ISEL – Instituto Superior de Engenharia de Lisboa

ADEETC – Área Departamental de Engenharia de Electrónica e Telecomunicações e de Computadores

LEIM

Licenciatura em Engenharia Informática e Multimédia

Unidade Curricular de Projeto

Media Timer

Leonardo Rafael Calado de Oliveira (45088)

Carlos Miguel da Silva Rocha (45117)

Orientador

Professor Doutor António Teófilo

Setembro, 2021

Resumo

"Time is money", uma frase com origem numa redação escrita por Benjamin Franklin em 1748 mas ainda com mais valor nos dias de hoje. Os dias têm tempo finito por isso é importante como gerimos o nosso tempo, seja um engenheiro informático, um atleta, ou um cozinheiro, deve-se trabalhar de forma eficiente e eficaz.

É neste contexto que entra este projeto, *Media Timer*, uma aplicação mobile android que permite a criação, organização, customização, e execução de temporizadores. Uma maneira eficaz de gerir o tempo utilizado em tarefas através de temporizadores.

Aplicação esta destinada a utilizadores que simplesmente procuram temporizadores simples, ou para aqueles que procuram uma opção mais customizável.

Abstract

"Time is money", a phrase that originated in an essay written by Benjamin Franklin in 1748 but with even more value today. The days have finite time so it is important how we manage our time, whether you are a computer engineer, an athlete, or a cook, you should work efficiently and effectively.

It is in this context that this project comes in, *Media Timer*, a mobile android application that allows the creation, organization, and customization of countdown timers. An effective way to manage time used on tasks through timers.

This application is aimed at users who are simply looking for simple timers, or for those who are looking for a more found option.

Índice

Lista de Tabelas

Lista de Figuras

Capítulo 1

Introdução

Este capítulo irá expor o contexto em que este trabalho se insere bem como os seus objetivos a ser desenvolvidos.

1.1 Contexto

Atualmente os temporizadores são uma ferramenta útil para diversas atividades do dia-a-dia. Uma pessoa pode querer regular / cronometrar uma atividade que queira desempenhar, seja ela um ou vários exercícios físicos, passos de uma receita, ou outra que o utilizador queira restringir a um tempo definido. Com a nossa aplicação vai poder personalizar os temporizadores para uma experiência mais rica, na medida em que vamos oferecer mais opções de personalização do que a típica aplicação de temporizadores.

1.2 Problema e solução

Neste momento as aplicações que se encontram no mercado têm várias limitações, nomeadamente:

- Incapacidade de associar imagens aos temporizadores;
- Falta de opções para personalizar os temas dos temporizadores;
- Inabilidade de criar grupos de temporizadores;
- Criar temporizadores progressivos.

Associar imagens permite dar mais significado à tarefa a ser executada durante o temporizador e ajuda também a identifica-lo. A criação de grupos e personalizar as cores dos temporizadores oferece um grau maior de organização que é necessário para quem utiliza bastantes temporizadores. E por fim os temporizadores progressivos, temporizadores que vão mudando consoante a sua utilização permitem a criação de rotinas e promovem um uso continuo da aplicação. As aplicações no mercado oferecem uma ou outra das funcionalidades acima mas nunca todas. Com esta aplicação pretendemos preencher o espaço que existe e oferecer uma aplicação versátil.

1.3 Objetivos

De seguida definimos os objetivos que queríamos alcançar com este trabalho. O utilizador deve ser capaz de realizar as funções que definimos como básicas, estas sendo:

- Iniciar, duplicar, e apagar temporizadores;
- Colocar vários temporizadores a funcionar ao mesmo tempo;
- Mudar som do alarme.

A aplicação deve suportar um sistema que permita a criação de grupos de temporizadores. Estes grupos terão várias opções de configuração como, por exemplo, poder correr temporizadores sequencialmente (assim que um temporizador termina, começa o próximo).

Deve ser possível partilhar temporizadores ou grupos com outros utilizadores, bem como importar ditos temporizadores/grupos.

Outro objetivo é que se possa personalizar a aplicação (através de temas pré-definidos), e os temporizadores/grupos (através de cores escolhidas pelo utilizador).

Aos temporizadores deve ser possível associar ficheiros media como imagens ou vídeo. Estes ficheiros media são um ponto fulcral pois é algo que não se encontra noutras aplicações deste género.

Ainda os temporizadores devem ter algumas opções mais avançadas como:

- Criação de temporizações intermédias;

- Personalização das temporizações intermédias;
- Temporizadores progressivos.

As temporizações intermédias permitem sons tocar durante o decorrer do temporizador. Estas temporizações podem ser de X em X tempo, podem ocorrer um X de vezes, ou ocorrer um número de vezes aleatório. O utilizador pode escolher que som toca em cada temporização.

Os temporizadores progressivos são temporizadores com parâmetros que o utilizador pode ajustar ao seu parecer. Estes são temporizadores que podem aumentar ou diminuir consoante variáveis. Por exemplo, podemos querer um temporizador que após ter sido usado 3 dias consecutivos aumente 15 segundos. Deste modo pode-se criar rotinas que evoluem automaticamente.

Capítulo 2

Trabalho relacionado e Tecnologias utilizadas

Antes de iniciar o trabalho houve vários pontos a ser definidos.

2.1 Análise de requisitos

Foi realizada uma pesquisa para averiguar quais as aplicações que estavam no mercado com que a nossa aplicação vai competir. Logo desde o inicio que reparámos o facto de não existir uma grande variedade de aplicações para temporizadores, e as que existem, por norma ocupam um destes dois espaços:

1. Criação de temporizadores simples a pensar em curta duração (segundos/minutos);

2. Criação de temporizadores a que podem ser associados ficheiros media mas criados a pensar em longa duração (semanas/meses).

Para o 1^{o} caso as aplicações mais populares são *Multi timer StopWatch* ou *Countdown timer*, e para o 2^{o} são *Countdown timer - Event Countdown*, e *Timer Until*. Criámos uma tabela com as funcionalidades que considerámos principais das outras aplicações. Na tabela 2.1 apresentamos essas funcionalidades comparando com o que nós apontamos desenvolver.

Comparação entre aplicações				
Funcionalidade	Multi Timer	Event Countdown	Timer Until	Nossa aplicação
Criar/apagar T	✓	✓	✓	✓
Criar/apagar G	✓			✓
Vários T a correr ao mesmo tempo	✓	✓	✓	✓
Definir T favoritos	✓			✓
TI baseadas em tempo	✓			✓
TI baseadas em número				✓
TI aleatórias				✓
Associar imagens		✓	✓	✓
Associar vídeos				✓
Mudar cores dos T				✓
Criar T progressivos				✓

Tabela 2.1: Tabela a comparar funcionalidades das diversas aplicações.

T - Temporizador;
G - Grupo;
TI - Temporizadores intermédios.

As aplicações do 2º caso não permitem criação de grupos e estão mais direcionadas para eventos, como por exemplo um temporizador para aniversário. As do 1º não permitem media mas funcionam melhor para temporizadores curtos e em grande quantidade. O *Multi timer StopWatch* é a aplicação mais relevante para o nosso trabalho pois tem uma grande variedade de opções e é parecida com o nosso objetivo, no entanto esta não permite inclusão de media.

2.2 Tecnologias utilizadas

Foi também realizada uma análise para escolher que tecnologias utilizar como por exemplo, escolher a linguagem de programação e a base de dados.

2.2.1 Kotlin

Primeiramente foi escolhida a linguagem Java como base visto que esta era a linguagem oficial do Android e uma linguagem que utilizámos bastante durante a licenciatura. O trabalho foi sendo desenvolvido utilizando Java durante um período inicial mas chegámos à conclusão que era beneficiário mudar para a linguagem Kotlin.

Kotlin é uma linguagem baseada em Java relativamente recente, criada em 2011 e oficialmente lançada em 2016. Foi criada pela empresa JetBrains porque não existia nenhuma linguagem no mercado com as caraterísticas que a empresa cria, à exceção do Java mas o Java tinha problemas com o seu tempo de compilação.

O kotlin trás algumas vantagens:

- Tem uma sintaxe parecida com Java mas é mais simples e compacta;
- Acrescenta um conceito que é *Null Safety*, podemos definir variáveis que não podem ser *Null* o que elimina os erros muito comuns *NullPointerException*. Isto não significa que nunca podemos ter variáveis a *Null*, simplesmente oferece essa opção;
- Tem uma relação bidirecional com Java, pode-se inserir código Kotlin em projetos Java e vice-versa;

- Todas as API que executam no Java funcionam também em Kotlin, por isso Kotlin não tem falta de bibliotecas adicionais.

O uso de Kotlin tem ganho tração e é cada vez mais uma linguagem bastante usada. É utilizada por mais de 60% dos programadores profissionais de Android. Nós partimos para este projeto sem experiência em Kotlin mas tendo em conta que tem uma sintaxe simples e tem muitos recursos exteriores que ajudam a aprender, juntamente com as vantagens já ditas decidimos desenvolver esta aplicação em Kotlin.

2.2.2 Firebase

A aplicação tem uma vertente online pois deve ser possível partilhar temporizadores com outras pessoas. Deste modo decidimos que os utilizadores teriam de criar uma conta e os temporizadores são guardados numa base de dados. Deste modo decidimos olhar para o Firebase.

Firebase é uma plataforma desenvolvida pelo Google que oferece funcionalidades de autenticação e armazenamento de dados. Como foi desenvolvida para aplicações web e móveis, tem uma excelente interatividade com aplicações para Android. O firebase, de uma maneira muito simples, oferece as funcionalidades:

- Criação de contas;
- *Login/Logout*;
- Recuperação de palavra-passe;
- Armazenamento de dados.
- Permitir que a base de dados funcione sem conexão à internet.

Firebase ajudou também a aumentar a velocidade de desenvolvimento pois a sua base de dados não utiliza SQL mas sim guarda os dados em um formato parecido com JSON. Por isso não foi preciso realizar o processo rigoroso e complexo da criação do modelo da base de dados em SQL.

Vantagens do Firebase são:

- Permite utilizar a base de dados mesmo sem ligação à internet, armazenado uma versão em *cache* e quando o utilizador se voltar a ligar à internet é feita uma sincronização entre o que está em *cache* e o base de dados;
- É grátis mas permite pagar por uma verão melhor, e tem integração com o *Google Analytics*;
- Suporta uma variedade de linguagens como HTML, CSS, e JavaScript.

Tendo em conta a facilidade de usar e o grande número de funcionalidades que disponibiliza, optamos por utilizar o software firebase.

2.2.3 Ambiente de trabalho

Como ambiente de desenvolvimento escolhemos o Android Studio pois tínhamos experiência prévia e o Android Studio oferece integração com a Firebase e com a versão de controlo que utilizámos.

A versão de controlo usada foi GitHub que é um serviço que permite hospedar repositório Git. O Android Studio possibilita, por exemplo, observar os ramos do repositório e fazer *merge* sem utilizar comandos no terminal, permite ligar ao Firebase, e automaticamente adiciona todas as dependências necessárias para a sua utilização.

Criámos um repositório comum com um ramo *master* que tem a ultima versão e dois ramos em que trabalhamos.

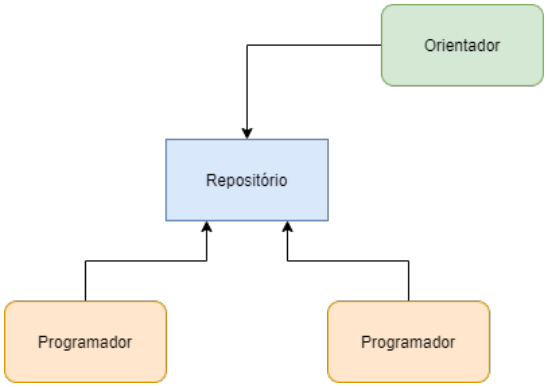


Figura 2.1: Modelo GitHub

Capítulo 3

Modelo proposto

Após a definição das tecnologias a utilizar e a análise de outras aplicações começámos por definir mais concretamente as funcionalidades da aplicação, os aspetos que a diferenciam das que foram observadas, o seu aspeto básico e como o utilizador poderá navegar entre páginas. Estabelecemos também como será organizada a base de dados, e o nosso modelo de monetização.

3.1 Análise de requisitos

Começámos por fazer uma tabela que os requisitos que na altura achámos obrigatórios a aplicação ter.

A aplicação não tem muitos processos a que o utilizador não tem acesso. Apenas a autenticação e o guardar media na base de dados são ações que o utilizador não tem contacto direto. Todos os outros são realizados, através de lógica básica da aplicação, após um input do utilizador.

Abaixo é apresentada a tabela 3.1 com ditos requisitos.

Requisitos		
Ref.	Funcionalidades	Categoria
R1.1	Registar utilizadores	Evidente
R1.2	Autenticar utilizador	Invisivel
R1.3	Criar um temporizador	Evidente
R1.4	Associar multimédia ao temporizador	Evidente
R1.5	Começar, parar e retomar o temporizador	Evidente
R1.6	Ver a multimédia associada ao temporizador	Evidente
R1.7	Guardar o temporizador	Invisivel
R1.8	Apagar o temporizador	Evidente
R1.9	Editar o temporizador	Evidente
R1.10	Partilhar o temporizador	Evidente
R1.11	Criar grupos	Evidente
R1.12	Editar grupos	Evidente
R1.13	Guardar o grupo	Invisivel
R1.14	Apagar o grupo	Evidente
R1.15	Partilhar o grupo	Evidente
R1.16	Suporte de várias temporizações ao mesmo tempo	Invisivel
R1.17	Escolher som do timer	Evidente
R1.18	Guardar media no servidor	Invisivel
R1.19	Parametrizar temporizações internas por tempo	Evidente
R1.20	Parametrizar temporizações internas por número	Evidente
R1.21	Parametrizar temporizações internas aleatórias	Evidente
R1.22	Escolher som das temporizações internas	Evidente
R1.23	Mudar tema da aplicação	Evidente
R1.24	Mudar cor das temporizações	Evidente
R1.25	Parametrizar temporizações progressivas	Evidente

Tabela 3.1: Tabela com os requisitos

O que destaca esta aplicação das outras, nomeadamente *Multi Timer StopWatch* (a aplicação mais completa atualmente), é a inclusão de media nos temporizadores, mudança de cores dos temporizadores/grupos, mais tipos de temporizadores internos, e os temporizadores progressivos.

A inclusão de media permite aos temporizadores mostrarem algo que o utilizador ache relevante ao temporizador, por exemplo se o utilizador estiver a fazer temporizadores para um receita culinária pode associar um vídeo que represente o que tem de se fazer.

Mudar a cores dos temporizadores/grupos possibilita o utilizador organizar os seus temporizadores/grupos, outras aplicações apenas permitem mudar o nome.

Por fim temos os temporizadores progressivos. Estes temporizadores alteram o seu tempo inicial consoante variáveis definidas pelo utilizador. O objetivo destes temporizadores é permiti-lo a criar "rotinas", por exemplo o utilizador cria um temporizador de 30 segundos para fazer um exercício físico e quer que após uma semana a fazer este exercício o temporizador aumente 15 segundos. Na nossa aplicação isso será possível e é algo que não se encontra noutras.

Estes são os requisitos que achámos necessários neste ponto, ao longo do trabalho e num futuro a seguir podem vir a ser adicionados mais funcionalidades.

3.2 Aspeto da aplicação (*Wireframe*)

Com os requisitos definidos começámos a definir o aspeto e a navegação que achámos pertinente a aplicação ter. Criámos um *Wireframe*.

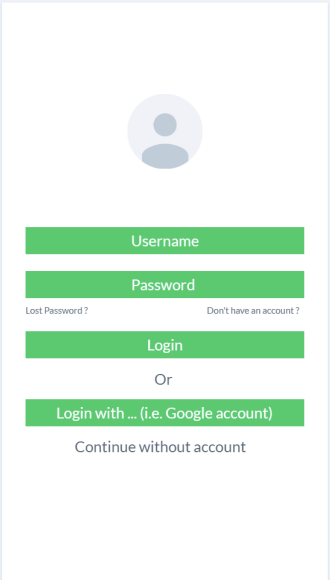


Figura 3.1: Fragmento Login

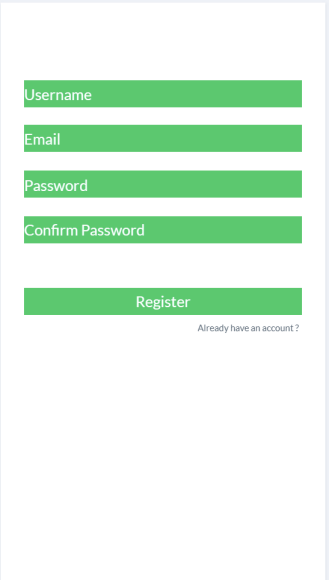


Figura 3.2: Fragmento Registo

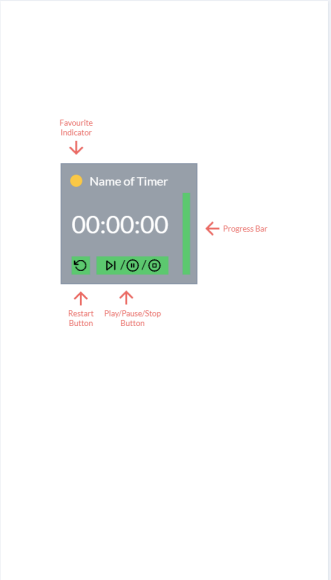
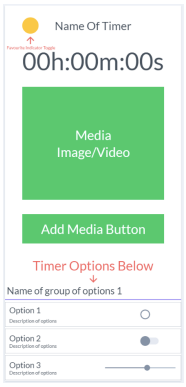
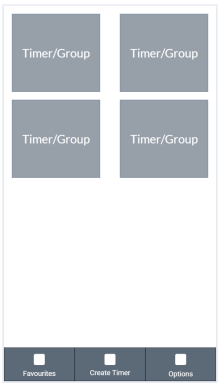


Figura 3.3: Fragmento Temporizador

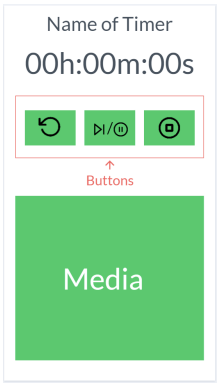


(a) Fragmento opções temporizador

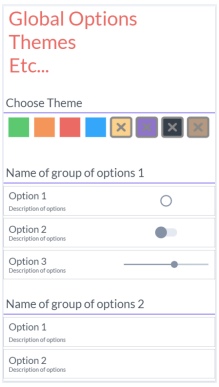


(b) Fragmento principal

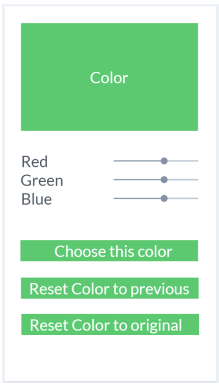
(c) Fragmento grupo temporizadores



(d) Fragmento ecrã inteiro



(e) Fragmento opções



(f) Fragmento escolha cor

Figura 3.4: Restantes fragmentos

O objetivo do *Wireframe* é criar uma base para as páginas de navegação sem nos preocuparmos com o seu aspeto. Mas desde o início que definimos que iríamos criar uma aplicação com um visual simples.

3.3 Organização da base de dados

Como já foi referenciado anteriormente (secção 2.2), a base de dados utilizada é da plataforma firebase. Das funcionalidades do firebase utilizamos:

- *Authentication*;
- *Realtime Database*;
- *Storage*.

O *authentication* permite guarda as contas dos utilizadores e a sua API já implementa todas as funcionalidades referentes ao *login*, registo, e recuperação da palavra-passe. Oferece também o *login* feito a partir de métodos externos como com a conta Google mas esta funcionalidade não foi utilizada.

Realtime Database é onde são guardados os temporizadores dos utilizadores. Esta base de dados é constantemente atualizada e pode ser utilizada sem conexão à internet. A informação é guardada num formato parecido com JSON.

Storage guarda no formato coleção e utilizamos para guardar ficheiros media.

Como já tínhamos experiência a usar a base de dados sabíamos que não poderíamos guardar os temporizadores de qualquer maneira pois estes iram ser objetos. Estes objetos têm de ser convertidos para JSON antes de serem guardados. Tem também de ser atribuído um ID única a cada objeto porque tinham de ser identificáveis. Para definir a que utilizador estes pertencem, é guardada uma lista de temporizadores dentro de uma entrada com um ID único que representa o utilizador.

Código 1: Exemplo do JSON usado

```
{"Timers": {
 id unico do timer: {
  nome do atributo 1 : atributo 1,
  nome do atributo 2 : atributo 2,
  ...
  }
 }
"Utilizadores": {
 id unico do utilizador: {
  "email" : email@xpto.pt,
  "nome" : nome do utilizador,
  "timers" : [id unico do timer 1, id unico do timer 2]
  }
 }

}
```

Este foi o JSON exemplo da base de dados. Temos uma entrada que é "timers"e contém todos os temporizadores, unicamente identificados, com os seus atributos e temos também uma entrada para os utilizadores com uma referencia para que temporizadores este tem.

Decidimos neste formato porque era necessário ter todos os temporizadores acessíveis para serem partilhados e se estivessem apenas dentro dos utilizadores

3.4 Modelo de monetização

Todas as aplicações que analisámos eram grátis, na loja Google Store é difícil encontrar aplicações deste género pagas o que já era de esperar. O modelo de monetização mais comum é anúncios.

Algumas aplicações usam também uma versão *premium* que remove os anúncios.A nossa ideia é de:

1. Anúncios;
2. Bloquear certas cores por detrás de versão *premium*;
3. Bloquear algumas funcionalidades dos grupos.

Gostaríamos de deixar a aplicação grátis porque não nos faz sentido ser paga, e a inclusão de anúncios é um dos meios mais populares de monetizar aplicações grátis.

As cores dos temporizadores são um ponto que não se encontra em outras aplicações por isso limitar certa cores por detrás de uma conta *premium*

pareceu-nos uma boa opção. Oferecer, por exemplo, 5 cores básicas e caso o utilizador deseje pode desbloquear o acesso a uma página onde este pode escolher o valor RGB que quer.

Nos grupos de temporizadores podem ser adicionados outros grupos permitindo assim um maior grau de organização, a profundidade que esta funcionalidade tem pode ser limitada, por exemplo para não deixar criar grupos dentro de grupos.

Capítulo 4

Arquitetura

Este capítulo expõe as nossas decisões relativas à arquitetura do trabalho.

4.1 Visão global

Utilizamos o modelo MVC, este modelo divide o desenvolvimento do trabalho em três camadas. Estas três camadas são o *Model*, *Controller*, e *View*.

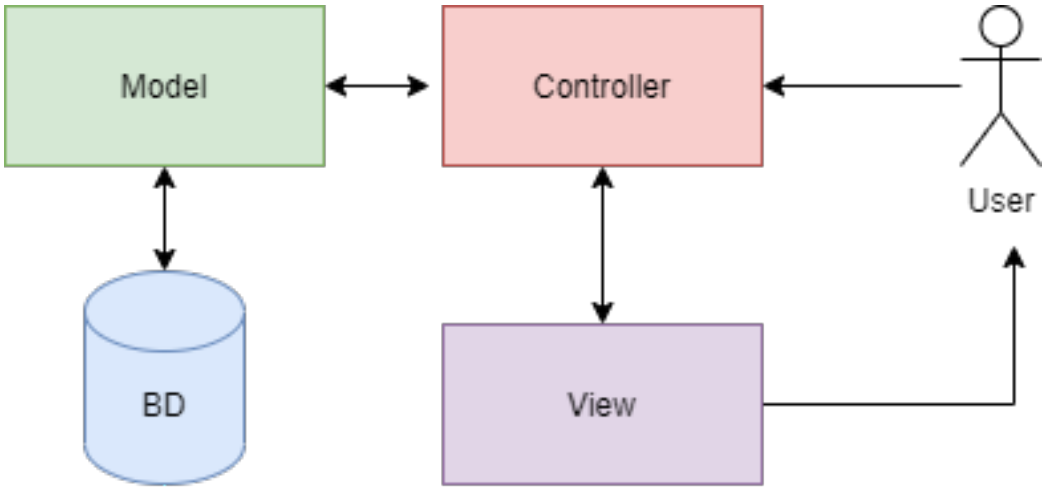


Figura 4.1: Arquitetura do modelo

4.1.1 Camada *View*

View é aquela que o utilizador observa. O Android funciona através de atividades e fragmentos como páginas de navegação.

O aspeto destas páginas é programado através de XML com *tags* explicitas do Android.

A navegação entre páginas é realizada com uma biblioteca fornecida pelo Android chamada "navigation component"e consiste em três partes:

- *navGraph*;
- *navHost*;
- *navController*.

O *navGraph* é um gráfico de navegação e contém toda a informação sobre a navegação. Contém referencias para todas as possíveis mudanças de página.

NavHost é um contentor para a página. Fica numa atividade e faz o *display* das páginas como fragmentos.

NavController é o objeto que controla o *navHost* e serve para, por exemplo saber quando é que se muda de página.

Este modelo de navegação oferece vantagens como:

- Lida com as ações para frente e para trás corretamente sem esforço adicional;
- Permite criar animações nas transições de uma maneira fácil;
- Consegue transportar data entre páginas.

4.1.2 Camada *Controller*

A camada Controller é a camada que liga o Model e a View, é a camada com o código que está dentro dos fragmentos. Cada fragmento tem uma classe associada e é essa classe que processa os *inputs* do utilizador.

4.1.3 Camada *Model*

A camada Model engloba as classes que contêm o bruto da lógica da aplicação, é responsável pelo acesso e manipulação dos dados. Nesta camada estão classes como "Timer", "TimerGroup"e "TimerCommon".

Os fragmentos não têm métodos

Capítulo 5

Implementação do modelo

5.1 Classe que contém temporizador ou grupo

Os grupos são basicamente uma lista de temporizadores por isso têm muitas parecenças com os temporizadores. Deste modo criámos uma classe "TimerCommon", esta classe irá conter um objeto da classe do temporizador "Timer"ou da classe do grupo "TimerGroup".

Código 2: Atributos da classe TimerCommon

```
var timer: Timer? = null
var group: TimerGroup? = null
var groupAssociated : TimerCommon? = null
var timerColors = TimerColors()
var active = true
var selecting = false
@Exclude lateinit var bigTextView: TextView
@Exclude lateinit var mediumTextView: TextView
@Exclude lateinit var smallTextView: TextView
@Exclude lateinit var nameTextView: TextView
@Exclude lateinit var progressBar: ProgressBar
@Exclude lateinit var context: Context
@Exclude lateinit var playPause:ImageButton
@Exclude lateinit var playPauseFullScreen:ImageButton
@Exclude lateinit var menu:ImageButton
var favourite = false
var id = ""
```

Estes atributos são comuns tanto a temporizadores como a grupos. Temos as views (objetos da interface) que têm de sofrer alterações durante o decorrer do temporizador (como por exemplo fazer update ao tempo que está a ser mostrado), existe a classe "TimerColors"que contêm as cores que o utilizador escolhe para o temporizador/grupo, um id único, e outros atributos. Esta classe tem também os métodos que alteram o valor destes atributos.

5.2 Criação dos temporizadores

Para os temporizadores utilizamos o objeto *CountDownTimer*, este objeto tem o método *onTick* que é chamado de X em X milissegundos, e tem outro método que é chamado quando o temporizador chega ao fim.

Código 3: Exemplo do objeto CountDownTimer

```
object : CountDownTimer(30000, 1000) {
 override fun onTick(millisUntilFinished: Long) {
  mTextField.setText("seconds remaining: " + millisUntilFinished / 1000)
 }

 override fun onFinish() {
  mTextField.setText("done!")
 }
}
```

Este objeto recebe dois argumentos, o primeiro é a duração do temporizador e o segundo é o número de milissegundos entre chamadas do método *onTick*. O objeto está na classe "Timer"juntamente com um os atributos necessários para todas as funcionalidades que implementámos.

5.3 Aspeto do temporizador ou grupo

Cada temporizador ou grupo é representado por um fragmento que fica contido num *recycler view*, este *recycler view* serve para podermos visualizar os temporizadores/grupos numa grelha.

Figura 5.1: Aspeto dos temporizadores

O temporizador tem um botão para marcar como favorito, tem os botões de parar, iniciar, e reiniciar, o seu nome, uma barra de progresso e um botão que abre um menu.

No centro fica o tempo pois este é o fator mais importante do temporizador, tivemos em conta também o tamanho dos botões para não oferecer uma experiência frustrante ao utilizador.

5.4 Ecrã inteiro

Para se focar totalmente no temporizador que quiser, disponibilizamos um ecrã com os botões para pausar e retomar o timer, nome e tempo, tal como no ecrã principal, mas também apresentamos a media que está associada ao timer em questão.

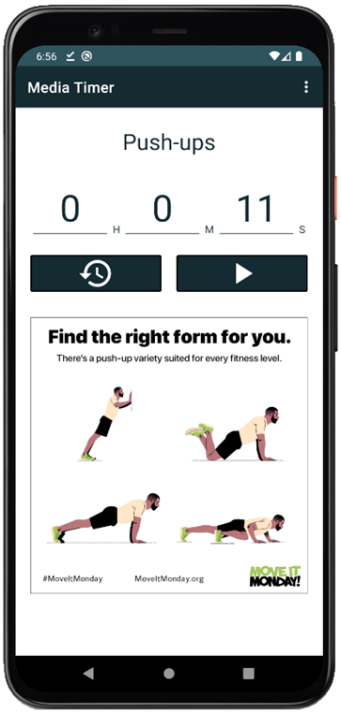


Figura 5.2: Fragmento ecrã inteiro

5.5 Guardar os temporizadores

Todos os temporizadores que o utilizador cria são guardados seu próprio telemóvel bem como na firebase, são guardados na firebase para efeitos de partilha (secção 5.6), e no próprio telemóvel para a aplicação ser usada sem problemas de conexão à internet. Com a ajuda da firebase e da autenticação, caso não seja apenas a primeira vez que o utilizador tenha instalada a aplicação, o utilizador pode voltar a ter acesso a todos os temporizadores que tinha antes de a desinstalar.

Nós criamos um objeto da classe "TimerToSave"que contém todos os atributos necessários para recriar o temporizador caso seja partilhado. Devido a limitações do JSON todos os atributos têm de ser primitivos ou uma lista de atributos primitivos.

Código 4: Atributos que são guardados

```
var type: String? = null
var name: String? = null
var initialTimerValue: Long? = null
var timerCountdownInterval: Int? = null
var mode: String? = null
var currentTimerValue: Long? = null
var intervalArrayNumber: ArrayList<Long>? = null
var intervalArrayTime: ArrayList<Long>? = null
var intervalArrayRandom: ArrayList<Long>? = null
var timerRunning: Boolean? = null
var timerCreated: Boolean? = null
var finished: Boolean? = null
var notifications: Notifications? = null
var sounds: ArrayList<String>? = null
var media: String? = null
var upload: String? = null
var initialTimerForProgressBar: Long? = null
var enableProgressiveTimer: Boolean? = null
var measurementIncrease: String? = null
var consecutiveIncrease: Int? = null
var secondsIncrease: Int? = null
var measurementDecrease: String? = null
var consecutiveDecrease: Int? = null
var secondsDecrease: Int? = null
var consecutiveIncreaseCounter: Int? = null
var consecutiveDecreaseCounter: Int? = null
var arrayDates: ArrayList<ArrayList<Int>>? = null
var arrayTimes: ArrayList<Long>? = null
var first: Boolean? = null
var alarmUri: String? = null
var timers: ArrayList<TimerToSave>? = null
var seq: Boolean? = null
var stopAtZero: Boolean? = null
var skipGroups: Boolean? = null
var currentRunning: Int? = null
var timerColors: TimerColors? = null
var id: String? = null
```

```
38 var canNotify: Boolean? = null
39 var arrayDatesHistory: ArrayList<ArrayList<Int>>? = null
40 var arrayTimesHistory: ArrayList<Long>? = null
41 var mainSoundVolume:Int? = null
42 var intervalsSoundVolume:Int? = null
43 var minProgressiveValue:Long? = null
44 var maxProgressiveValue:Long? = null
45 var typeOfAlarm:String? = null
46 var alarmRepetitions:Int? = null
47 var alarmTime:Long? = null
48 var favorite:Boolean = false
49 var backgroundSoundVolume:Int? = null
```

Nota: Ainda a pensar na partilha dos temporizadores, tivemos também de guardar toda a media que o utilizador associa a cada timer, uma vez que essa média provavelmente não existe no telemóvel do utilizador com quem foi partilhado o timer.

5.6 Partilha de temporizadores

Uma vez que, tal como foi dito no ponto sobre guardar os temporizadores (secção 5.6), todos os temporizadores são guardados na firebase, facilmente conseguimos ter acesso a todos os temporizadores em cada telemóvel que execute a nossa aplicação.

Nesta versão inicial da aplicação, cada timer tem guardado o id da sua entrada na base de dados, quando o utilizador decide partilhar o timer em questão, enviamos esse id da entrada na base de dados, e no ecrã para fazer o download do timer partilhado, o utilizador que recebeu o código pode inseri-lo passando assim a ter na sua aplicação uma cópia desse mesmo temporizador.

Idealmente em versões seguintes da aplicação, não seria partilhado esse id, mas sim de uma outra forma mais segura, a idealizar, onde não seja envolvido diretamente o identificador do timer na base de dados.

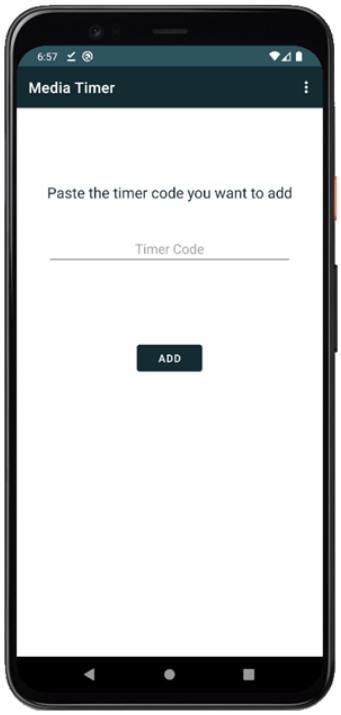


Figura 5.3: Partilha temporizador

Os atributos associados a esse id são obtidos da base de dados e é criado um objeto com esses atributos. Este novo objeto vai ter um id diferente do original, escolhemos esta opção porque se tiverem o mesmo id as mudanças

feitas por um utilizador seriam aplicadas ao outro, algo que não queríamos que acontecesse para não criar confusões. No entanto este é um conceito que no futuro poderíamos utilizar, temporizadores comuns a vários utilizadores e que quando um alterasse algo, os outros também veriam automaticamente.

5.7 Escolha das cores

Pode-se mudar as cores dos temporizadores/grupos, deste modo o utilizador consegue organizar como quer.

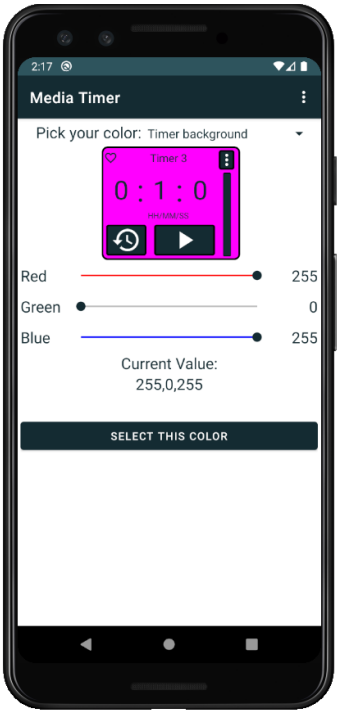


Figura 5.4: Fragmento escolha cores

Nesta página o utilizador pode escolher as cores:

- Do fundo do temporizador;
- Do texto;
- Do fundo dos botões;
- Do botão de favorito;
- Do primeiro plano dos botões.

O que quer alterar pode ser escolhido através de um menu suspenso, deste modo o utilizador sabe sempre exatamente que elemento está a mudar.

A escolha da cor é feita através de RGB e o utilizador pode observar as mudanças que esta a fazer em tempo real através do temporizador no meio

do ecrã. Quando tiver a cor que quer carrega no botão para a escolher, se não carregar nesse botão e sair da página ou selecionar outra opção do menu a cor é revertida à ultima que está guardada.

5.8 Notificações

Neste ecrã o utilizador pode escolher se quer definir intervalos, e de que maneira os que definir, seja um número de intervalos específico no decorrer do temporizador, intervalos de x em x tempo ou intervalos totalmente aleatórios. É também disponibilizada a opção de escolher tanto o som final do temporizador bem como os sons de cada um dos intervalos criados.

Finalmente o utilizador pode escolher se quer ser notificado com vibração ou não.

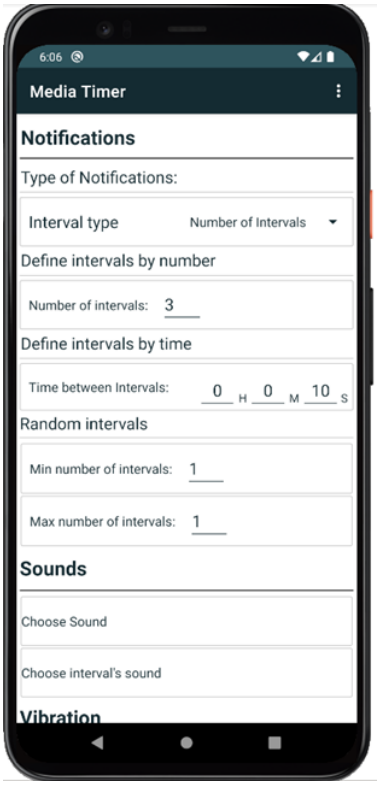


Figura 5.5: Fragmento notificações

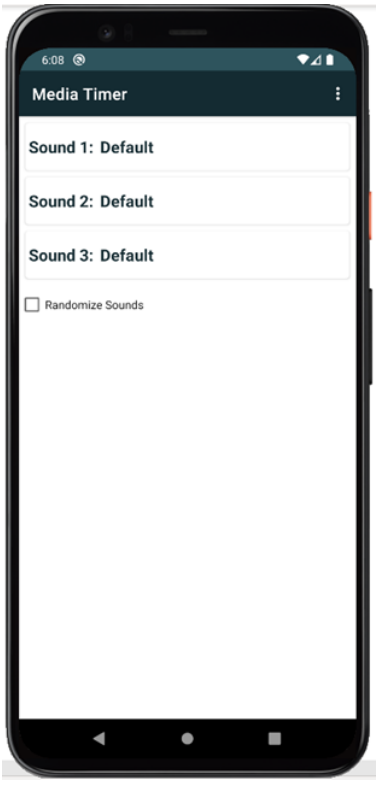


Figura 5.6: Fragmento sons

5.9 Página de personalização

Para além do ecrã principal onde o utilizador pode visualizar cada um dos seus temporizadores ou grupos, clicando em qualquer um dos temporizadores individuais, o utilizador é presenteado com a primeira página para personalizar o timer específico. Neste primeiro ecrã o utilizador pode alterar tanto o nome como o tempo inicial do temporizador (se o temporizador não estiver a correr), alterar a unidade de tempo em que o timer é apresentado, trocando entre Horas:Minutos:Segundos e Dias:Horas:Minutos, adicionar uma imagem, um vídeo, ou um som para correr como som de fundo do temporizador.

Para além das funcionalidades já enumeradas o utilizador vai poder escolher as Cores do Temporizador (secção 5.7), podendo destacar-se dos restantes temporizadores ou grupos e temporizadores dentro do grupo (se tiver dentro de um grupo). Vai poder configurar se e quando quer ter intervalos e os seus sons, nas Opções de Notificação (secção 5.8), e, por fim, se quer que conforme a frequência de utilização, o seu tempo inicial altere, na entrada Temporizador Progressivo (secção 5.10).

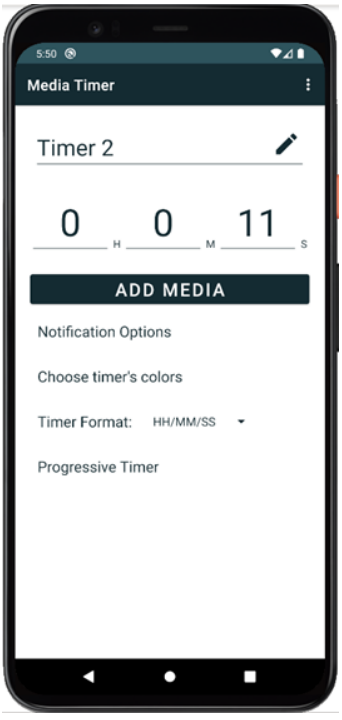


Figura 5.7: Opções temporizador

5.10 Temporizadores progressivos

Aqui o utilizador pode personalizar o timer de forma que se o utilizar numa determinada sequencia de dias, semanas ou meses, o tempo inicial do temporizador aumenta um número determinado de tempo definido por si. No sentido contrário, se utilizador estiver uma sequência determinada de dias, semanas ou meses sem utilizar o temporizador, o seu tempo inicial diminui num número determinado de segundos. Finalmente, o utilizador vai poder verificar como evoluiu o tempo inicial do temporizador através de um gráfico que demonstra todas as variações e a data em que elas aconteceram.

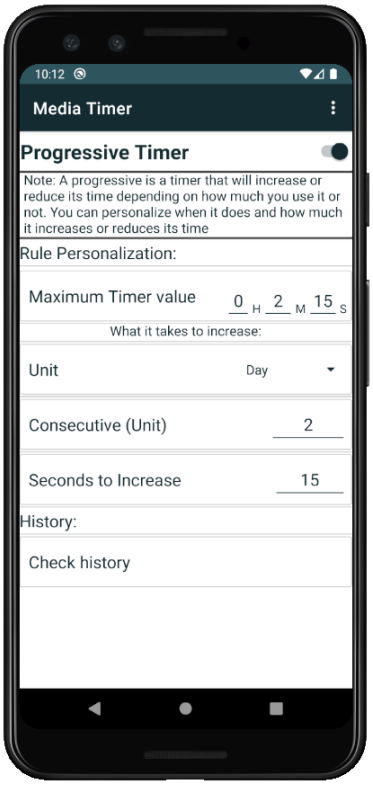


Figura 5.8: Fragmento Temporizadores progressivos

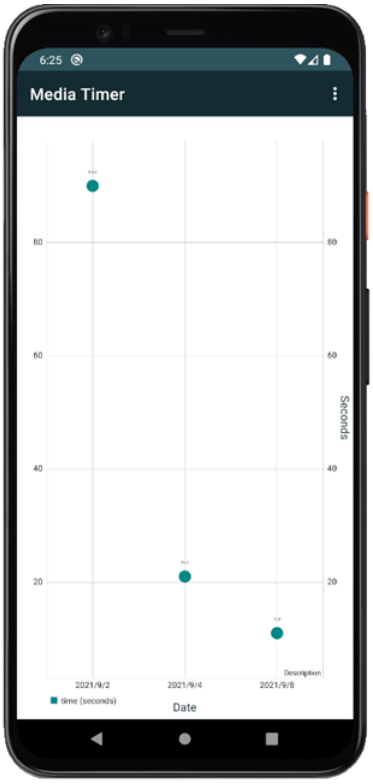


Figura 5.9: exemplo do gráfico do histórico

Capítulo 6

Testes

6.1 Sem internet

Obviamente, na sua versão mais básica (sem qualquer tipo de media) os *timers* funcionariam se o utilizador estiver ou não ligado à internet, visto que todos os *timers* são guardados não só na firebase mas também no próprio telemóvel. Os *timers* só são guardados na firebase para poderem ser facilmente partilhados, e para, caso o utilizador tenha desinstalado a app ou a tenha instalado noutro dispositivo, o utilizador ter acesso aos *timers* que tinha guardado. Mas, visto que oferecemos essa funcionalidade de partilha, neste teste podemos ver como se comporta a aplicação quando o utilizador tenta fazer upload de um timer.

Visto que não temos acesso à internet estamos obrigados a partilhar o timer por SMS uma vez que as restantes apps de troca de mensagens usam a internet. Após ter recebido esse código por SMS, seguimos o procedimento normal e colocamos o texto copiado da mensagem no campo do upload. Visto que a firebase guarda a última versão da base de dados no telemóvel (antes de perder o acesso á internet), se o temporizador tiver sido criado e guardado antes de perdermos esse acesso a partilha funciona perfeitamente. Caso o timer tenha sido criado e guardado depois de o utilizador ter tido acesso á internet, é impossível carregar o timer partilhado. Se o timer já existisse, mas tivesse sido alterado, vamos obter apenas a versão que estava na firebase antes de perdermos o acesso à internet.

Para além do processo de partilha, toda a media associada aos *timers* é guardada também na firebase, para, tal como já foi dito, o utilizador com

quem aquele timer for partilhado poder ver a media que foi associada ao mesmo. Finalmente, o utilizador pode também ter apagado essa media do telemóvel, e assim já não a poderia ver nos *timers*.

Tendo tomado esta opção, os utilizadores só poderão associar e ver a media nos temporizadores, quando tiverem online.

6.2 Questionário

Gostaríamos de ter realizado alguns testes com utilizadores para obter *feedback* sobre o aspeto e funcionalidade da nossa aplicação, mas infelizmente não os conseguimos organizar.

Capítulo 7

Conclusão

Este trabalho foi desenvolvido com o pressuposto de que existiria um esforço contínuo para trazer atualizações mesmo após a conclusão.

7.1 Conquista dos objetivos

Os objetivos principais estabelecidos na análise de requisitos foram alcançados.

É possível criar temporizadores/grupos, utilizá-los ao mesmo tempo e partilhá-los. Existem vários temas disponíveis para os utilizadores escolherem e é possível personalizar as cores dos temporizadores/grupos.

Consegue-se associar imagens e vídeos, e visualiza-los em ecrã inteiro. Pode-se escolher como se quer os intervalos intermédios e alterar os seus sons.

Os temporizadores progressivos também foram concretizados, incluindo a disponibilização de um gráfico que mostra o histórico da sua execução.

7.2 Conclusão geral

Com os objetivos que estabelecemos concretizados esta aplicação oferece algo que não se encontra no mercado da Google Store. As funcionalidades são diversas e a aplicação serve como ferramenta para um tipo muito variado de pessoas.

7.3 Trabalho futuro

Existem pontos que gostaríamos de fazer mas ainda não foram concretizados:

- Implementar o modelo de monetização;
- Criar mais opções de notificações, tais como:
  - Associar media a temporizações intermédias;
  - Aumentar/diminuir número de temporizações intermédias consoante o uso do temporizador;
- Poder pesquisar temporizadores criados por outras pessoas;
- Mais temas da aplicação;
- Adicionar *login* por métodos externos;
- Poder associar texto a temporizadores;
- Adicionar modo cronometro;
- Melhorar a interface.

Capítulo 8

Anexos

8.1 UML

A baixo segue a figura que representa o UML das nossas classes presentes no *model*:

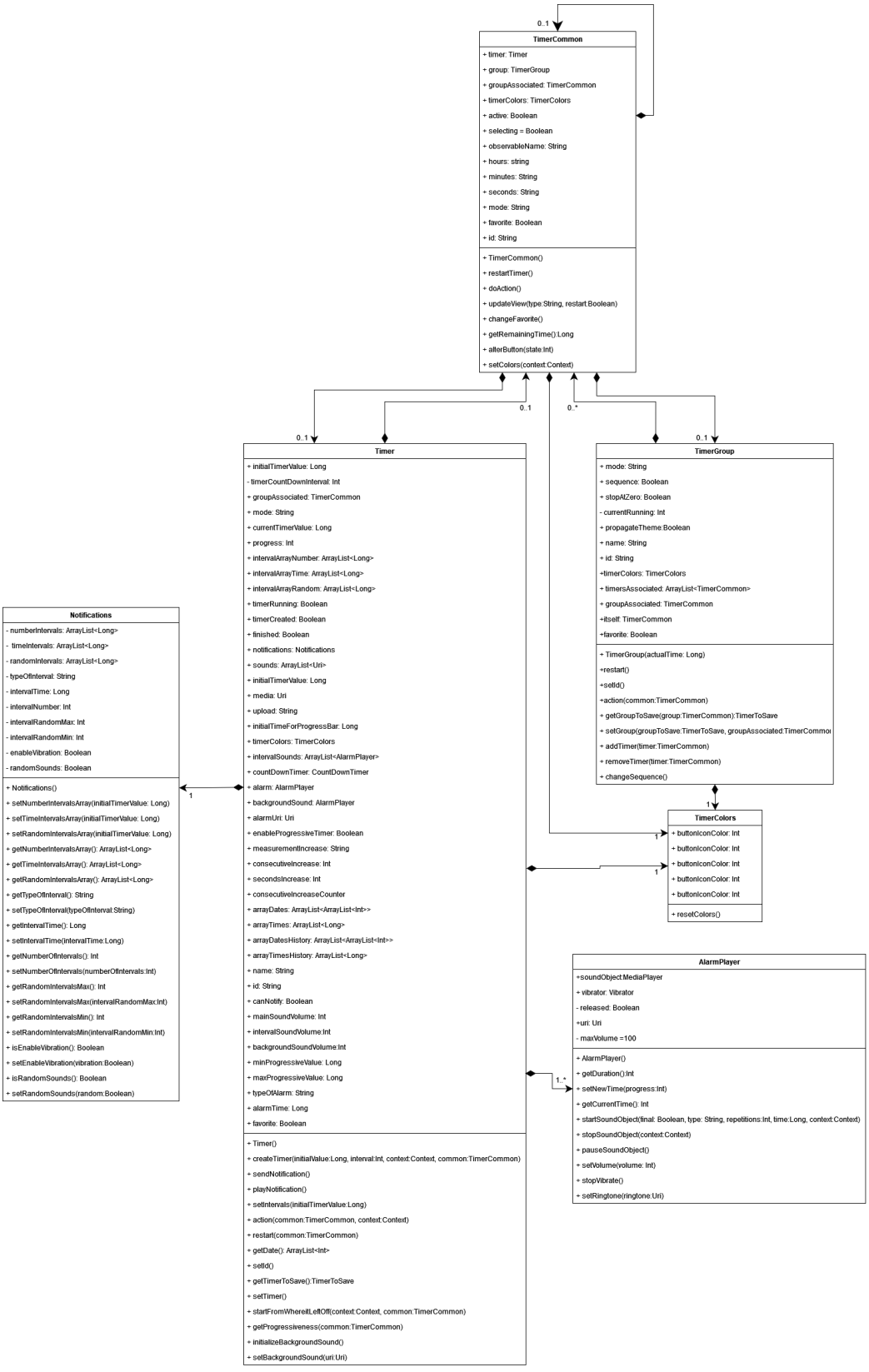


Figura 8.1: UML da camada Model

Bibliografia

[Google, 2021] Google (2021). Android studio 4.2.2. https://developer.android.com/studio/intro.

[Hopkins, 2013] Hopkins, C. (2013). Model view controller. *https://www.sitepoint.com/the-mvc-pattern-and-php-1/*.

[JetBrains, 2021] JetBrains (2021). Kotlin 1.5. https://kotlinlang.org/docs/home.html.

[JSON,] JSON. lightweight data-interchange forma. https://www.json.org/json-en.html.

[Marvel,] Marvel, A. Marvel app, application design website. https://marvelapp.com/.

[Oracle, 2021] Oracle (2021). Java 15.0.2. https://www.oracle.com/java/technologies/javase/15-0-2-relnotes.html.

[Tamplin e Lee,] Tamplin, J. e Lee, A. Firebase. https://firebase.google.com/docs.

[Tom Preston-Werner,] Tom Preston-Werner, Chris Wanstrath, P. J. H. S. C. Github, version control tool. https://github.com/.

[UML,] UML. Unified modeling language. https://pt.wikipedia.org/wiki/UML.