
                                usernameView.text.toString(),
                                refToAcc,
                                 context: this)

                            predefinedBouquetsCreation()

                            Toast.makeText( context: this,  text: "Account Created Successfully",
                                Toast.LENGTH_SHORT).show()

                            // Goes back to login screen
                            finish()
                        }
                    }
            } else {

                // If sign in fails, display a message to the user.
                Toast.makeText( context: this,  text: "Password too Short. Try Again!",
                    Toast.LENGTH_SHORT).show()
            }
        }
}
```

Figura 13 - SignUpActivity - newAccountCreation (2/2)

Parte do método responsável pela criação da conta na Firebase com a informação inserida pelo utilizador.

Método predefinedBouquetsCreation

```
// Creates 3 predefined Bouquets that are stored in Firebase
private fun predefinedBouquetsCreation() {

    refToDatabase =  FirebaseDatabase.getInstance().getReference( path: refToAcc.currentUser!!.uid +"/Available Bouquets")

    var flowersListForPredefinedBouquet: MutableList<Flowers> = ArrayList<Flowers>()

    //First Bouquet- 100 sunflowers

    for(x in 0..99  ) flowersListForPredefinedBouquet.add(Sunflower())


    // gets id to store
    var idToSave = refToDatabase.push().key

    // Created predefined bouquet and stores it
    var bouquetToSave = Bouquets( name: "Shooting Star",flowersListForPredefinedBouquet,R.drawable.shootingstar)
    bouquetToSave.id = "PredefinedBouquet_1"

    refToDatabase.child(idToSave!!).
        setValue(bouquetToSave)
```

Figura 14 - SignUpActivity - predefinedBouquetsCreation

Método responsável pela criação de Bouquets predefinidos e aloca os mesmo na Firebase por conta. (Exemplo acima mostrado repete-se para os restantes)

Método methodForImageChoosing

```
private fun methodForImageChoosing(){

    // Sets up image type choosing View
    val bottomSheetDialog = BottomSheetDialog( context: this)
    val view = layoutInflater.inflate(R.layout.bottom_sheet_layout,  root: null)
    bottomSheetDialog.setContentView(view)
    bottomSheetDialog.show()

    // Manages type click
    view.cameraId.setOnClickListener{ it: View!
        pickImageFromCamera()
    }
    view.galleryId.setOnClickListener{ it: View!
        pickImageFromGallery()
    }
}
```

Figura 15 - SignUpActivity - methodForImageChoosing

Método responsável por criar e mostrar um Dialog com as opções para escolha de imagem (Câmera ou galeria)

Método pickImageFromCamera & pickImageFromGallery

```
private fun pickImageFromCamera(){

    // Intent used for taking picture from camera to be used
    Intent(MediaStore.ACTION_IMAGE_CAPTURE).also { pictureIntent ->
        pictureIntent.resolveActivity(this.packageManager!!)?.also { it: ComponentName
            startActivityForResult(pictureIntent, requestCode: 1)
        }
    }
}

private fun pickImageFromGallery(){

    // Intent used for picking gallery photo
    val intent = Intent()
    intent.type = "image/*"
    intent.action = Intent.ACTION_GET_CONTENT
   startActivityForResult(Intent.createChooser(intent, title: "Select Image"), requestCode: 2)
}
```

Figura 16 - SignUpActivity - pickImageFromCamera & pickImageFromGallery

Métodos responsáveis pela criação de intent conforme a Acão escolhida pelo utilizador.

Método onActivityResult

```
override fun onActivityResult(requestCode: Int, resultCode: Int, data: Intent?) {
    super.onActivityResult(requestCode, resultCode, data)

    // If photo is taken from camera successfully
    if (requestCode == 1 && resultCode == Activity.RESULT_OK)
    {
        // Gets bitmap image from camera
        var bitmapImage  = data?.extras?.get("data") as Bitmap

        // Updates ImageView
        ProfileAndImageManaging.updateView(newAccountImageView, bitmapImage)
    }

    // If photo is selected from gallery
     if(requestCode == 2 && resultCode == Activity.RESULT_OK)
     {
         // Converts Uri to bitmap
        val inputStream = contentResolver.openInputStream(data!!.data!!)
         var bitmapImage = BitmapFactory.decodeStream(inputStream)

         // Updates ImageView
         ProfileAndImageManaging.updateView(newAccountImageView, bitmapImage)
    }
}
```

Figura 17 - SignUpActivity - onActivityResult

Conforme o tipo de Acão escolhida a imagem é retornada em tipos diferentes (Bitmap pela câmera e URI pela galeria). Este método e então responsável pela conversão de URI para Bitmap no caso da galeria e pela atualização da ImageView com a imagem escolhida.

Capítulo VII – ResetPasswordActivity

Método onCreate

```
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_reset_password)
    supportActionBar!!.hide()

    // Reference to Auth Firebase
    ref = FirebaseAuth.getInstance()

    // Manages button to send reset password email
    ConfirmResetPasswordButtonView.setOnClickListener { it: View!
        resetPassword()
    }
}
```

Figura 18 - ResetPasswordActivity - onCreate

Método responsável pela inicialização da referência à Firebase Authentication e pelo manuseamento do botão.

Método resetPassword

```
private fun resetPassword() {

    // Error management--------------------------
    if (emailToResetView.text.toString().isEmpty()) {
        emailToResetView.error = "Please Enter Email"
        emailToResetView.requestFocus()
        return
    }
    if (!Patterns.EMAIL_ADDRESS.matcher(emailToResetView.text.toString()).matches()) {

        emailToResetView.error = "Please Enter a Valid Email"
        emailToResetView.requestFocus()
        return
    }
    // Error management--------------------------

    // Sends email to reset password (forgot)
    ref.sendPasswordResetEmail(emailToResetView.text.toString())
        .addOnCompleteListener(this) { task ->
            if (task.isSuccessful) {
                Toast.makeText(
                    context: this, text: "Email Sent Successfully",
                    Toast.LENGTH_SHORT
                ).show()
                finish()
            } else {
                Toast.makeText(
                    context: this, text: "No Account Associated with this Email",
                    Toast.LENGTH_SHORT
                ).show()
            }
        }
}
```

Figura 19 - ResetPasswordActivity - resetPassword

Metodo responsável pela verificação de erros tipo e caso o email inserido seja valido e tenha uma conta associada é mandado um email para o reset da password.

Capítulo VIII - MainMenuActivity

Método onCreate

```
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)

    setContentView(R.layout.activity_main_menu)
    supportActionBar!!.hide()

    // Gets instance of current player for data display
    ref = FirebaseAuth.getInstance()

    // Sets up initial view with data from profile
    initialViewSetup()

    // Manages Edit profile button click
    editButtonView.setOnClickListener{ it: View!

        confirmPasswordDialog()
    }

    // Manages the 4 main buttons for different options
    mainMenuButtonsManager()

    // Manages instagram clicks
    instagramButtonsManager()
}
```

Figura 20 - MainMenuActivity - onCreate

Metodo responsável pela inicialização da referência à Firebase Authentication, pelo chamar do método responsável pelo setup inicial das Views e manuseamento dos botões.

Método inicialViewSetup

```
private fun initialViewSetup(){

    // Updates avatar ImageView
    ProfileAndImageManaging.updateView(mainAvatarImageView,ref.currentUser!!.photoUrl!!, context: this)

    // Updates TextViews with profile info
    usernameTextView.text = ref.currentUser!!.displayName
    emailTextView.text = ref.currentUser!!.email
}
```

Figura 21 - MainMenuActivity - inicialViewSetup

Método responsável pelo setup inicial das views de acordo com as informações do perfil da Firebase Authentication.

Método confirmPasswordDialog

```
private fun confirmPasswordDialog(){

    var dialog = AlertDialog.Builder( context: this, AlertDialog.THEME_HOLO_DARK)
    val dialogView = layoutInflater.inflate(R.layout.dialog_password_check, root: null)
    dialog.setView(dialogView)
    dialog.setCancelable(true)

    dialog.show()

    dialogView.DialogPasswordButtonView.setOnClickListener{ it: View!

        if (dialogView.dialogPasswordView.text.toString().isEmpty()){
            dialogView.dialogPasswordView.error = "Please Enter Password"
            dialogView.dialogPasswordView.requestFocus()
        }
        else
        ref.signInWithEmailAndPassword(ref.currentUser!!.email.toString(), dialogView.dialogPasswordView.text.toString())
            .addOnCompleteListener(this) { task ->
                if (task.isSuccessful) {
                    startActivityForResult(Intent( packageContext: this, AccountSettingsActivity::class.java), requestCode: 1)
                }
                else {
                    dialogView.dialogPasswordView.error = "Wrong Password"
                    dialogView.dialogPasswordView.requestFocus()
                }
            }
    }
}
```

Figura 22 - MainMenuActivity - confirmPasswordDialog

Método responsável por gerar um Dialog que verifique se o utilizador é o mesmo utilizador da conta atual através da introdução da password, dando acesso às AccountSettings caso este se verifique.

Método mainMenuButtonsManager

```
private fun mainMenuButtonsManager(){

    historyButtonView.setOnClickListener{ it: View!

        var intent = Intent( packageContext: this, HistoryActivity::class.java)
        startActivity(intent)
    }

    availableBouquetsButtonView.setOnClickListener{ it: View!

        var intent = Intent( packageContext: this, AvailableBouquetsActivity::class.java)
        startActivity(intent)

    }

    createCustomBouquetButtonView.setOnClickListener{ it: View!

        var intent = Intent( packageContext: this, CreateCustomBouquetActivity::class.java)
        startActivity(intent)
    }

    transactionHistoryButtonView.setOnClickListener{ it: View!

        var intent = Intent ( packageContext: this, HistoryTransactionActivity::class.java)
        startActivity(intent)

    }
}
```

Figura 23 - MainMenuActivity - mainMenuButtonsManager

Método responsável pelo manuseamento dos botões principais da View,

Método instagramButtonsManager

```
private fun instagramButtonsManager(){
    // According to which insta selected it opens up for web or app view

    pauloInstaButtonView.setOnClickListener{ it: View!
        var url : String = "https://www.instagram.com/pauloamm2000/"
        var intent = Intent(Intent.ACTION_VIEW)
        intent.data = Uri.parse(url)
        startActivity(intent)
    }

    luisInstaButtonView.setOnClickListener{ it: View!
        var url : String = "https://www.instagram.com/luismsilva99/"
        var intent = Intent(Intent.ACTION_VIEW)
        intent.data = Uri.parse(url)
        startActivity(intent)
    }
}
```

Figura 24 - MainMenuActivity - instagramButtonsManager

Método responsável por abrir as páginas dos Instagram dos criadores da aplicação.

Método onActivityResult

```
override fun onActivityResult(requestCode: Int, resultCode: Int, data: Intent?) {
    super.onActivityResult(requestCode, resultCode, data)

    // Manages result from AccountSettingsActivity(updating views)
    if (requestCode == 1 && resultCode == Activity.RESULT_OK  )
    {
        var updateInformationType : String = data?.getStringExtra( name: "UpdateInformation")!!

        when (updateInformationType) {
            "UpdateProfile" ->{

                // Updates Image View according to the new photo stored in profile
                ProfileAndImageManaging.updateView(mainAvatarImageView, ref.currentUser!!.photoUrl!!,  context: this)

                // Updates main username view according to new username saved in profile in settings activity
                usernameTextView.text = ref.currentUser!!.displayName
            }

            // Updates Email TextView according to new email stored in account
            "UpdateEmail" -> emailTextView.text = ref.currentUser!!.email


            // After account deletion send you back to login screen
            "DeleteAccount" -> finish()
        }

    }
}
```

Figura 25 - MainMenuActivity - onActivityResult

Método responsável pelo update das Views com a informação alterada pelo utilizador na AccountSettings e pelo regresso ao ecrã de Login caso este tenha removido a sua conta.

Capítulo IX – AccountSettingsActivity

Método onCreate

```
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_account_settings)
    supportActionBar!!.hide()

    // Defaults result to cancelled in case the user decides to leave the activity without choosing an action
    setResult(Activity.RESULT_CANCELED)

    // Reference to Firebase Auth current user
    refAcc = FirebaseAuth.getInstance()

    // Sets up initial view (imageView and textViews)
    viewSetup()

    // Manages clicks of buttons
    buttonManager()
}
```

Figura 26 - AccountSettingsActivity - onCreate

Metodo responsável pela inicialização da referência da Firebase Authentication, da View e do manuseamento dos botões.

Método viewSetup

```
private fun viewSetup(){

    // Updates avatar ImageView
    ProfileAndImageManaging.updateView(avatarImageView,refAcc.currentUser!!.photoUrl!!, context: this)

    // Updates TextViews/EditTextViews with profile info
    usernameTextViewSettings.text = Editable.Factory.getInstance().newEditable(refAcc.currentUser!!.displayName)
    emailTextViewSettings.text = refAcc.currentUser!!.email
}
```

Figura 27 - AccountSettingsActivity - viewSetup

Método responsável pelo setup das views a partir das informações do utilizador da Firebase Authentication

Método buttonManager

```
private fun buttonManager(){

    // Manages image clicking for changing profile picture
    avatarImageView.setOnClickListener{ it: View!
        methodForImageChoosing()
    }

    // Manages Update profile button click
    updateProfileButtonManager()

    // Manages new email button click
    newEmailButtonView.setOnClickListener { it: View!
        newEmailAccount()
    }

    // Manages new password button click
    newPasswordButtonView.setOnClickListener{ it: View!
        newPasswordAccount()
    }

    // Manages delete account button click
    deleteAccountButtonView.setOnClickListener{ it: View!
        deleteAccount()
    }
}
```

Figura 28 - AccountSettings - buttonManager

Método responsável pelo manuseamento dos botões principais

Método updateProfileButtonManager

```
private fun updateProfileButtonManager(){

    profileUpdateButton.setOnClickListener{ it: View!

        // Gets Username written in edit text for update profile displayName
        var newUsername = usernameTextViewSettings.text.toString()

        if (newUsername.isEmpty()){
            usernameTextViewSettings.error = "Name Required"
            usernameTextViewSettings.requestFocus()
            return@setOnClickListener
        }

        // Only updates when written username is not empty
        var intent = Intent()
        intent.putExtra( name: "UpdateInformation", value: "UpdateProfile")

        setResult(Activity.RESULT_OK, intent)

       // Updates profile and finishes this activity once is all done
        ProfileAndImageManaging.imageStorageAndProfileUpdate(avatarImageView.drawable.toBitmap(),newUsername,refAcc,  context: this,{ finish() })

    }
}
```

Figura 29 - AccountSettingsActivity - updateProfileButtonManager

Método responsável pelo click do botão para o update das informações do perfil.

Métodos methodForImageChoosing, pickImageFromCamera, pickImageFromGallery & onActivityResult explicados na SignUpActivity.

Método newPasswordAccount

```
private fun newPasswordAccount() {

    // Error management-----------------------------
    if (newPasswordView.text.toString().isEmpty()) {
        newPasswordView.error = "Please Enter the New Password"
        newPasswordView.requestFocus()
        return
    }
    if (confirmNewPasswordView.text.toString().isEmpty()) {
        confirmNewPasswordView.error = "Please Confirm the New Password"
        confirmNewPasswordView.requestFocus()
        return
    }

    if (newPasswordView.text.toString() != confirmNewPasswordView.text.toString()) {
        confirmNewPasswordView.error = "Passwords do not Match"
        confirmNewPasswordView.requestFocus()
        return
    }
    // Error management-----------------------------

    // Updates password of current user
    refAcc.currentUser!!.updatePassword(newPasswordView.text.toString())
        .addOnCompleteListener(this) { task ->
            if (task.isSuccessful) {

                Toast.makeText(
                    context: this, text: "Password Changed Successfully",
                    Toast.LENGTH_SHORT
                ).show()

                finish()

            } else {
                // If fails, display a message to the user.
                Toast.makeText(
                    context: this, text: "An Error Occurred.",
                    Toast.LENGTH_SHORT
                ).show()
            }
        }
}
```

Figura 30 - AccountSettingsActivity - newPasswordAccount

Método responsável update da nova password inserida pelo utilizador. (método para a atualização do Email é semelhante à logica aqui representada)

Método deleteAccount

```
private fun deleteAccount(){

    // Removes the node from the Firebase of the selected account
    FirebaseDatabase.getInstance().getReference(refAcc.currentUser!!.uid).removeValue()

    // Removes profile image of current account
    FirebaseStorage.getInstance().reference.child( pathString: "pics/${refAcc.currentUser!!.uid}").delete()

    // Removes the Account
    refAcc.currentUser!!.delete()

    // Send intent with type of action chosen and sets RESULT_OK
    var intent = Intent()
    intent.putExtra( name: "UpdateInformation", value: "DeleteAccount")
    setResult(Activity.RESULT_OK, intent)
    finish()
}
```

Figura 31 - AccountSettingsActivity - deleteAccount

Método responsável pela remoção de toda a informação referente ao utilizador na Firebase Authentication, Database e Storage.

Capítulo X – HistoryActivity

```
class HistoryActivity : AppCompatActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_history)
        supportActionBar!!.hide()

        // Gets textView scrolling down
        historyTextView.movementMethod = ScrollingMovementMethod()

        // History pre-written
        historyTextView.text = "Nascida em 1926 com óbito em 1996, Dona Lurdes era" +
                " uma simples senhora da aldeia, com o típico coração alegre e simples!\n\n" +
                "Proveniente de família pobre, esta sobreviveu colhendo e vendendo" +
                " flores do quintal de seus pais, ficando conhecida na sua terra" +
                " como a casamenteira dos casais novos com todo o tipo de combinações " +
                "e especialidades de arranjos.\n\n" +
                "Em idade adulta decidiu então abrir uma florista que rapidamente" +
                " ganhou fama e sucesso devido ao ambiente acolhedor e simples.\n\n" +
                "Seus filhos então continuaram o legado de sua mãe mantendo esta " +
                "florista como um marco da sua terra, um marco histórico e que a todos derrete!\n\n" +
                "Assim, foi desenvolvida esta aplicação, não só para manter este" +
                "negócio em evolução nos tempos modernos mas também para dar a " +
                "conhecer a tão bela história que até hoje marca os corações de " +
                "quem conheceu a senhora e sua família!"
    }
}
```

Figura 32 - HistoryActivity

Classe responsável pelo display em modo Scroll com um texto predefinido sobre a história da Florista.

Capítulo XI – HistoryTransactionActivity

Método onCreate

```
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_history_transaction)
    supportActionBar!!.hide()

    ref = FirebaseDatabase.getInstance().getReference( path: UserIdFirebase.UID!! + "/Transaction History")

    mExpandingList = findViewById(R.id.expanding_list_main)

    readingDataFirebase()
}
```

Figura 33 - HistoryTransactionActivity - onCreate

Método responsável pela inicialização da referência da Firebase Database com o diretório correspondente, pela inicialização de uma variável expanding list da livraria usada e pelo método responsável pela leitura dos nodes na referência.

Método readingDataFirebase

```
private fun readingDataFirebase(){

    ref.addValueEventListener(object : ValueEventListener {
        override fun onCancelled(p0: DatabaseError) {
            TODO( reason: "not implemented") //To change body of created functions use File | Settings | File Templates.
        }
        override fun onDataChange(p0: DataSnapshot) {
            if(p0.exists()){
                for (h in p0.children){

                    // Gets current node Bouquet
                    var transactionInCurrentNode = h.getValue(Transaction::class.java)

                    transactionList.add(transactionInCurrentNode!!)

                    addItem(transactionInCurrentNode)
                }
            }

            if (transactionList.size == 0) bonus.visibility = View.VISIBLE
            else bonus.visibility = View.INVISIBLE
        }
    })
}
```

Figura 34 - HistoryTransactionActivity - readingDataFirebase

Método responsável pela leitura da informação alocada na Firebase Database, apenas mostrando a lista se existir algum objeto. (caso esteja vazia apresenta um ecrã de fundo diferente)

Método addItem

```
private fun addItem(currentTransaction : Transaction) {
    //Let's create an item with R.layout.expanding_layout
    val item = mExpandingList!!.createNewItem(R.layout.expanding_layout)

    //If item creation is successful, let's configure it
    if (item != null) {
        //It is possible to get any view inside the inflated layout. Let's set the text in the item

        // Sets up all information for viwew of transaction(principal items)

        item.titleTextView.text = "Total Price: " + currentTransaction.totalPrice.toString() + "€"
        item.descriptionTextView.text = "Total Bouquets: " + currentTransaction.totalBouquets.toString()

        item.dateTextView.text = currentTransaction.currentDateString
        item.timeTextView.text = currentTransaction.currentTimeString

        //We can create items in batch.
        item.createSubItems(currentTransaction.bouquetsBoughtList.size)
        for (i in 0 until item.subItemsCount) {

            //Let's get the created sub item by its index
            val view = item.getSubItemView(i)

            //Let's set some values in
            configureSubItem(view, currentTransaction.bouquetsBoughtList[i], currentTransaction.quantitiesList[i])
        }
    }
}
```

Figura 35 - HistoryTransactionActivity - addItem

Método responsável por adicionar os itens resgatados no método anterior à expanding list e cria os seus correspondentes sub_itens, chamando depois a função responsável pelo setup da View de cada um dos sub_itens.

Método configureSubItem

```
private fun configureSubItem(view: View, currentBouquet: Bouquets, currentBoquetQuantity : Int) {

    // Sets up each subItem view

    view.sub_titleTextView.text = currentBouquet.name

    //sets up flower count info on screen
    var flowersNumbers =
        "x" + currentBouquet.sunflowerCounter.toString() + " Sunflowers \n" +
                "x" +currentBouquet.roseCounter.toString() + " Roses \n" +
                "x" + currentBouquet.orchidCounter.toString() + " Orchids\n" +
                "Total: " +currentBouquet.numberOfFlowers.toString()

    view.sub_descriptionTextView.text = flowersNumbers

    view.sub_bouquetCounterTextView.text = "x" + currentBoquetQuantity.toString()

    view.sub_currentBouquetTotalPrice.text = "Price: " + currentBouquet.totalPrice * currentBoquetQuantity + "€"

    view.sub_currentBouquetImageView.setImageResource(currentBouquet.image!!)
}
```

Figura 36 - HistoryTransactionActivity - configureSubItem

Método responsável pelo setup da View de cada sub_item

Capitulo XII - CreateCustomBouquet

Método onCreate

```
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_create_custom_bouquet)
    getSupportActionBar()!!.setTitle("D.Lurdes");

    // Gets reference from correspondent node in firebase of Bouquet storage
    ref = FirebaseDatabase.getInstance().getReference( path: "Bouquets")

    // Sets up custom adapter
    allFlowerTypeView.adapter = FlowerTypeListAdapter()

    // Manages confirmButton action
    confirmButtonManagement()
}
```

Figura 37 - CreateCustomBouquetActivity - onCreate

Método responsável pelo chamar dos outros métodos essenciais para a Activity e pelo inicializar e definição a valores iniciais da mesma.

Método confirmButtonManagement

```
// Manages Confirm Button action
private fun confirmButtonManagement(){

    // Manages button click
    confirmAdd.setOnClickListener{ it: View!

        // Creates Bouquet from selected flowers
        var customBouquet = createCustomBouquet()

        // If there were flowers selected
       if(customBouquet.numberOfFlowers!! > 0){

            // Gets new id in the Firebase for the new created bouquet
            val bouquetId = ref.push().key
            customBouquet.id = bouquetId

            // Adds the new bouquet to the Firebase
            ref.child(bouquetId!!).setValue(customBouquet).addOnCompleteListener{ it: Task<Void!>

                // Makes pop up message confirming the save
                Toast.makeText( context: this, text: "Bouquet Saved!", Toast.LENGTH_LONG).show()
            }
        }
        else{

            // Makes pop up message telling there weren't flowers selected so the bouquet was not saved
            Toast.makeText( context: this, text: "No Flowers Selected\nBouquet Not Saved!", Toast.LENGTH_LONG).show()
        }
        // Closes current activity and return to main activity
        finish()
    }
}
```

Figura 38 - CreateCustomBouquetActivity - confirmButtonManagement

Método responsável pelo guardar o Bouquet recém-criado com as flores selecionadas pelo utilizar na Firebase com um ID único gerado pela mesma. Este, porém, só é guardado se existirem flores selecionadas.

Metodo createCustomBouquet

```
// Creates custom Bouquet from selected flowers
private fun createCustomBouquet() : Bouquets{

    // Creates temporary flower list for custom bouquet creation
    var flowerListForCustomBouquet : MutableList<Flowers> = ArrayList<Flowers>()

    for(x in 1..flowerSelectionManager.numberSunflowerSelected) flowerListForCustomBouquet.add(Sunflower())
    for(x in 1..flowerSelectionManager.numberRoseSelected) flowerListForCustomBouquet.add(Rose())
    for(x in 1..flowerSelectionManager.numberOrchidSelected) flowerListForCustomBouquet.add(Orchid())

    // Gets custom name if the user enters one, otherwise gives it a default name ("Custom Bouquet")
    var customName = when{

        customNameTextView.text.toString().isEmpty() -> "Custom Bouquet"
        else -> customNameTextView.text.toString()
    }
    return  Bouquets(customName, flowerListForCustomBouquet, imageChoosing())
}
```

Figura 39 - CreateCustomBouquetActivity - createCustomBouquet

Método responsável pela criação do Bouquet com as flores selecionadas pelo utilizador e com o nome customizável.

Método imageChoosing

```
private fun imageChoosing() : Int{

     var selectedImageforShow : Int

    // Priority list in case its equal number-> Venus - BloodyMary - Shooting Star

    if (flowerSelectionManager.numberOrchidSelected >= flowerSelectionManager.numberRoseSelected)
    {
        if (flowerSelectionManager.numberOrchidSelected >= flowerSelectionManager.numberSunflowerSelected)
        {
            selectedImageforShow = R.drawable.venus
        }
        else selectedImageforShow = R.drawable.shootingstar
    }
    else
    {
        if (flowerSelectionManager.numberRoseSelected >= flowerSelectionManager.numberSunflowerSelected)
        {
            selectedImageforShow = R.drawable.bloodymary
        }
        else selectedImageforShow = R.drawable.shootingstar
    }
    return selectedImageforShow
}
```

Figura 40 - CreateCustomBouquetActivity - imageChoosing

Método responsável pela gestão da imagem para o CustomBouquet dependendo da flor mais selecionada.

FlowerTypeListAdapter

```
inner class FlowerTypeListAdapter : BaseAdapter() {

    override fun getView(position: Int, convertView: View?, parente: ViewGroup?): View {

        var currentFlower : Flowers = getItem(position) as Flowers

        // gets view information
        var v = layoutInflater.inflate(R.layout.flowertype_row, parente,  attachToRoot: false)

        var textViewNome = v.findViewById<TextView>(R.id.flowerTypeNameView)
        textViewNome.text = currentFlower.name.toString()


        var flowerImageView = v.findViewById<ImageView>(R.id.flowerTypeImageView)
        flowerImageView.setImageResource( currentFlower.image!!)



        // Gets adding and removing flowers buttons
        var minusButtonView = v.findViewById<Button>(R.id.minusButton) as Button
        var plusButtonView = v.findViewById<Button>(R.id.plusButton) as Button


        // Gets current flower type number
        var currentFlowerTypeSelectionView =  v.findViewById(R.id.flowerTypeNumberSelection) as EditText


        var currentNumber = currentFlowerTypeSelectionView.text.toString().toInt()

        currentFlowerTypeNumberStoring(currentNumber,currentFlower)

        minusButtonView.setOnClickListener { it: View!

            var currentNumber = currentFlowerTypeSelectionView.text.toString().toInt()
            currentNumber--


            currentFlowerTypeSelectionView.text = Editable.Factory.getInstance().newEditable(currentNumber.toString())

            currentFlowerTypeNumberStoring(currentNumber,currentFlower)
        }

        plusButtonView.setOnClickListener { it: View!

            var currentNumber = currentFlowerTypeSelectionView.text.toString().toInt()

            currentNumber++

            currentFlowerTypeSelectionView.text =  Editable.Factory.getInstance().newEditable(currentNumber.toString())

            currentFlowerTypeNumberStoring(currentNumber,currentFlower)
        }
        return  v
    }

    private fun currentFlowerTypeNumberStoring (currentNumber : Int, currentFlower : Flowers) {

        when(currentFlower){

            is Sunflower -> {

                flowerSelectionManager.numberSunflowerSelected = currentNumber
            }
            is Rose ->{

                flowerSelectionManager.numberRoseSelected = currentNumber
            }
            is Orchid ->{

                flowerSelectionManager.numberOrchidSelected = currentNumber
            }
        }
    }
```

Figura 41 - CreateCustomBouquetActvity - FlowerTypeListAdapter

Adapter responsável pela apresentação da lista de flores com a sua respetiva seleção e alteração dos valores conforme o utilizador os altera.

Capítulo XIII - AvailableBouquets

Método onCreate

```
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_main)
    setSupportActionBar(toolbar)
    getSupportActionBar()!!.setTitle("D.Lurdes");

    // Gets reference from correspondent node in Firebase of Bouquet storage
    ref = FirebaseDatabase.getInstance().getReference( path: "Bouquets")

    // Created predefined bouquets(not stored in Firebase)
    predefinedBouquetsCreation()

    // Sets up adapter for the list
    bouquetListView.adapter = BouquetAdapter()

    // Reads custom bouquets from Firebase
    readingFirebaseData()

    // Calls and manages result from CreateCustomBouquetActivity
    addNewBouquetManager()

    // Manages the button for the CheckoutActivity
    checkoutManager()
}
```

Figura 42 - AvailableBouquets - onCreate

Método responsável pelo chamar dos outros métodos essenciais para a Activity e pelo inicializar e definição a valores iniciais da mesma.

Método readingFirebaseData

```
// Reads the data from the associated Firebase and stores them in the list
private fun readingFirebaseData(){

    ref.addValueEventListener(object : ValueEventListener{
        override fun onCancelled(p0: DatabaseError) {
            TODO( reason: "not implemented") //To change body of created functions use File | Settings | File Templates.
        }
        override fun onDataChange(p0: DataSnapshot) {
            if(p0.exists()){

                for (h in p0.children){

                    // Bool to check if the node is'nt already stored in the list
                    var alreadyInList : Boolean = false

                    // Gets current node Bouquet
                    var bouquetInCurrentNode = h.getValue(Bouquets::class.java)


                    // Checks the list for a bouquet with the same id
                    for(b in bouquetList){

                        // If it finds one it changes the bool variable to true
                        if(b.id != null && b.id == bouquetInCurrentNode!!.id ) {

                            alreadyInList = true
                            break
                        }
                    }

                    // If the bouquet in the current node of the firebase is'nt stored in the list it stores it
                    if(alreadyInList == false){

                        bouquetList.add(bouquetInCurrentNode!!)
                    }
                }
                // Updates listView
                bouquetListView.adapter = BouquetAdapter()
            }
        }
    })
}
```

Figura 43 - AvailableBouquets - readingFirebaseData

Método responsável pela leitura de dados da FireBase para armazenamento da lista, tendo sido utilizada uma condição que compare o id de cada node na FireBase à lista para a verificação se esta já se encontra guardado ou não.

Metodo addNewBouquetManager

```
// Manages the button for the CreateCustomBouquetActivity
 private fun addNewBouquetManager(){

     addNewBouquet.setOnClickListener { it: View!

         val intent = Intent( packageContext: this@MainActivity, CreateCustomBouquetActivity::class.java)

         startActivity(intent)
     }
 }
```

Figura 44 - AvailableBouquets - addNewBouquetManager

Método responsável pela gestão do botão responsável da inicialização da CreateNewBouquetActivity.

Metodo checkoutManager

```
// Manages the button for the CheckoutActivity
private fun checkoutManager(){

    checkoutButtonView.setOnClickListener{ it: View!
        val intent = Intent( packageContext: this@MainActivity, CheckoutActivity::class.java)

        var bouquetCounter : Int = 0

        for(checkBouquet in bouquetList){

            if(checkBouquet.isChecked == true){

                bouquetCounter++

                var bouquetKey = "BouquetNumber" + bouquetCounter

                intent.putExtra(bouquetKey, checkBouquet)
            }
        }

        intent.putExtra( name: "BouquetCounter", bouquetCounter)
        startActivity(intent)
    }
}
```

Figura 45 - AvailableBouquets - checkoutManager

Método responsável pela gestão do botão responsável pela inicialização da checkoutActivity, mandando para esta todos os elementos selecionados.

Bouquet Adapter

```
// Bouquet Adapter
inner class BouquetAdapter : BaseAdapter() {

    override fun getView(position: Int, convertView: View?, parente: ViewGroup?): View {

        // Gets current bouquet
        var currentBouquet : Bouquets = getItem(position) as Bouquets

        // gets view information
        var v = layoutInflater.inflate(R.layout.bouquet_row, parente, attachToRoot: false)

        var textViewNome = v.findViewById<TextView>(R.id.bouquetNameView) as TextView
        textViewNome.text = bouquetList[position].name

        var textViewFlowerCount = v.findViewById<TextView>(R.id.bouquetFlowerCountView) as TextView
        textViewFlowerCount.text = bouquetList[position].numberOfFlowers.toString()

        var imageViewBouquet = v.findViewById<ImageView>(R.id.bouquetImageView) as ImageView
        imageViewBouquet.setImageResource( bouquetList[position].image!!)

        var checkView = v.findViewById(R.id.checkBuyView) as CheckBox
        currentBouquet.UpdateCheck(checkView)

        var priceView = v.findViewById<TextView>(R.id.bouquetPriceView)
        priceView.text = currentBouquet.totalPrice.toString()

        var checkBox = v.findViewById<CheckBox>(R.id.checkBuyView)
        checkBox.setOnCheckedChangeListener(CompoundButton.OnCheckedChangeListener {
                compoundButton, b -> currentBouquet.UpdateCheck(checkBox)
        })

        //sets up flower count info on screen
        var flowersNumbers =
            "x" + currentBouquet.sunflowerCounter.toString() + " Sunflowers \n" +
                    "x" +currentBouquet.roseCounter.toString() + " Roses \n" +
                    "x" + currentBouquet.orchidCounter.toString() + " Orchids\n" +
                    "Total: " +currentBouquet.numberOfFlowers.toString()

        textViewFlowerCount.text = flowersNumbers

        // If the bouquet is not a predefined one it can Update or be deleted
        if(currentBouquet.id != null)
        v.setOnClickListener{ it: View!


            var intent = Intent( packageContext: this@MainActivity, EditBouquetActivity::class.java)

            intent.putExtra( name: "CurrentBouquet", getItem(position) as Bouquets)

            startActivityForResult(intent, requestCode: 1)
        }
        return v
    }

    override fun getItem(position: Int): Any {
        return bouquetList[position]
    }

    override fun getItemId(position: Int): Long {
        return 0
    }

    override fun getCount(): Int {
        return bouquetList.size
    }

}
```

Figura 46 - AvailableBouquets - BoquetAdapter

Adapter customizado para a apresentação da lista de Bouquets e responsável pela gestão do Click em elementos da lista (exceto os predefinidos). É este que inicia a EditBouquetActivity para a obtenção de resultados.

Método onActivityResult

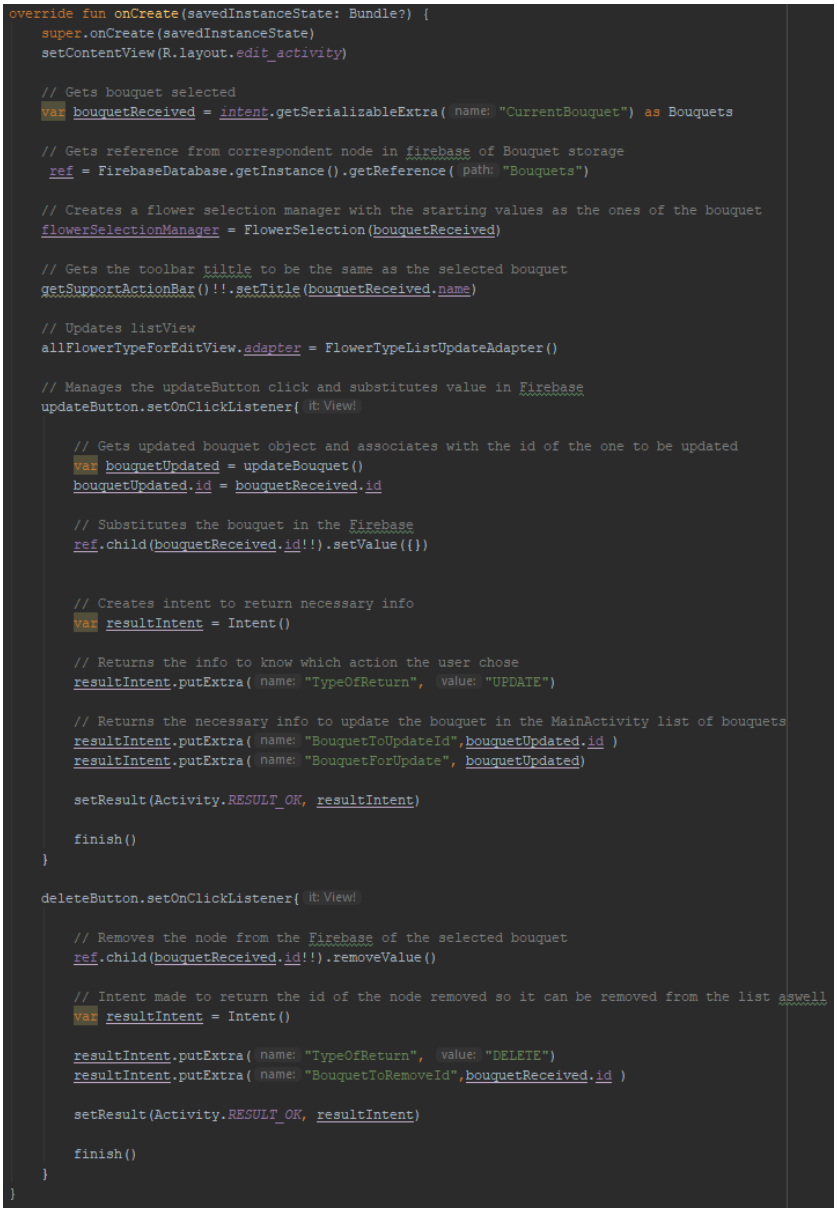
```
// Manages activity results from EditBouquetActivity
override fun onActivityResult(requestCode: Int, resultCode: Int, data: Intent?) {
    super.onActivityResult(requestCode, resultCode, data)

    // Result from EditBouquetActivity
     if(requestCode == 1 && resultCode == Activity.RESULT_OK){

        // Gets which action was made (Update)
        var typeOfReturn : String = data?.getStringExtra( name: "TypeOfReturn")!!

        // UPDATE ACTION
         if(typeOfReturn == "UPDATE"){

            // Gets data returned from the EditBouquetActivity

            // Gets the id of the bouquet to update
            var bouquetIdToUpdate = data?.getStringExtra( name: "BouquetToUpdateId")

            // Gets the updated bouquet
            var bouquetUpdated =  data?.getSerializableExtra( name: "BouquetForUpdate") as Bouquets

            // Variable responsible for storing the index of the bouquet in the list to update
            var indexToSubstitute : Int = 0

            for(b in bouquetList)
            {
                if(b.id == bouquetIdToUpdate) indexToSubstitute = bouquetList.indexOf(b)
            }

            // Substitutes the bouquet with the same id
            bouquetList[indexToSubstitute] = bouquetUpdated

            // Small message pop up to show it went sucessfully
            Toast.makeText( context: this, text: "Bouquet Updated", Toast.LENGTH_LONG).show()
        }

        // DELETE ACTION
        else if(typeOfReturn == "DELETE"){

             // Gets the data of the id of the bouquet to remove
            var bouquetIdToRemove = data?.getStringExtra( name: "BouquetToRemoveId")

             // Searches the bouquet list for the bouquet with the same id to be removed
            for(b in bouquetList){

                if(b.id == bouquetIdToRemove){

                    bouquetList.remove(b)
                    break
                }
            }

             // Makes small message pop up
             Toast.makeText( context: this, text: "Bouquet Deleted", Toast.LENGTH_LONG).show()
        }

        // Updates listView
        bouquetListView.adapter = BouquetAdapter()

    }
}
```

Figura 47 - AvailableBouquets - onActivityResult

Método responsável obtenção e manuseamento de dados dependendo dos retornados pela EditBouquetActivity.

Capítulo XIV - EditBouquetActivity

Método onCreate



```
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.edit_activity)

    // Gets bouquet selected
    var bouquetReceived = intent.getSerializableExtra( name: "CurrentBouquet") as Bouquets

    // Gets reference from correspondent node in firebase of Bouquet storage
     ref = FirebaseDatabase.getInstance().getReference( path: "Bouquets")

    // Creates a flower selection manager with the starting values as the ones of the bouquet
    flowerSelectionManager = FlowerSelection(bouquetReceived)

    // Gets the toolbar tiltle to be the same as the selected bouquet
    getSupportActionBar()!!.setTitle(bouquetReceived.name)

    // Updates listView
    allFlowerTypeForEditView.adapter = FlowerTypeListUpdateAdapter()

    // Manages the updateButton click and substitutes value in Firebase
    updateButton.setOnClickListener{ it: View!

        // Gets updated bouquet object and associates with the id of the one to be updated
        var bouquetUpdated = updateBouquet()
        bouquetUpdated.id = bouquetReceived.id

        // Substitutes the bouquet in the Firebase
        ref.child(bouquetReceived.id!!).setValue({})


        // Creates intent to return necessary info
        var resultIntent = Intent()

        // Returns the info to know which action the user chose
        resultIntent.putExtra( name: "TypeOfReturn",  value: "UPDATE")

        // Returns the necessary info to update the bouquet in the MainActivity list of bouquets
        resultIntent.putExtra( name: "BouquetToUpdateId",bouquetUpdated.id )
        resultIntent.putExtra( name: "BouquetForUpdate", bouquetUpdated)

        setResult(Activity.RESULT_OK, resultIntent)

        finish()
    }

    deleteButton.setOnClickListener{ it: View!

        // Removes the node from the Firebase of the selected bouquet
        ref.child(bouquetReceived.id!!).removeValue()

        // Intent made to return the id of the node removed so it can be removed from the list aswell
        var resultIntent = Intent()

        resultIntent.putExtra( name: "TypeOfReturn",  value: "DELETE")
        resultIntent.putExtra( name: "BouquetToRemoveId",bouquetReceived.id )

        setResult(Activity.RESULT_OK, resultIntent)

        finish()
    }
}
```

Figura 48 - EditBouquetActivity - onCreate

Método responsável pelo chamar dos outros métodos essenciais para a Activity e pelo inicializar e definição a valores iniciais da mesma. Além disso é também responsável pela gestão da ação que o utilizador pretende executar (Update ou deleção do Bouquet selecionado).

Método updateBouquet & imageChoosing

```
private fun updateBouquet() : Bouquets{

    //creates temporary flower list for custom bouquet creation
    var flowerListForCustomBouquet : MutableList<Flowers> = ArrayList<Flowers>()

    for(x in 1..flowerSelectionManager.numberSunflowerSelected) flowerListForCustomBouquet.add(Sunflower())
    for(x in 1..flowerSelectionManager.numberRoseSelected) flowerListForCustomBouquet.add(Rose())
    for(x in 1..flowerSelectionManager.numberOrchidSelected) flowerListForCustomBouquet.add(Orchid())

    return Bouquets( name: "Custom Bouquet", flowerListForCustomBouquet, imageChoosing())
}

private fun imageChoosing() : Int{

    var selectedImageforShow : Int

    // Priority list in case its equal number-> Venus - BloodyMary - Shooting Star

    if (flowerSelectionManager.numberOrchidSelected >= flowerSelectionManager.numberRoseSelected)
    {
        if (flowerSelectionManager.numberOrchidSelected >= flowerSelectionManager.numberSunflowerSelected)
        {
            selectedImageforShow = R.drawable.venus
        }
        else selectedImageforShow = R.drawable.shootingstar
    }
    else
    {
        if (flowerSelectionManager.numberRoseSelected >= flowerSelectionManager.numberSunflowerSelected)
        {
            selectedImageforShow = R.drawable.bloodymary
        }
        else selectedImageforShow = R.drawable.shootingstar
    }
    return selectedImageforShow
}
```

Figura 49 - EditBouquetActivity - updateBouquet & imageChoosing

Métodos de função semelhantes aos da página 34.

FlowerTypeListUpdateAdapter

```
inner class FlowerTypeListUpdateAdapter : BaseAdapter() {

    override fun getView(position: Int, convertView: View?, parente: ViewGroup?): View {

        var currentFlower : Flowers = getItem(position) as Flowers

        // gets view information
        var v = layoutInflater.inflate(R.layout.flowertype_row, parente,  attachToRoot: false)

        var textViewNome = v.findViewById<TextView>(R.id.flowerTypeNameView)
        textViewNome.text = currentFlower.name.toString()

        var flowerImageView = v.findViewById<ImageView>(R.id.flowerTypeImageView)
        flowerImageView.setImageResource( currentFlower.image!!)

        // Gets adding and removing flowers buttons
        var minusButtonView = v.findViewById<Button>(R.id.minusButton) as Button
        var plusButtonView = v.findViewById<Button>(R.id.plusButton) as Button

        // Gets current flower type number
        var currentFlowerTypeSelectionView =  v.findViewById(R.id.flowerTypeNumberSelection) as EditText

        when(currentFlower){

            is Sunflower -> currentFlowerTypeSelectionView.text = Editable.Factory.getInstance().newEditable(flowerSelectionManager.numberSunflowerSelected.toString())
            is Rose -> currentFlowerTypeSelectionView.text = Editable.Factory.getInstance().newEditable(flowerSelectionManager.numberRoseSelected.toString())
            is Orchid -> currentFlowerTypeSelectionView.text = Editable.Factory.getInstance().newEditable(flowerSelectionManager.numberOrchidSelected.toString())

        }

        var currentNumber = currentFlowerTypeSelectionView.text.toString().toInt()

        currentFlowerTypeNumberStoring(currentNumber,currentFlower)

        minusButtonView.setOnClickListener { it: View!

            var currentNumber = currentFlowerTypeSelectionView.text.toString().toInt()
            currentNumber--


            currentFlowerTypeSelectionView.text = Editable.Factory.getInstance().newEditable(currentNumber.toString())

            currentFlowerTypeNumberStoring(currentNumber,currentFlower)
        }

        plusButtonView.setOnClickListener { it: View!

            var currentNumber = currentFlowerTypeSelectionView.text.toString().toInt()

            currentNumber++


            currentFlowerTypeSelectionView.text =  Editable.Factory.getInstance().newEditable(currentNumber.toString())

            currentFlowerTypeNumberStoring(currentNumber,currentFlower)

        }
        return  v
    }

    private fun currentFlowerTypeNumberStoring (currentNumber : Int, currentFlower : Flowers) {

        when(currentFlower){

            is Sunflower -> {

                flowerSelectionManager.numberSunflowerSelected = currentNumber

            }
            is Rose ->{

                flowerSelectionManager.numberRoseSelected = currentNumber
            }
            is Orchid ->{

                flowerSelectionManager.numberOrchidSelected = currentNumber
            }
        }
    }

    override fun getItem(position: Int): Any {
        return flowerSelectionManager.allDifferentFlowerTypes[position]
    }
```

Figura 50 - EditBouquetActivity - FlowerTypeListUpdateAdapter

Adapter responsável pela apresentação da lista de flores com a sua respetiva seleção e alteração dos valores conforme o utilizador os altera (Neste caso os valores iniciam com os valores do Bouquet selecionado).

Capítulo XV - CheckoutActivity

Método onCreate

```
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_checkout)
    supportActionBar!!.hide()

    // Gets the number of selected bouquets
    var checkedBouquetCounter = intent.getIntExtra( name: "BouquetCounter", defaultValue: 0)

    for(x in 1.. checkedBouquetCounter){

        // Gets all bouquets sent through intent (the ones selected) and adds them to the list
        var bouquetKey = "BouquetNumber" + x
        var currentBouquet = intent.getSerializableExtra(bouquetKey) as Bouquets
        checkoutBouquetList.add(currentBouquet)
        bouquetsQuantity.add(1)
    }

    // Initial total price(1 of each bouquet)
    for(b in checkoutBouquetList){

        priceToPay += b.totalPrice

    }
    paypalAmountCheckout.text = "" + priceToPay + "€"

    config = PayPalConfiguration().environment(PayPalConfiguration.ENVIRONMENT_SANDBOX).clientId(
        UserInfo.client_id)
    var intent = Intent( packageContext: this, PayPalService::class.java)
    intent.putExtra(PayPalService.EXTRA_PAYPAL_CONFIGURATION,config)
    startService(intent)

    paypalButton.setOnClickListener{ it: View!

        amount = priceToPay.toDouble()

        var payment = PayPalPayment(BigDecimal.valueOf(amount),"EUR","DLurdes", PayPalPayment.PAYMENT_INTENT_SALE)
        var intent = Intent( packageContext: this, PaymentActivity::class.java)
        intent.putExtra(PayPalService.EXTRA_PAYPAL_CONFIGURATION, config)
        intent.putExtra(PaymentActivity.EXTRA_PAYMENT, payment)
        startActivityForResult(intent, requestCode: 1)
    }

    // Sets up custom adapter
    checkoutListView.adapter = CheckoutListAdapter()
}
```

Figura 51 - CheckoutActivity - onCreate

Método responsável pelo preenchimento das lista necessárias para a criação da transaction (valores provenientes do Intent) e configuração do botão Paypal com o valor resultante das quantidades de Bouquets decididos pelo utilizador.

Método totalPriceUpdate

```
private fun totalPriceUpdate(valueToRemove : Int, valueToAdd: Int){

    priceToPay -= valueToRemove
    priceToPay += valueToAdd
    paypalButton.text = "PayPal: " + priceToPay + "€"

}
```

Figura 52 - CheckoutActivity - totalPriceUpdate

Método responsável por dar update ao preço total a pagar quando são feitas alterações no número de Bouquets selecionados.

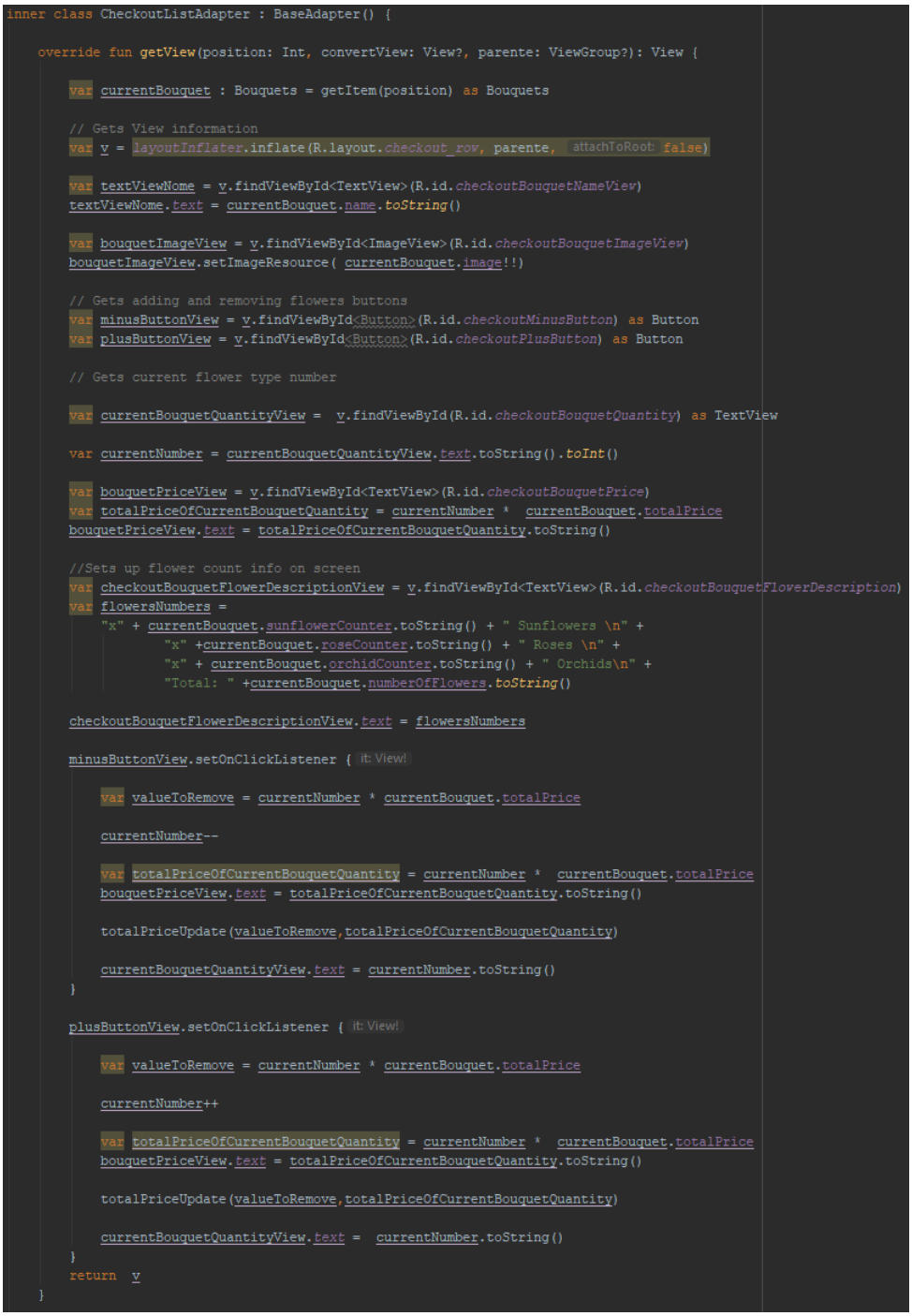
Metodo onActivityResult

```
override fun onActivityResult(requestCode: Int, resultCode: Int, data: Intent?) {
    super.onActivityResult(requestCode, resultCode, data)

    if (requestCode == 1){
        if (resultCode == Activity.RESULT_OK)
        {
            var refToSaveTransaction = FirebaseDatabase.getInstance().getReference( path: UserIdFirebase.UID!! + "/Transaction History")

            var newTransactionID = refToSaveTransaction.push().key
            var newTransaction = Transaction(bouquetsQuantity, checkoutBouquetList, priceToPay)

            refToSaveTransaction.child(newTransactionID!!).setValue(newTransaction)

            Toast.makeText( context: this, text: "Transaction Completed!", Toast.LENGTH_LONG).show()
            finish()
        }
        else
        {
            Toast.makeText( context: this, text: "Transaction Failed!", Toast.LENGTH_LONG).show()
            finish()
        }
    }
}
```

Figura 53 - CheckoutActivity - onActivityResult

Método responsável por comunicar ao utilizador se a transação foi bem ou malsucedida, e no caso de bem-sucedida é criada e guardada a transaction na Firebase Database.

CheckoutListAdapter



```
inner class CheckoutListAdapter : BaseAdapter() {

    override fun getView(position: Int, convertView: View?, parente: ViewGroup?): View {

        var currentBouquet : Bouquets = getItem(position) as Bouquets

        // Gets View information
        var v = layoutInflater.inflate(R.layout.checkout_row, parente,  attachToRoot: false)

        var textViewNome = v.findViewById<TextView>(R.id.checkoutBouquetNameView)
        textViewNome.text = currentBouquet.name.toString()

        var bouquetImageView = v.findViewById<ImageView>(R.id.checkoutBouquetImageView)
        bouquetImageView.setImageResource( currentBouquet.image!!)

        // Gets adding and removing flowers buttons
        var minusButtonView = v.findViewById<Button>(R.id.checkoutMinusButton) as Button
        var plusButtonView = v.findViewById<Button>(R.id.checkoutPlusButton) as Button

        // Gets current flower type number

        var currentBouquetQuantityView =  v.findViewById(R.id.checkoutBouquetQuantity) as TextView

        var currentNumber = currentBouquetQuantityView.text.toString().toInt()

        var bouquetPriceView = v.findViewById<TextView>(R.id.checkoutBouquetPrice)
        var totalPriceOfCurrentBouquetQuantity = currentNumber *  currentBouquet.totalPrice
        bouquetPriceView.text = totalPriceOfCurrentBouquetQuantity.toString()

        //Sets up flower count info on screen
        var checkoutBouquetFlowerDescriptionView = v.findViewById<TextView>(R.id.checkoutBouquetFlowerDescription)
        var flowersNumbers =
            "x" + currentBouquet.sunflowerCounter.toString() + " Sunflowers \n" +
                    "x" +currentBouquet.roseCounter.toString() + " Roses \n" +
                    "x" + currentBouquet.orchidCounter.toString() + " Orchids\n" +
                    "Total: " +currentBouquet.numberOfFlowers.toString()

        checkoutBouquetFlowerDescriptionView.text = flowersNumbers

        minusButtonView.setOnClickListener { it: View!

            var valueToRemove = currentNumber * currentBouquet.totalPrice

            currentNumber--

            var totalPriceOfCurrentBouquetQuantity = currentNumber *  currentBouquet.totalPrice
            bouquetPriceView.text = totalPriceOfCurrentBouquetQuantity.toString()

            totalPriceUpdate(valueToRemove,totalPriceOfCurrentBouquetQuantity)

            currentBouquetQuantityView.text = currentNumber.toString()
        }

        plusButtonView.setOnClickListener { it: View!

            var valueToRemove = currentNumber * currentBouquet.totalPrice

            currentNumber++

            var totalPriceOfCurrentBouquetQuantity = currentNumber *  currentBouquet.totalPrice
            bouquetPriceView.text = totalPriceOfCurrentBouquetQuantity.toString()

            totalPriceUpdate(valueToRemove,totalPriceOfCurrentBouquetQuantity)

            currentBouquetQuantityView.text =  currentNumber.toString()
        }
        return  v
    }
```

Figura 54 - CheckoutActivity - CheckoutListAdapter

Adapter responsável pela apresentação da lista de Bouquets a comprar com a sua respetiva seleção e alteração dos valores conforme o utilizador os altera, com os respetivos preços e o total a pagar pelo todo.

Capítulo XVI – UserInfo

```
class UserInfo
{
    //Paypal acc:
    //dlurdes@inwmail.net
    //Dlurdesflorista!1

    //bus-dlurdes@hotmail.com (business)
    //p-dlurdes@hotmail.com (client used for testing)
    //p2-dlurdes@hotmail.com
    //123456789

    //developer.paypal.com


    // Account where the money is send to
    companion object {
        var client_id :String = "Ab_pVm2LgBAGcKo4NILBTQ_cwroUGQi9J0o4evZ1XzfhcvinEraft6dhrQdFXtNaGQ6VeKGrK_IIEt9F"
    }

}
```

Figura 55 - UserInfo

Classe responsável pelo armazenamento da conta, a qual recebe pagamentos.

Capítulo XVII– Analise Critica

Esta análise crítica pretende fazer uma reflexão sobre o projeto. Tendo em conta os nossos objetivos no início do desenvolvimento do projeto, cumprimos e superamos todas as nossas metas.

Em suma, perante todo o trabalho aplicado e tendo em conta todos os pontos e funcionalidades já citadas anteriormente, o nosso grupo, Paulo Macedo 17011 e Luis Silva 17012, defende que o trabalho deverá merecer um 20!

Conclusão

O nosso relatório consistiu na realização do projeto, para conseguir concluir este projeto foi preciso ter uma grande capacidade de autonomia e persistência, aspetos esses que se foram desenvolvendo à medida que o concebíamos, planeávamos e executávamos.

A execução do projeto envolveu um grande esforço e dedicação. Para além das competências técnicas e diversas capacidades que o trabalho exigiu, pensamos que foi benéfico para nós a nível profissional e a nível social, uma vez que no futuro iremos enfrentar outros projetos tão ou mais importante que este.

Naturalmente que, no decorrer da realização do projeto sentimos diversas dificuldades, as quais só puderam ser ultrapassadas com grande esforço da nossa parte.

Web Grafia

General:

https://www.youtube.com/watch?v=F9UC9DY-vIU

Layout:

https://www.youtube.com/watch?v=AKvYOpbDYKA

Lists and Arrays:

https://www.youtube.com/watch?v=Je_YXshSFmY

https://www.youtube.com/watch?v=_SgmgA7Kz2g

ListView:

https://www.youtube.com/watch?v=EwwdQt3_fFU

https://www.youtube.com/watch?v=95QWxTZG_Z0

https://www.youtube.com/watch?v=BkmXPDY-1GE

https://www.youtube.com/watch?v=fwwu2mDD4cw

Intent:

https://www.youtube.com/watch?v=S1isQRnYAF4

https://www.youtube.com/watch?v=JjvY6bExiyg

https://www.youtube.com/watch?v=dcd6stWGObk

https://www.youtube.com/watch?v=JDxuBbsua_E

https://www.youtube.com/watch?v=2W41M9fWf6I

Object Keyword ("Static"):

https://www.youtube.com/watch?v=kH4pSAFSheU

https://www.youtube.com/watch?v=mK-0Zdjhcuk

Expandable List View:

https://www.youtube.com/watch?v=rBvqqy1kGjY&t

Firebase Database API:

https://www.youtube.com/watch?v=I485b7LzYkM

https://www.youtube.com/watch?v=ZB1liwuQCP8

https://www.youtube.com/watch?v=kc3LVeCDy14

Firebase Authentication & Storage:

https://www.youtube.com/watch?v=0hkyXuKTFYY

https://www.youtube.com/watch?v=MA8WbvROrLs&t

https://www.youtube.com/watch?v=-Kuhqq2ipXM

https://www.youtube.com/watch?v=j4F_EVl9Ja0

Paypal API:

https://www.youtube.com/watch?v=YO85GJoxkqE

https://www.youtube.com/watch?v=fYvXblhsjIg&t

https://www.youtube.com/watch?v=jrhPAiwDv1U